Universidade Estadual de Campinas – UNICAMP
Faculdade de Engenharia Elétrica e de Computação

Um Modelo Independente de Plataforma para a Execução de Processos de Negócios em Arquiteturas baseadas em Modelos

Autor: Cláudio Rodrigues Muniz da Silva
Orientador: Prof. Manuel de Jesus Mendes, Dr. Ing.

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Banca Examinadora

Prof. Dr. Luiz Eduardo Galvão Martins	UNIMEP
Prof. Dr. Cléver Ricardo Guareis de Farias	UNISANTOS
Prof. Dr. Manuel de Jesus Mendes	DCA/FEEC/UNICAMP
Prof. Dr. Ivan Luiz Marques Ricarte	DCA/FEEC/UNICAMP
Prof. Dr. Ricardo Ribeiro Gudwin	DCA/FEEC/UNICAMP

Julho de 2003

Campinas, SP – Brasil.

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

Si38m

Silva, Cláudio Rodrigues Muniz da
Um modelo independente de plataforma para a execução de processos de negócios em arquiteturas baseadas em modelos / Cláudio Rodrigues Muniz da Silva --Campinas, SP: [s.n.], 2003.

Orientador: Manuel de Jesus Mendes.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Engenharia de software. 2. Modelagem de dados. 3. UML (Linguagem de modelagem padrão). 4. Negócios. I. Mendes, Manuel de Jesus. II.Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

"Organizational progress parallels that in science and technology, permitting ultimate simplicity through intermediate complexity. "

Thomas Sowell

Dedicatória

Aos meus Pais, Márcio e Terezinha,

À minha esposa Iracema Augusta e

Ao meu filho Felipe.

Agradecimentos

Ao senhor meu Deus, pela graça da vida, esperança e luz que representa para a existência humana.

Aos meus pais, pelo apoio incondicional e constante em todos os momentos de minha vida, pelo exemplo de vida e fé que me tem inspirado desde a mais tenra idade e pelo amor e amizade que me têm sido dedicados.

A minha esposa, Iracema Augusta, pelo carinho, dedicação e paciência com relação aos momentos em que estive ausente durante a preparação deste trabalho, principalmente em sua etapa final.

Às Minhas Irmãs, Carla e Viviane, pelos momentos em que me apoiaram e incentivaram ao longo deste trabalho.

Ao Prof. Manuel Mendes, pela orientação, pela paciência e pela independência com a qual me permitiu realizar este trabalho.

Aos amigos André Coelho, Mário Ernesto, Radi, Diego, Alexandre e Vinícius, pelas discussões, sugestões e contribuições para este trabalho, além é claro dos momentos agradáveis e enriquecedores de nossa convivência no departamento de engenharia de computação e automação da UNICAMP.

Ao pessoal do Centro de Pesquisa Renato Archer pelo suporte e presença constante nas várias reuniões da fase final do trabalho.

À Universidade Federal do Rio Grande do Norte por permitir meu afastamento para doutoramento e à CAPES pela concessão da bolsa de estudos.

Resumo

O surgimento das plataformas de middleware no início da década de 80 foi uma solução para boa parte dos problemas oriundos da heterogeneidade de sistemas operacionais, protocolos de comunicação e linguagens de programação existentes na época. As plataformas de middleware permitiram a criação de um nível intermediário de abstração entre as aplicações dos usuários e os sistemas e protocolos que faziam parte de sua infra-estrutura de comunicação. A idéia era obter interoperabilidade através das interfaces utilizadas pelos serviços e seus usuários sem se preocupar com a forma como estas seriam implementadas. No entanto, nas últimas duas décadas, muitas foram as soluções de middleware que surgiram sem que nenhuma prevalecesse sobre as demais, cada uma com características próprias de programação e utilização, fazendo com que os problemas associados com a heterogeneidade voltassem a ser motivo de preocupação para grande parte dos desenvolvedores de sistemas de software.

Uma saída para esta situação aponta para a utilização de técnicas de modelagem e meta-modelagem que descrevem as aplicações de forma independente de suas plataformas de middlewares, separando os aspectos funcionais daqueles associados com as suas implementações. Estas soluções estão normalmente relacionadas com arquiteturas baseadas em modelos, entre as quais está a arquitetura MDA (*Model-driven Architecture*).

Apesar de alguns princípios da arquitetura MDA já terem sido considerados em diferentes projetos piloto, muitos aspectos ainda não estão devidamente amadurecidos e necessitam de mais estudos e novas provas de conceitos. Um destes aspectos é o processo de definição de modelos independentes de plataforma que sejam completos e adequados o suficiente para os domínios de serviço que venham a especificar. Um destes domínios diz respeito aos sistemas de gerência de workflows. Os sistemas de gerência de workflows são sistemas que automatizam a execução de processos de negócios e já foram objetos de padronização em uma série de organizações, entre as quais está a OMG. Na OMG, a especificação EDOC (em seu perfil UML para processos de negócios [OMG02b]) foi um primeiro passo importante para a criação de modelos independentes de plataformas que representam soluções baseadas nestes processos. No entanto, as primeiras implementações práticas mostraram a necessidade de se completar e estender os conceitos propostos na especificação EDOC no que se refere ao ambiente de execução necessário para os processos de negócios descritos [OMG02d].

Este trabalho propõe um modelo independente de plataforma para descrever ambientes de execução de processos de negócios, como uma forma de complementar a forma de descrever processos de negócios na arquitetura MDA.

Palavras Chaves : modelos, meta-modelos, perfis UML, arquiteturas baseadas em modelos (MDA), arquiteturas de software, meta-objetos, processos de negócios.

Abstract

The advent of the middleware platforms in the beginning of 80 decade was a solution to most of the problems associated with the heterogeneity of existing operating systems, communication protocols and programming languages. The middleware platforms compose an intermediary abstraction level between the user applications and the operating systems, aiming to achieve interoperability through the definition of interfaces between services and their client applications. Although, in these latest two decades, there were many proposed middleware solutions, each with its own characteristics to enable programming and utilization. However, none of them prevailed over the others and it seems that the problems associated with heterogeneity are back in a higher level of abstraction.

A possible solution to this situation points to the use of modeling and meta-modeling techniques to describe applications and systems in an independent way of their execution middleware platforms, separating their functional aspects from the implementational ones. This solution is used in model-driven architectures as the one specified in OMG as the Model-driven Architecture (MDA).

Although some of the principles of the MDA have already been validated by several proof-of-concepts, many other aspects are still in the process of maturation and need new studies and new proof-of–concepts. One of these aspects relates to the specification of convenient platform independent models for service domains they have to represent. Among these domains is the workflow management systems one. The workflow management systems automate the execution of business processes and have already been subject for standardization in several organizations. OMG is one of them and has contributed to the workflow domain in MDA with the UML profile for Business Process available in the EDOC specification. Unfortunately, the first implementations of business process solutions that were based in the EDOC specifications have raised some issues that point to the need to complete or extend EDOC with further standardization with the objective to describe the environment in which the process should be executed [OMG02d].

This work proposes a platform independent model to describe business process execution environment, as a way to complete and extend the business process based solutions in the context of the MDA architecture.

Keywords : model, meta-models, model-driven architecture (MDA), software architectures, meta-object facility (MOF), UML profiles, workflows, business process.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Acrônimos, Siglas e Abreviações

AGMC	*Ambiente de Gerência de Meta-componentes*
BPMS	*Business Process Management System*
BPRI	*Business Process Runtime Interfaces*
CCA	*Component Collaboration Architecture*
CCM	*CORBA Component Model*
CDIF	*Case Data Interchange Format*
CIM	*Computation Independent Business Model*
COM	*Component Object Model*
COM+	*Component Object Model Plus (Nova versão para o COM)*
CORBA	*Common Object Request Broker Architecture*
CWM	*Common Warehouse Metamodel*
CWMI	*Common Warehouse Metamodel Interchange Format*
CWMX	*Common Warehouse Metamodel Extensions*
DSTC	*Distributed Systems Technology Centre*
EAI	*Enterprise Application Integration*
ECA	*Enterprise Collaboration Architecture*
EDOC	*Enterprise Distributed Object Computing*
EDOC-BP	*Enterprise Distributed Object Computing – Business Process Profile*
EDOC-Eventos	*Enterprise Distributed Object Computing – Event Profile*
EIA	*Electronic Industries Association*
EJB	*Enterprise Java Beans*
JMI	*Java Metadata Interface*
MDA	*Model-driven Architecture*
MOF	*Meta-Object Facility*
OLAP	*Online Analytical Processing*
OMA	*Object Management Architecture*
OMG	*Object Management Group*
OMG-WF	*Modelo de Interfaces CORBA para Workflow Facility da OMG*
PDM	*Platform Description Model*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
RFP	*Request for Proposal*
RM-ODP	*Reference Model for Open Distributed Processing*
SPEM	*Software Process Engineering Metamodel*
UML	*Unified Modeling Language*
WfMC	*Workflow Management Coalition*
WfMS	*Workflow Management System*
XMI	*XML Metadata Interchange format*
XML	*Extensible Markup Language*

Capítulo 1
Introdução

As duas últimas décadas foram marcadas por sucessivas tentativas de se definir plataformas de softwares para aplicações distribuídas. Estas plataformas eram chamadas de *middlewares* em meados da década de 80, como uma forma de fazer referência ao nível intermediário abstrato definido entre os sistemas operacionais e suas aplicações distribuídas. No início eram constituídos basicamente por softwares de gerência de conexões de rede e o termo *middleware* ainda não era muito conhecido. No entanto, quando as tecnologias de rede atingiram maior penetração e larga visibilidade na década seguinte, o conceito de *middleware* passou a ser amplamente usado. Desde então, o que se entendia por *middleware* na época, evoluiu para um conjunto muito mais abrangente de paradigmas e serviços com o objetivo de ajudar a tornar mais fácil e gerenciável o processo de construção de aplicações distribuídas. O termo foi inicialmente associado com os bancos de dados relacionais. No entanto, alguns anos depois já era associado a conceitos similares aos *middlewares* de hoje, recebendo denominações tais como: sistemas operacionais de redes, sistemas operacionais distribuídos e ambientes de computação distribuída [GEI01].

Neste processo, a indústria de software introduziu, em média, uma nova plataforma de *middleware* a cada ano que passou. Cada uma destas plataformas prometia, de certa forma, substituir as existentes ao oferecerem maior simplicidade de uso ou universalidade de aplicação. Algumas delas eram voltadas para alguns nichos de rede, onde poderiam alegar algum tipo de vantagem sobre as demais soluções. No entanto, estas novas plataformas nunca cumpriram com a prometida substituição das demais. Desta forma, cada nova introdução, na verdade, aumentou o número de plataformas que as organizações deveriam suportar.

Como este processo continua até hoje e não há sinais de que esta corrida por novas plataformas esteja perto de seu fim, uma provável saída é a elevação do nível de abstração em que os sistemas são criados através da utilização do que chamamos de *arquiteturas baseadas em modelos*. No contexto deste trabalho, uma arquitetura de software é uma estrutura que compreende os componentes de software, suas propriedades externas visíveis e seus relacionamentos dentro desta estrutura [KAZ99]. As arquiteturas de software baseadas em modelos apresentam algumas peculiaridades que as tornam especialmente interessantes, pois

apresentam uma abordagem neutra e independente de fabricante, que utiliza modelos e meta-modelos para descrever aspectos estruturais e seus impactos em questões relacionadas com interoperabilidade, ao invés de dar ênfase, simplesmente, às interfaces e seqüências de interações específicas dentro de uma plataforma previamente determinada. O problema é particularmente relevante hoje em dia, em virtude das mudanças ocasionadas pelo surgimento da arquitetura MDA (*Model-driven Architecture*) [WAD02].

A MDA é uma iniciativa da OMG e faz parte do direcionamento da indústria, que objetiva aumentar o nível de abstração na forma de se descrever sistemas de software. A idéia principal é usar linguagens de modelagem, que sejam padrões bem estabelecidos, como linguagens de programação, ao invés de utilizá-las como simples linguagens de projeto e especificação. Sendo assim, os desenvolvedores constroem modelos independentes de plataforma que são processados por compiladores de modelos para realizarem transformações nos mesmos para algum contexto ou para gerar código final em uma determinada linguagem e modelo de componentes.

O conceito de independência de plataforma é relativo e já foi inicialmente usado dentro da arquitetura OMA (*Object Management Architecture*), onde um de seus elementos principais, CORBA, era considerado independente de plataforma e permitia um certo grau de independência das linguagens de programação e dos sistemas operacionais. No entanto, o conceito de arquitetura baseada em modelos permite aumentar o grau de abstração muito além do objetivado anteriormente. No contexto deste trabalho, ser independente de plataforma significa oferecer:

- Independência de sistemas operacionais (Windows, Unix, VM, etc);
- Independência de linguagens de implementação (C++, Java, etc);
- Independência de tecnologias de formatação das informações (XML DTD's, XML *Schemas*, etc); e
- Independência de *middleware* (J2EE, CORBA, .NET, etc).

O surgimento da arquitetura MDA na OMG causou um forte impacto nos vários domínios de serviços que já haviam sido especificados em conformidade com a metodologia da arquitetura OMA (através de interfaces IDL CORBA), pois a idéia da independência de plataforma, própria da arquitetura MDA, abre espaço para uma nova forma de especificar estes domínios, de forma que estes não fiquem restritos ao mundo CORBA conhecido. Um destes domínios diz respeito aos sistemas de gerência de workflows. Estes por sua vez, são sistemas que automatizam processos de negócios e já foram objeto de padronização em uma série de organizações. Os processos de negócios são conjuntos de um ou mais procedimentos que coletivamente representam um objetivo de negócios de uma organização[WFM99]. Duas especificações representam de forma mais significativa o processo de padronização de workflows que se desenvolveu na OMG: a especificação de serviço de gerência de workflows baseado em CORBA [OMG00a] e

a especificação EDOC[1] [OMG02b] que entre outros objetivos, procurou definir a automação dos processos de negócios de uma forma independente de plataforma.

Esta tese aborda os problemas relacionados com a especificação EDOC (particularmente em seu perfil para processos de negócios) , no que diz respeito às suas limitações e pendências para especificar, adequadamente, soluções baseadas em processos de negócios que sejam plenamente interoperáveis e independentes de plataforma. Desta forma, o trabalho procura complementar e estender a especificação, através da adição de um modelo independente de plataforma, para o ambiente de execução dos componentes de processos de negócios especificados.

As idéias da tese serviram de base para a criação de dois protótipos: um protótipo chamado de AGMC (*Ambiente de Gerência de Meta-Componentes*) que implementa um ambiente de suporte para a criação de modelos independentes de plataforma e um protótipo que foi desenvolvido a partir do primeiro, para ser um ambiente de execução de processos de negócios compatível com o modelo proposto.

1.1 Definição do Problema

A criação de modelos e meta-modelos está intimamente relacionada com o conceito de metadados. Os metadados são dados sobre dados, que uma vez estruturados e transformados em informação, passam a representar um conceito mais refinado: as meta-informações. As meta-informações definem estrutura e significado para objetos de dados e são usadas para definir, relacionar e manipular objetos de dados. Sua principal importância está no fato de que, sem eles, os dados não podem ser devidamente interpretados automaticamente.

A ausência de meta-linguagens que se baseiem nas meta-informações, dificulta o intercâmbio de informações e aplicações em arquiteturas que dependem da gerência de meta-informações, tais como: ferramentas de desenvolvimento de aplicações, ambientes baseados em componentes, *data warehouses*, *middlewares* e portais de informações.

Os principais benefícios do uso de sistemas de gerência de meta-informação são: a simplificação da integração de sistemas heterogêneos; o aumento da interoperabilidade entre aplicações, ferramentas e serviços; uma maior reutilização de módulos, sistemas e dados; e o suporte para a criação de arquiteturas baseadas em modelos. Entre os principais requisitos de um sistema de gerência de meta-informações podemos identificar: a necessidade de artefatos bem definidos para representar as meta-informações e a necessidade de diversos níveis de abstração que representem níveis diferentes de modelos e meta-modelos complexos.

Algumas iniciativas tentaram atender a estes requisitos através do uso de sistemas baseados em XML [W3C00] e de suas DTD (*Document Type Definitions*) ou

[1] *A especificação EDOC (Enterprise Distributed Object Coputing) é um padrão da OMG que define artefatos de modelagem UML para processos de negócios e sistemas de software em geral. Ver capítulo 3.*

XML *Schemas*. Infelizmente, o resultado não foi plenamente satisfatório, pois XML não é um sistema de tipos, nem mesmo um modelo de objetos. XML possui um modelo de dados baseado em estruturas de *tags* e não pode expressar a semântica de meta-informações gerais que envolvam classes e associações complexas, pois seus DTDs definem apenas uma forma de intercâmbio de metadados [IYE02] [OMG00e]. Além do mais, as vantagens oferecidas pelos esquemas XML (XML *Schemas*) se limitam a melhorar as características de intercâmbio dos DTDs XML tais como: herança, tipos de dados, entidades parametrizadas, etc. As características de interoperabilidade e integração de aplicações, ferramentas e serviços são deficientes sem o perfeito entendimento e descrição de seus vários aspectos semânticos.

Com o surgimento da especificação MOF [OMG00d] e, posteriormente, das especificações XMI [OMG00e] (na OMG) e a JMI [SUN01] (na comunidade Java), uma série de valores foram agregados ao que se desejava para os sistemas de gerência de meta-informações. Entre estes valores temos:

- Modelo semântico comum;
- Modelos de Programação padronizados para CORBA e Java;
- Formato de intercâmbio comum e
- Modelagem com múltiplos níveis de abstração.

A indústria da computação, em seu constante intento de buscar por novas maneiras de melhorar a produtividade no desenvolvimento de software, bem como a qualidade e longevidade dos sistemas criados, experimentou importantes paradigmas que contribuíram para este contexto, como por exemplo: a orientação a objetos, desenvolvimentos baseados em componentes, *design patterns* e infra-estruturas para computação distribuída. O surgimento da especificação MDA também foi fortemente influenciado por esta indústria de software.

A arquitetura baseada em modelos proposta na especificação MDA objetiva aumentar o nível de abstração na forma de se descrever sistemas de software, através do uso de linguagens de modelagem, já estabelecidas como padrões da OMG (MOF, UML e CWM). A Programação com linguagens de modelagem, com tais características, levaria a um aumento de produtividade, de qualidade e de reutilização dos esforços de programação desenvolvidos. Em termos mais gerais poder-se-ia dizer que os objetivos chaves da arquitetura são propiciar a gerência de definição, desdobramento e integração de sistemas e dados de aplicação complexos, mascarando as complexidades dos ambientes de computação distribuída heterogêneos.

A especificação EDOC contribui para este contexto com um framework[2] de modelagem, baseado em UML 1.4, o que aumenta o nível de abstração na forma de

[2] *Framework – é um conjunto de classes que incorpora um projeto abstrato de solução que pode ser aplicada a uma família de problemas relacionados [JOH88].*

especificar sistemas de software, através do uso de perfis UML que representam diferentes domínios de modelagem que podem ser combinados para compor uma solução completa. Um destes perfis UML é o perfil UML para processos de negócios (perfil EDOC-BP), o qual descreve um conjunto de extensões UML que podem ser usados para modelar o comportamento de um sistema com relação ao s seu(s) processo(s) de negócio(s) interno(s). Além disso, visando possíveis interações com repositórios de meta-informações MOF, a especificação EDOC propôs uma série de mapeamentos que permitiam que seus artefatos de perfis pudessem ser disponibilizados ou representados através de meta-modelos MOF e vice-versa.

Como o mecanismo de perfis UML usado pela especificação EDOC também é utilizado na arquitetura MDA para elevar o nível de abstração na definição dos sistemas de softwares, várias aplicações MDA fizeram uso dos conceitos de composição e colaboração EDOC para criar exemplos concretos de aplicações independentes de plataforma, que pudessem servir de provas conceituais dos princípios MDA envolvidos nestes exemplos. Entre estas provas conceituais, estavam algumas aplicações de negócio que faziam uso maciço do perfil UML de processos de negócios e que foram apresentadas posteriormente em documento da OMG que reunia uma série de experiências piloto para os conceitos da especificação EDOC [OMG01u].

Os problemas observados nestas implementações serviram de base para o consenso de que o perfil EDOC-BP não era suficiente para garantir a devida interoperabilidade desejada, para que diferentes aplicações de negócios pudessem interagir sem os problemas causados por pontos não abordados no perfil. Um destes pontos é a forma de interpretar os diversos artefatos do perfil EDOC-BP. Como conseqüência, em meados de 2002, a OMG oficializou esta situação com uma RFP (*Request for Proposal*) [OMG02d] solicitando um modelo independente de plataforma (PIM – *Platform Independent Model*) para o ambiente de execução interno de um componente de processos de negócios.

1.2 Análise de Necessidade

O trabalho desenvolvido no nosso grupo de pesquisa, ao longo dos últimos quatro anos e as publicações que surgiram em cada uma de suas etapas ([SIL00], [RIC02], [SIL02a] e [SIL02b]) foram voltados, principalmente, para a criação de arquiteturas de sistemas de gerência de workflows e para a utilização de meta-dados dentro deste contexto.

A partir do ano 2000, no entanto, a indústria convergiu de tal forma para o paradigma de orientação a modelos que o papel dos metadados e seus repositórios no contexto dos processos de negócios teve que ser revisto, pois não estava mais apenas restrito a aspectos adaptáveis do comportamento de um sistema de

gerência de processos de negócios, mas passou a envolver também, a essência do projeto do mesmo para que fosse independente de sua plataforma de execução.

1.2.1. Motivação

Com o surgimento da RFP de um ambiente de execução de processos de negócios [OMG02d], um novo horizonte se abriu para os grupos de trabalho que, de uma forma ou de outra, já haviam feito uso de modelos de metadados para flexibilizar protótipos de sistemas de gerência de processos de negócios. Principalmente, se estes protótipos fizessem uso de modelagem de processos de negócios através do perfil UML EDOC-BP e gerência de metadados baseada na especificação MOF.

A principal idéia por trás da RFP é a definição de um modelo independente de plataforma para ambientes de execução de processos de negócios que permita que diferentes soluções MDA que fazem uso de processos de negócios utilizem o mesmo modelo de ambiente de execução para a geração de seus componentes, diminuindo assim, as chances de estas encontrem problemas de interoperabilidade durante suas colaborações em diferentes contextos (Figura 1-1). Além disso, o modelo a ser proposto deve levar em consideração outros padrões relacionados com workflows e processos de negócios que tenham sido definidos no âmbito da OMG e não foram considerados na criação do perfil EDOC-BP, como é o caso do *Workflow Management Facility* [OMG00a]. O modelo deve também propor soluções para melhor integrar os repositórios de modelos de processos de negócios com o modelo de execução propriamente dito.

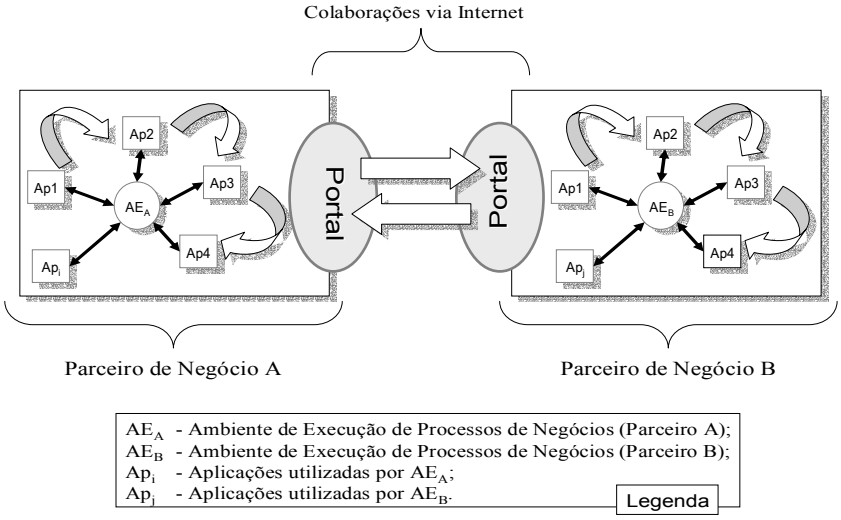

Figura 1-1 - Exemplo de colaborações entre parceiros de negócios.

Na Figura 1-1, duas organizações distintas (*Parceiros de Negócio* A e B) são representadas para ilustrar alguns dos problemas de interoperabilidade relacionados com a integração de processos de negócios que foram desenvolvidos de forma independente e que são executados por diferentes ambientes em cada uma das organizações (AE_A e AE_B). Um dos problemas mais comuns neste tipo de interação é a falta de um modelo de informações comum que permita que todas as informações necessárias para uma operação conjunta possam ser obtidas por introspecção entre processos envolvidos e seus repositórios. Isto normalmente leva a um processo de integração demorado em que cada organização é re-adaptada para estar preparada para fazer uso dos recursos de uma outra organização. Por exemplo, os ambientes AE_A e AE_B e suas respectivas aplicações precisariam ser adaptados para participar nas interações conjuntas. Outro problema comum é redefinição das políticas de segurança das empresas envolvidas. Quanto maior o número de empresas, mais crítica será a tarefa de criar esquemas de colaborações que não criem algum tipo de falha de segurança para alguma das organizações consideradas. Alem disso, a diversidade de protocolos de segurança e de plataformas de *middleware* envolvidas tornam o problema particularmente complexo à medida que cresce o número de organizações que devem interagir entre si. É neste contexto que criar modelos para os ambientes de execução que devem ser usados nas diferentes organizações se torna interessante, pois permite que haja uma abordagem única (do ponto de vista conceitual) para todas as questões de interoperabilidade envolvidas. Por exemplo, se não é possível exigir que todas as organizações usem a mesma plataforma de middleware (ou política de segurança), faz-se uso de modelos para especificar aplicações, políticas e ambientes apropriados e aplicam-se transformações que geram componentes interoperáveis para as diferentes plataformas e políticas. Se não é possível um modelo de informações único para todas as organizações, faz-se uso de meta-modelos comuns que permitam a representação dos diferentes modelos de uma forma que todos possam entender e aprender a usar. É nesta direção que o modelo proposto nesta tese procura contribuir para o contexto dos processos de negócios nas arquiteturas baseadas em modelos.

1.2.2. Cenários

A definição de um modelo independente de plataforma está intimamente relacionada com os cenários de engenharia de software que são utilizados na arquitetura MDA. Estes cenários representam o ciclo de vida do sistema correspondente ao modelo independente de plataforma a ser desenvolvido, e apresentam as diferentes perspectivas das pessoas e sistemas que interagem com uma ferramenta MDA para desempenhar uma função específica dentro do processo de construção de uma solução baseada em modelos. Apesar da grande importância dos cenários de engenharia de software, eles não são os únicos que

podem ser considerados no contexto das arquiteturas baseadas em modelos [STE03]. Sendo assim, o último cenário da lista apresentada a seguir, considera a gerência de metadados a partir de componentes gerados nos três primeiros.

- **Cenário 1 – A Meta-Modelagem** : Este cenário mostra o ponto de vista dos projetistas de meta-modelos de informação para diferentes domínios de interesse. As definições feitas aqui são usadas para nortear os projetos de software relacionados com os meta-modelos definidos. Na Figura 1-2, no trecho correspondente ao número 1, vê-se projetistas criando e adicionando meta-modelos de informação para organizações, processos de negócios, componentes, etc. Estes meta-modelos podem ser também extensões para o meta-modelo UML conhecidas como *perfis UML*. Sendo assim, neste cenário devem ser definidos todos os perfis UML que servirão de base para o modelo independente do próximo cenário. Por exemplo, neste cenário poderiam ser criados artefatos de modelagem que seríam usados para representar um modelo de um processo de negócio.

- **Cenário 2 – A Modelagem :** Este cenário mostra o ponto de vista dos programadores. Deste ponto de vista, os diversos tipos de modelos da arquitetura MDA são construídos a partir das definições feitas no cenário anterior. É neste cenário, por exemplo, que o modelo independente de plataforma, solicitado pela RFP [OMG02d] deve ser especificado. Além disso, estes modelos podem evoluir ao longo do tempo, a medida em que são refinados ou alterados, para se tornarem específicos de uma determinada plataforma escolhida. Na Figura 1-2, no trecho correspondente ao número 2, temos a definição de instâncias dos modelos de informações definidos no cenário anterior. Por exemplo, neste cenário os artefatos do cenário anterior que descrevem um modelo de processo de negócio seríam instanciados para criar um modelo de um processo de negócio específico (ex. processo de licitação para compra de produtos oferecidos por um conjunto de fornecedores cadastrados).

- **Cenário 3 – A Geração de Código:** Este cenário mostra o ponto de vista dos meta-programas. Meta-programas são softwares que manipulam meta-informações para gerar programas. Deste ponto de vista, modelos específicos de plataforma podem ser interpretados para dar origem a modelos específicos para uma plataforma de software, para depois serem transformados em código fonte equivalente para a plataforma em que foi especificado. Na Figura 1-2, no trecho correspondente ao número 3, temos meta-programas gerando código fonte a partir de modelos criados no cenário anterior. Por exemplo, um modelo de processo de negócios independente de plataforma pode ser transformado para se tornar um

modelo dependente de plataforma que pode ser também transformado em código fonte da plataforma escolhida (ex. CORBA, EJB).

- **Cenário 4 – A gerência de metadados :** Este cenário mostra o ponto de vista dos componentes de software e eventualmente, também das plataformas, que tenham sido gerados a partir de modelos e que precisem fazer uso de outros modelos que representem aspectos variáveis de seus comportamentos. Estes componentes devem ser capazes de fazer uso, em tempo de execução, do ambiente em que foram gerados para se configurarem dinamicamente. Na Figura 1-2, no trecho correspondente ao número 4, têm-se as interações entre componentes de processos de negócios que interagem entre si a partir de definições de processo que estão descritas no ambiente de gerência. Além das definições de processos, os componentes podem também acessar outras meta-informações que sejam importantes para o comportamento reflexivo que lhe foi definido em projeto. Por exemplo, um componente que foi gerado para implementar um processo de negócio correspondente ao processo de licitação para compras de produtos de fornecedores poderia ser atualizado dinamicamente para acomodar mudanças nos procedimentos de licitação.

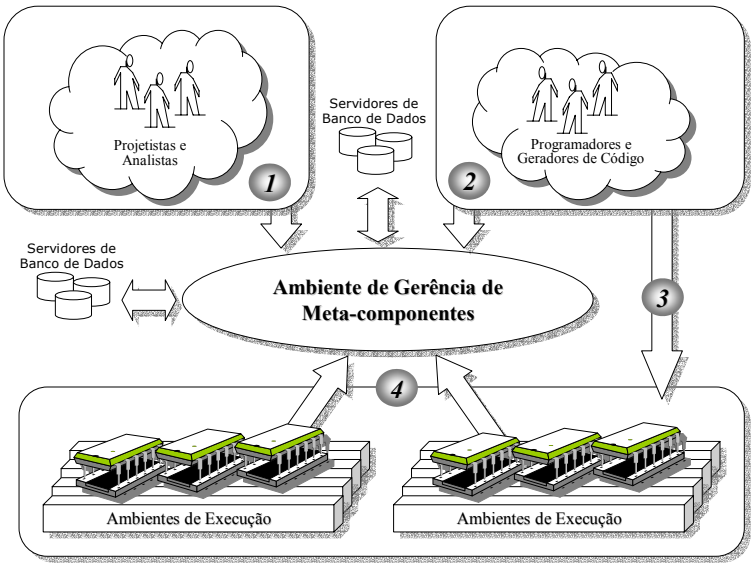

Figura 1-2 - Cenários para arquiteturas baseadas em modelos.

1.2.3. Objetivos

Esta tese tem o objetivo geral de explorar os fundamentos das arquiteturas baseadas em modelos e sua aplicabilidade no contexto dos sistemas de gerência de processos de negócios. Neste sentido, um modelo independente de plataforma

para um ambiente de execução de processos é desenvolvido conceitualmente, a partir da experiência obtida na criação de diferentes protótipos de sistemas de gerência de workflows, e submetido aos vários cenários de engenharia de software apresentados anteriormente, para que possa permitir a geração de um componente de software correspondente ao ambiente de execução desejado.

Para tanto, um protótipo de ambiente de modelagem e meta-modelagem foi desenvolvido para concretizar os cenários mencionados para o modelo conceitual proposto. Ao final do processo de desenvolvimento, o componente de software resultante foi testado juntamente com o protótipo do ambiente de meta-modelagem para verificar sua capacidade de interagir com o mesmo em tempo de execução, para se configurar dinamicamente.

1.3 Contribuição

As seguintes contribuições são feitas como resultado deste trabalho:

1. Criação de modelo independente de plataforma para ambientes de execução de processos de negócios, definidos através de perfis UML EDOC;
2. Criação de um protótipo de ambiente de desenvolvimento para concretizar os cenários relacionados com o ciclo de vida do modelo independente de plataforma proposto;
3. Criação de um protótipo de ambiente de execução de processos a partir das interfaces geradas pelo protótipo anterior.

1.4 Metodologia

A metodologia adotada neste trabalho consistiu nas seguintes etapas: 1) levantamento das principais técnicas e padrões para manipulação de metadados que surgiram nos últimos cinco anos; 2) Estudo do papel potencial destas técnicas e padrões na viabilização do conceito de arquiteturas baseadas em modelos, definindo para isto as principais características e cenários de utilização; 3) análise dos cenários definidos na etapa anterior, visando a construção de um ambiente de meta-modelagem para a concretização dos cenários; 4) Apresentação da proposta de modelo independente de plataforma para o ambiente de execução de processos de negócios; 5) concretização e validação do modelo proposto nos vários cenários considerados através de sua formalização no ambiente de meta-modelagem desenvolvido; e por fim, o componente gerado a partir do ambiente de meta-modelagem será validado através de suas interações com o ambiente que o gerou.

1.5 Organização

Esta tese está estruturada da seguinte forma: o capítulo 2 apresenta os principais fundamentos relacionados com sistemas de modelagem e meta-modelagem; o capítulo 3 apresenta um resumo da teoria de workflow e processos de negócios; o capítulo 4 apresenta os principais requisitos necessários para o trabalho; o capítulo 5 descreve uma proposta de modelo independente de plataforma para ambientes de execução de processos de negócios; o capítulo 6 apresenta a validação dos protótipos desenvolvidos nos cenários considerados e uma análise dos principais resultados obtidos; Por fim, o capítulo 6 conclui nossos resultados fazendo as considerações finais e apresentando algumas propostas de trabalhos futuros.

1.6 Resumo

Este capítulo apresentou, sucintamente, as bases e estrutura deste trabalho de tese. Ele introduziu o problema da pesquisa e suas principais questões e hipóteses, procurando justificar o trabalho no contexto em que se insere. A metodologia adotada foi rapidamente explicada, bem como a organização dos pontos discutidos no trabalho.

Capítulo 2
Conceitos, Tecnologias e Padrões

2.1 Introdução

Este capítulo apresenta uma breve revisão dos principais conceitos, tecnologias e padrões que são fundamentais para o entendimento da proposta deste trabalho. Entre eles estão duas áreas de pesquisa particularmente importantes. A primeira delas é a área dos sistemas reflexivos e a segunda é a área dos sistemas de modelagem e meta-modelagem de informações. Esta apresentação se concentra nos tópicos: princípios básicos, modelos de arquiteturas, padrões e tecnologias principais destas áreas. O principal objetivo é fornecer a terminologia que será adotada nos capítulos seguintes.

2.2 Sistemas Reflexivos

Um bom começo para se entender os sistemas reflexivos no contexto dos sistemas de computação é partir da definição do termo *reflexão* no senso comum e por analogia, chegar ao seu significado dentro deste novo contexto. No dicionário Aurélio [FER99], há sete definições para este termo. Destas, duas das definições são mais adequadas para se entender reflexão computacional a partir do seu significado no senso comum. São elas:

- **Definição 1** - "Volta da consciência, do espírito, sobre si mesmo, para examinar o seu próprio conteúdo por meio do entendimento, da razão." e
- **Definição 2 –** "Modificação da direção de propagação de uma onda que incide sobre uma interface que separa dois meios diferentes e retorna para o meio inicial."

Quando observamos que alguns sistemas computacionais podem ser desenvolvidos para possuírem características parecidas com as definições acima, começamos a ver sentido na expressão "sistemas reflexivos" ou "reflexividade computacional". Tal comparação inspirou B. C. Smith a formalizar pela primeira

vez, os fundamentos dos sistemas de computação reflexivos. Estes fundamentos são resumidos através de sua Hipótese de Reflexão [SMI82], que diz que um sistema pode ser desenvolvido para manipular uma representação de suas características internas, da mesma forma que este manipula representações de seu domínio de aplicação. Esta representação de si mesmo é o que nos faz dizer que o sistema em questão tem uma *meta-representação* e permite a introspecção destas características. Se além da representação, existir também um relacionamento de causa-efeito entre a meta-representação do sistema e seu estado e comportamentos atuais que são suas características internas, o sistema diz-se, também, ser reflexivo [MAE87]. Este relacionamento de causa-efeito é denominado de conexão causal. A meta-representação de um sistema pode ser total, ou seja, um princípio seguido ao longo do projeto de um sistema, o que caracteriza uma arquitetura reflexiva ou parcial quando considera apenas alguns aspectos do mesmo.

2.2.1. Arquiteturas Reflexivas

O modelo básico de uma arquitetura reflexiva deve representar a essência de um sistema reflexivo, ou seja, a conexão causal que existe entre os sistemas do domínio da aplicação e a meta-representação que reflete seus estados internos. Na Figura 2-1, o domínio da aplicação é chamado de nível base e é independente e irrestrito, enquanto que o domínio da meta-representação está sempre vinculado com o domínio da aplicação.

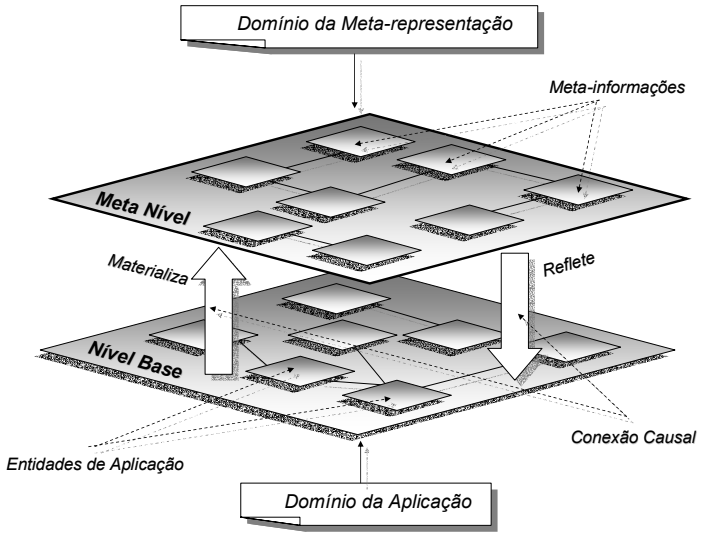

Figura 2-1 - Modelo básico de arquitetura reflexiva.

A *reificação* (do inglês, *reification*), ou simplesmente materialização, é a disponibilização das características internas do nível de informação (base) e

implica em uma representação do sistema em termos de elementos de programação que podem ser utilizados durante uma execução normal deste sistema.

A reflexão é o processo que ao perceber alterações na meta-representação (ex. ocorrência de nova materialização) reflete estas alterações nos elementos materializados do nível base. Estes processos de materialização e reflexão são conseqüências diretas da propriedade de conexão causal descrita anteriormente.

Dois outros conceitos importantes a se considerar são os conceitos de *meta-espaço* e de *separação de interesses*. *Meta-espaço* é um conceito que está intimamente associado com um elemento do nível base e corresponde ao conjunto formado por todas as entidades de um meta-nível que fazem parte, de alguma forma, da representação deste elemento do nível base no meta-nível em questão. Já o conceito de separação de interesses consiste em separar os aspectos funcionais próprios dos sistemas que executam no nível base daqueles aspectos não funcionais[3] que estejam relacionados com o sistema do nível base. Estes aspectos não funcionais são representados no meta-nível.

É importante que fique claro que o modelo básico da arquitetura reflexiva (Figura 2-1) pode ser estendido para uma arquitetura hierárquica com um nível base e vários meta-níveis. Neste caso, os relacionamentos de materialização e reflexão passam a existir também entre os pares de meta-níveis vizinhos. Além disso, a separação de conceitos, que existia entre nível base e meta-nível, também existirá entre os meta-níveis da arquitetura hierárquica (Figura 2-2).

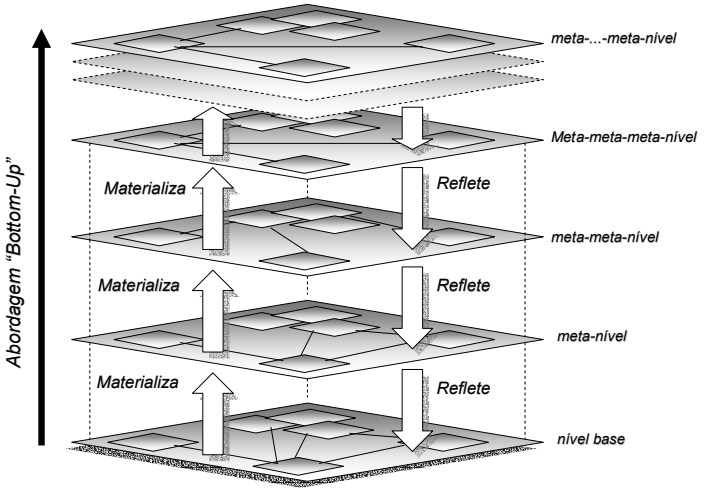

Figura 2-2 - Modelo de arquitetura reflexiva de vários meta-níveis.

[3] *Aspectos não funcionais – são aspectos relacionados com as propriedades não funcionais de um sistema. Por exemplo: performance, confiabilidade, segurança, etc.*

A arquitetura reflexiva hierárquica acima é também chamada de *torre reflexiva de multi-níveis* [OKA95]. Em princípio, esta torre pode ter infinitos níveis hierárquicos. Na prática, entretanto, o processo de recursão reflexivo mostrado termina pela implementação do último nível com um interpretador embutido no código, o que o torna fixo e não sujeito à reflexão. Tipicamente, a torre de reflexão é idealizada de uma forma meta-circular, usando recursão entre dois níveis adjacentes, com exceção do último, sendo o mesmo implementado na mesma linguagem do nível que ele materializa. Em outras palavras pode-se dizer que os ciclos de materialização e reflexão se repetem a cada par de níveis adjacentes. Além disso, é importante observar que, na torre reflexiva, a direção do processo de criação vai de baixo para cima, ou seja, é uma abordagem "*bottom-up*" [COS01], pois tudo começa na análise e seleção das entidades do nível base que vão ser representadas no meta-nível, só depois é que entidades do meta-nível superior são criadas para representar as anteriores e assim por diante.

2.2.2. Classificação

Infelizmente, ainda não podemos dizer que existe uma sistemática definitiva para categorizar os vários tipos de reflexão computacional, pois não há um consenso a respeito de quais os critérios que devem ser usados. No entanto, entre as várias tentativas, podemos citar: [CAZ98] que apresentou um dos primeiros conjuntos de critérios de classificação; [KIC92] que sugere quatro princípios que se baseiam nas características de protocolos que regem o comportamento de objetos nos planos de meta-informação; [OLI98] selecionou um subconjunto da proposta de [CAZ98] e adicionou mais um critério, proposto inicialmente por [RID99], para criar um conjunto final de critérios baseados nos aspectos: vinculação, reificação / materialização, intervenção e execução; e [COS01] apresenta uma série de comparações com os principais conceitos associados com diferentes estilos de reflexão. Para resumir esta discussão os principais conceitos e comparações são mostrados a seguir.

Reflexão Comportamental e Reflexão Estrutural

A reflexividade estrutural é definida como a habilidade de uma linguagem fornecer uma materialização do conteúdo e estrutura de um programa em execução, junto com os tipos de dados abstratos que são parte de um programa (ex. a API Reflexiva Java) [WAT98][MAL96]. Por outro lado a reflexividade comportamental, também conhecida como reflexividade computacional é a habilidade de uma linguagem fornecer uma representação completa de sua própria semântica, em termos dos aspectos internos de seu ambiente em tempo de execução [MAE87][WAT98]. Isto permite, por exemplo, que um programa possa se auto descrever, se auto analizar e se auto modificar dinâmicamente. A

reflexividade estrutural normalmente lida com as propriedades funcionais de um sistema, enquanto que a reflexividade comportamental está tipicamente relacionada com as propriedades não funcionais [PUL01] do mesmo. É importante observar que estes dois conceitos são complementares entre si e a maioria das soluções que implementam arquiteturas reflexivas fornece ambos os tipos [COS01].

Reflexão Explícita e Reflexão Implícita

A diferenciação entre estes dois conceitos é feita através do aspecto visibilidade da transição de um nível para o outro dentro de uma arquitetura reflexiva. Quando esta transição é visível pelo próprio nível inferior, seja ele o nível base ou qualquer outro meta-nível, diz-se que a reflexão é explícita, caso contrário implícita. Repare-se que, quanto mais explícita for esta transição, mais acoplados serão os níveis da arquitetura o que implica em dificuldades de manutenção ao longo do ciclo de software.

Reflexão Procedimental, Reflexão Declarativa e Mista

Na reflexividade procedimental, a meta-representação do sistema é a própria implementação do mesmo, o que significa que sempre que a relação causal é implementada de forma implícita, ela é automaticamente garantida (ex. linguagens reflexivas orientadas a objetos). Em contraste, a reflexividade declarativa, a meta-representação é diferente da implementação do sistema, e consiste de sentenças e propriedades sobre seu comportamento e estrutura. Neste caso, a relação causal deve ser explicitamente mantida (ex. linguagem Maude) [MAE88]. Como conseqüência da abordagem declarativa, apenas um número limitado de aspectos pode ser reificado na prática, o que limita seu grau de flexibilidade. Por outro lado, a reflexividade procedimental permite que qualquer elemento do sistema possa ser reificado e manipulado, permitindo que comportamentos completamente novos possam ser introduzidos no mesmo. A reflexividade mista combina as características das duas anteriores, com o objetivo de evitar a complexidade das soluções que adotam reflexividade procedimental e as limitações das que usam reflexividade declarativa.

2.2.3. Protocolos de Meta-objetos

Protocolos de meta-objetos (*meta-object protocol* - MOP) [KIC91][OLI98] são interfaces e regras que permitem acesso reflexivo às implementações de meta-objetos nos vários meta-níveis de uma arquitetura reflexiva. Da mesma forma que as interfaces dos objetos do nível-base fornecem um protocolo para acesso às

funcionalidades da aplicação, as interfaces dos meta-objetos fornecem um meta-protocolo que permite acesso reflexivo às implementações do sistema.

A essência dos protocolos de meta-objetos é permitir que os usuários possam ajustar a implementação de um sistema ou aplicação para que fique mais adequado às suas necessidades particulares. Estes protocolos se baseiam na reflexão computacional e na orientação a objetos. A reflexão permite que o sistema exponha a implementação em um alto nível de abstração, tornando-a fácil de entender enquanto mantém a eficiência e portabilidade da implementação original. A orientação a objetos, por sua vez, fornece uma interface para a implementação por meio de classes, métodos, de tal forma que as variantes da implementação original possam ser geradas através de especialização por herança. As instâncias de tais classes são chamadas de meta-objetos. A noção do protocolo se relaciona aqui com a interação entre objetos e meta-objetos. Tipicamente, as interfaces para os meta-objetos contêm, pelo menos, métodos para criação e exclusão de instâncias, para leitura e escrita de atributos e definição de ligações entre meta-objeto e objeto.

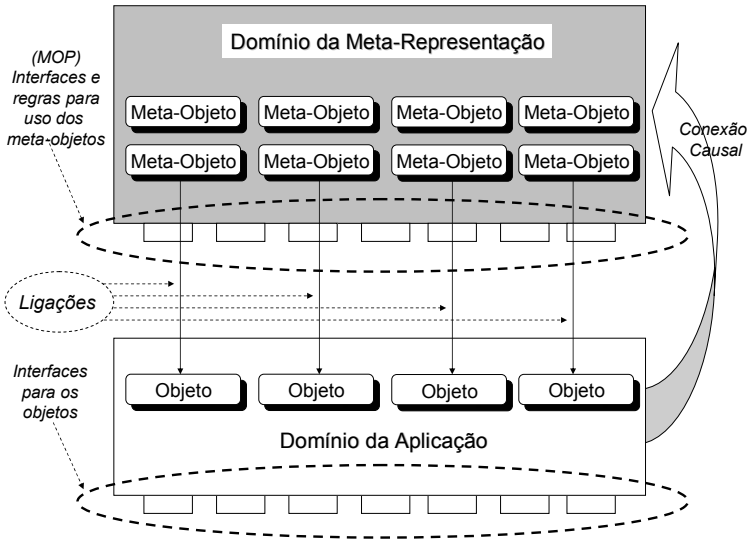

Figura 2-3 - O MOP nas arquiteturas reflexivas..

Na Figura 2-3, o MOP é mostrado através das interfaces para os meta-objetos que formam a meta-representação de um sistema. Nela os requisitos funcionais de um sistema são satisfeitos pelos objetos do domínio da aplicação, enquanto que os requisitos não funcionais são satisfeitos pelos meta-objetos da meta-representação.
É importante observar que não é suficiente definir um conjunto de meta-objetos associados a um grupo de requisitos não-funcionais para dizer que um sistema no domínio de aplicação já pode se beneficiar da funcionalidade oferecida pelo MOP. É preciso que os meta-objetos estejam ligados (*bound*) aos objetos da aplicação para que o comportamento de um objeto de aplicação possa não só usufruir das

funcionalidades do meta-objeto, mas também ser modificado em tempo de execução, bastando para tal, mudar o meta-objeto ao qual ele estaria ligado. O conceito de MOPs é importante para a nossa abordagem e pode vir a se tornar complexo à medida que os meta-objetos representam entidades complexas do domínio de aplicação e se aumente o número de níveis da meta-representação de um sistema.

2.3 Modelagem e Meta-modelagem

Podemos definir modelagem e meta-modelagem com base no produto final de cada uma delas. Sendo assim, a modelagem de informações é basicamente a atividade de construir modelos para as informações segundo algum princípio. Enquanto que a meta-modelagem seria o equivalente a construir modelos para descrever modelos. Estes modelos de modelos seriam chamados de meta-modelos. Para construir os modelos e os meta-modelos é necessário o que chamamos de metadados. Os metadados são definidos como *dados que descrevem dados* [IAN98]. Eles são também freqüentemente usados como sinônimo para um conceito parecido que é o conceito de meta-informação. Isto é válido para muitos contextos, mas para alguns outros, cabe uma pequena diferenciação entre os dois, pois há uma sutil diferença de generalidade que deve ser esclarecida. Esta diferença existe também entre os radicais das duas palavras, sendo o termo *dado* definido como *informação factual*, ou seja, baseada apenas na observação de um fato e que não tem ainda um significado agregado. A partir do momento que um dado ou conjunto de dados é interpretado e passa a ter um significado, ele deixa de ser apenas um simples dado e se torna uma informação [FER99].

Desta forma, alguns trabalhos na literatura fazem distinção entre os dois conceitos para explicitar aspectos estruturais e semânticos na definição de meta-informações. Por exemplo, [COS01] considera o conceito de meta-informação como uma maneira de descrever aspectos de estrutura e semântica relacionados com entidades de primeira classe de um sistema.

2.3.1. Arquiteturas de Meta-modelagem

Considerando que um processo de meta-modelagem pode ser recursivo, ou seja, um dado meta-modelo também pode ser representado por um modelo chamado de meta-meta-modelo chegaríamos também a uma estrutura hierárquica de vários níveis, semelhante à mostrada para as arquiteturas reflexivas.

Na Figura 2-4 o nível mais baixo representa o nível de informações M0, onde estão as entidades (dados/informações) normalmente manipuladas por sistemas e aplicações; no nível de modelos (M1), estão os modelos com as entidades (metadados/meta-informações) que são usadas para modelar os sistemas e as

aplicações; no nível de meta-modelos (M2), estão as entidades (meta-metadados/meta-meta-informações) que são usadas para representar os modelos do nível inferior; no nível de meta-meta-modelos (M3), estão as entidades (meta-meta-metadados/meta-meta-meta-informações) que são usadas para representar os meta-modelos do nível inferior e assim por diante, se houverem mais níveis.

É importante observar que além das descrições dos níveis, existem outras diferenças a se considerar quando as duas arquiteturas hierárquicas (Figura 2-2 e Figura 2-4) são comparadas. A primeira diferença diz respeito à abordagem *top-down*, própria das arquiteturas de meta-modelagem, e a segunda diz respeito ao tipo de relacionamento que existe entre os vários níveis. No primeiro caso, a abordagem *top-down* vem do fato de que a arquitetura de meta-modelagem é normalmente criada antes dos sistemas e aplicações que dela farão uso, desta forma, estes sistemas são ditos dependentes da arquitetura. Quanto ao relacionamento que existe entre os níveis, neste caso o relacionamento chama-se instanciação. Na instanciação, os elementos de um nível são instanciados a partir de elementos existentes no nível superior.

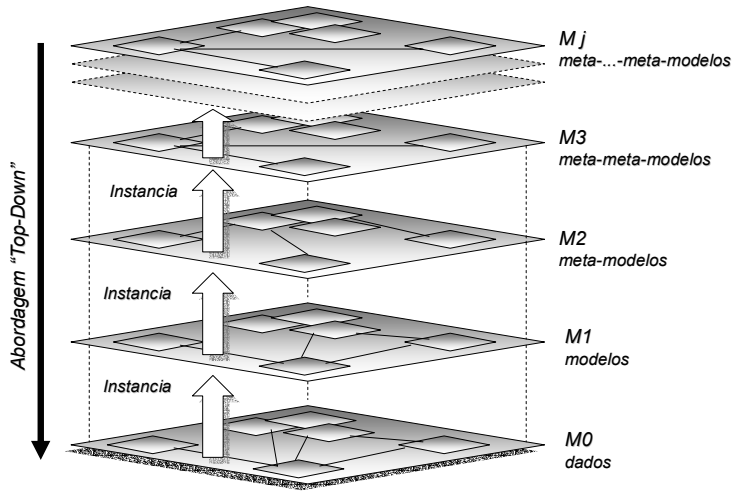

Figura 2-4 - Arquitetura de meta-modelagem.

2.3.2. Sistema de Gerência de Meta-informações

Para que sistemas e aplicações possam fazer uso de uma arquitetura de meta-modelagem é fundamental que esta arquitetura seja considerada a partir de um sistema de software que concretize as idéias conceituais da arquitetura e que agregue uma série de facilidades que permitam a definição, manutenção e intercâmbio de meta-informações e seus relacionamentos [CRA97]. Uma constante na maiorias das implementações de sistemas de gerência é a imposição de que,

uma vez definida e publicada uma meta-informação em alguns dos níveis da arquitetura, ela tenha que ser imutável, de forma que seus elementos instanciados nos demais níveis inferiores sejam mantidos consistentes [ISO01][COS01].

2.4 Tecnologias de Gerência de Meta-informações

Após a apresentação, nas seções anteriores, dos conceitos básicos relacionados com os sistemas reflexivos e dos sistemas de gerência de meta-informações, esta seção apresenta um apanhado das principais especificações e tecnologias baseadas no paradigma da orientação a objetos, que são usadas para a modelagem e meta-modelagem de informações no contexto desta tese.

2.4.1. A especificação MOF

A especificação MOF (*Meta-object Facility*) é o padrão da OMG para a gerência de meta-informações [OMG00d]. Ela define uma série de serviços CORBA que permitem aos objetos clientes: criar, alterar e eliminar metadados, em tempo de execução, através de servidores de repositórios dentro da arquitetura OMA (*Object Management Architecture*).

Atualmente, a especificação MOF é a principal tecnologia da estratégia da OMG para gerência de meta-informações, sendo bem mais abrangente do que a especificação anterior que definia o repositório de interfaces CORBA, pois fornece uma arquitetura genérica e adaptável para gerência dos modelos e meta-modelos definidos em sua arquitetura.

O objetivo inicial do padrão MOF era gerenciar os meta-modelos para auxiliar na análise e projeto de sistemas de software. No contexto da OMG, isto significou tornar MOF o ponto de unificação para integrar os diferentes meta-modelos que são usados no processo de desenvolvimento de software. No entanto, o escopo foi estendido de forma a permitir a definição de meta-modelos baseados em MOF para domínios arbitrários de aplicação. Como exemplo deste re-direcionamento, um conjunto de cenários mais geral foi apresentado na última versão da especificação listando domínios, tais como: desenvolvimento de software, gerência de tipos, gerência de meta-informações e gerência de *data warehouses* como os principais domínios de aplicação. Em geral, o único requisito para MOF ser aplicável em um dado domínio é que alguma estrutura possa ser imposta na meta-informação a ser representada.

A especificação MOF, em sua versão 1.4 [OMG02a], propõe uma arquitetura de meta-modelagem, um meta-meta-modelo (modelo MOF), um mapeamento abstrato, um mapeamento de MOF para interfaces IDL CORBA e um conjunto de interfaces reflexivas para os elementos da arquitetura proposta.

2.4.1.1 A Arquitetura MOF

A arquitetura MOF baseia-se no paradigma de orientação a objetos e no modelo de arquitetura de meta-modelagem de vários níveis de meta-informações já apresentado anteriormente. Isto significa que sua arquitetura é organizada em níveis onde as entidades em um dado nível são definidas como instâncias das entidades no nível imediatamente superior. Para maior clareza na apresentação dos conceitos, o padrão mostra o modelo com quatro níveis de abstração das informações: M0 (objetos), M1 (modelos), M2 (meta-modelos) e M3 (meta-meta-modelo).

Na Figura 2-5, um exemplo de arquitetura MOF é apresentado com alguns possíveis elementos e pacotes de elementos para cada um dos níveis definidos. Com exceção do nível M3 que contém o modelo MOF, todos os demais podem ser alterados e estendidos. A razão desta limitação está associada à discussão feita anteriormente a respeito da forma de limitar o número de níveis de uma arquitetura onde o nível de cima descreve o nível imediatamente inferior e é descrito pelo nível imediatamente superior. Uma das formas de parar este processo é definir o último nível em termos de si mesmo. O efeito colateral é a sua imutabilidade e este é o caso do nível M3 no MOF.

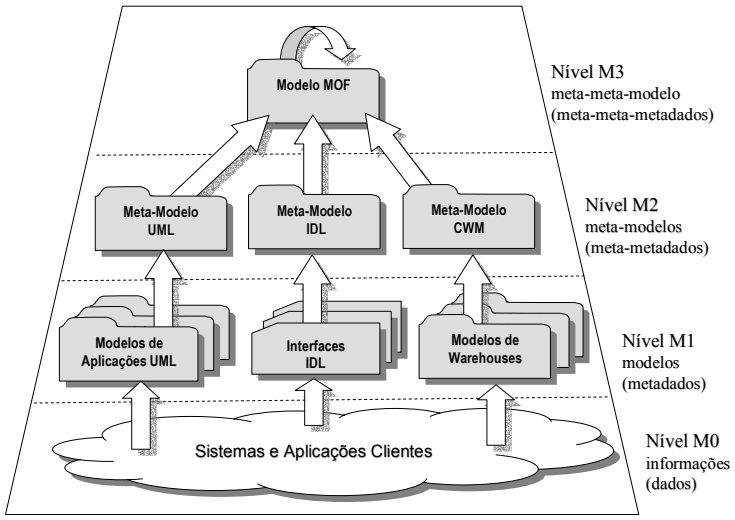

Figura 2-5 - Exemplo de uso da arquitetura MOF.

O nível M3 contém o modelo MOF (meta-meta-modelo) e é o unificador da arquitetura como um todo, pois fornece a linguagem abstrata que é usada para representar os meta-modelos definidos na arquitetura MOF.

O nível M2 contém os meta-modelos descritos através do modelo MOF. Na figura acima, três exemplos de meta-modelos são mostrados. São eles: o meta-modelo UML, o meta-modelo IDL e meta-modelo CWM. Os três são representados

através de elementos (artefatos) da linguagem do nível de cima, que é o meta-meta-modelo do M3. É importante deixar claro que a linguagem é suficientemente geral para representar outros pacotes definidos pelos usuários ou não. O fato de apenas três meta-modelos estarem presentes é puramente ilustrativo.

O nível M1, onde estão as meta-informações reais, consiste de modelos representando aplicações ou sistemas específicos. Cada modelo é definido de acordo com um simples meta-modelo M2, que fornece as meta-meta-informações para instanciar e interpretar os elementos do modelo.

Finalmente, no nível M0, temos as entidades que são o objetivo principal da modelagem, tais como os objetos de dados de um sistema. Note-se que, embora o nível M0 não seja explicitamente tratado na especificação, ele é crucial no contexto mais abrangente desta tese, pois suas entidades são o objetivo principal do ambiente de gerência proposto.

2.4.1.2 O Modelo MOF

O Modelo MOF consiste de um conjunto de elementos de modelagem que permitem a descrição de meta-modelos arbitrários na arquitetura MOF. Estes elementos são, na verdade, os tipos do meta-meta-modelo definido no nível M3, os quais são usados para instanciar elementos que formam os meta-modelos no nível M2. Os principais elementos do modelo MOF são mostrados na Figura 2-6 e descritos na tabela Tabela 2-1 a seguir:

Tabela 2-1 - Principais Elementos do Modelo MOF

Elemento	Descrição
Package	O elemento *Package* é usado para encapsular os demais elementos do modelo MOF e viabilizar a modularização na criação de meta-modelos. Ele é essencialmente um mecanismo de escopo para os elementos definidos em seu contexto.
Class	O elemento *Class* é o principal elemento do modelo MOF. Ele pode ter três tipos de elementos internos: *Attributes*, *References* e *Operations*.
Association	O elemento *Association* descreve o relacionamento entre dois elementos *Class*. Ele basicamente define um conjunto de dois pontos terminais que são elementos *Class* e é um dos pontos fracos da especificação, por não permitir relacionamentos com mais de dois elementos *Class*.
Data Type	*Um Data Type* é usado para representar um tipo anônimo que não corresponda a um elemento identificado de um meta-modelo. Normalmente, ele é usado como parte da especificação de outro elemento do meta-modelo, tal como para dar o tipo de um atributo de um *Class* ou parâmetro de um *Operation*.
Constraint	Este elemento permite que uma definição feita dentro de um meta-modelo seja expandida com relação a sua semântica, na forma de regras que descrevem a consistência de modelos derivados de um meta-modelo. Em princípio, *Constraints* podem ser expressos em qualquer linguagem, incluindo uma linguagem natural. No entanto, a especificação MOF recomenda o uso de uma linguagem formal, o OCL (*Object Constraint Language*), que é definida como

	parte da UML.
Reference	Uma *Reference* é usada internamente no elemento *Class* para explicitar o papel que suas instâncias desempenham quando fazem parte de uma associação representada pelo elemento *Association*.
Operation	O elemento Operation também é usado internamente no elemento Class. Ele fornece uma maneira de se associar operações a um elemento do meta-modelo que está em definição.
Exception	*Exceptions* representam exceções que podem acontecer na execução normal de uma operação definida pelo elemento *Operation*.
Constant	Constant representa uma constante, uma ligação entre um nome e um valor.
Tag	É o elemento que viabiliza uma alteração na forma de interpretar algumas instâncias de elementos do modelo MOF, no momento de realizar os mapeamentos tecnológicos previstos na especificação.

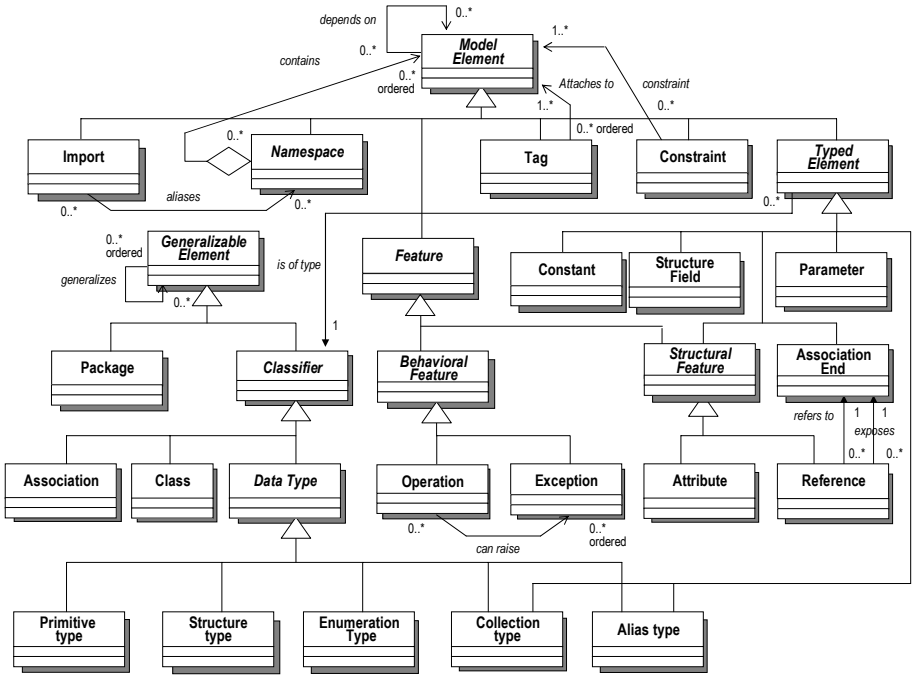

Figura 2-6 – O Modelo MOF (meta-meta-modelo) [OMG00d].

2.4.1.3 Mapeamentos tecnológicos

A arquitetura e o modelo MOF apresentados anteriormente estão relacionados com conceitos abstratos que não dependem de uma tecnologia específica de implementação. Deve-se ressaltar que nem mesmo com relação a CORBA, há uma dependência explicita até este ponto. Somente após a definição

dos mapeamentos propostos pela especificação, a dependência é estabelecida e conseguimos ver um possível modelo computacional para sua utilização.

Há três possíveis abordagens para se construir uma implementação concreta do ambiente de gerência de meta-informações. A primeira delas seria um implementação *ad hoc,* a segunda seria baseada nos mapeamentos propostos pela especificação e a terceira faria uso de uma ferramenta previamente desenvolvida que forneça o serviço de automatização do mapeamento tecnológico a partir da definição de um meta-modelo em termos do modelo MOF.

A abordagem *ad hoc* tem a vantagem de ser independente da tecnologia proposta no padrão, mas por outro lado, tem o inconveniente de deixar a cargo dos implementadores todos os detalhes relacionados com a implementação dos repositórios (se forem usados), formatos de intercâmbio de meta-informações, APIs e ferramentas, o que é certamente um caminho para futuros problemas de interoperabilidade com outras implementações disponíveis.

A segunda abordagem considera os mapeamentos tecnológicos sugeridos no padrão e tem a desvantagem de ser dependente da tecnologia sugerida, mas permite uma série de vantagens significativas no que diz respeito às informações fornecidas no padrão definido pela especificação.

A terceira abordagem parte do pressuposto que o implementador vai fazer uso de uma ferramenta que automatiza os mapeamentos disponíveis no padrão para só se preocupar com os detalhes não tratados pela especificação.

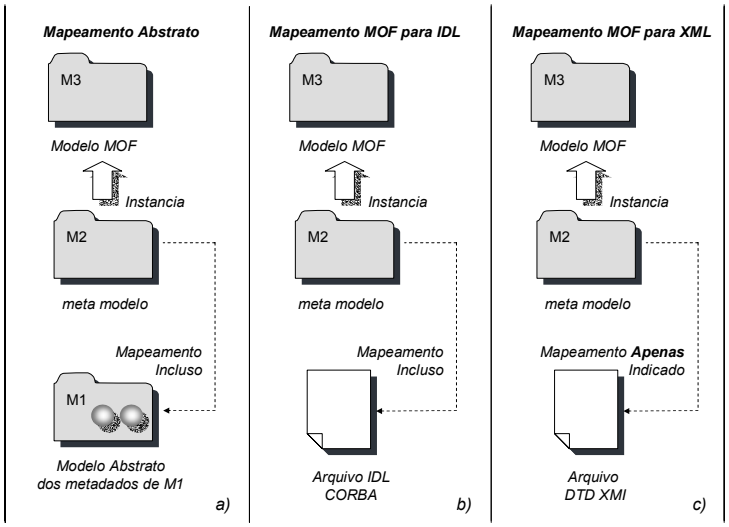

Figura 2-7 - Mapeamentos previstos na Especificação MOF.

Independente de qual abordagem venha a ser escolhida, é importante saber que há duas definições de mapeamentos definidos na especificação e uma apenas mencionada e posteriormente definida em uma outra especificação. São eles: o

mapeamento abstrato, o mapeamento de MOF para IDL CORBA (Figura 2-7 b) e o mapeamento de MOF para DTD XMI (Figura 2-7 c).

O mapeamento abstrato (Figura 2-7 a) descreve os relacionamentos que existem entre um meta-modelo (*Classes* e *Data Types* do M2) e suas correspondentes instâncias (valores do M1).

O mapeamento de MOF para IDL CORBA consiste na geração de definições de interfaces a partir das instâncias definidas no nível M2 da arquitetura. Este mapeamento permite que, após a implementação das interfaces geradas, os usuários dentro da arquitetura OMA possam gerenciar as meta-informações definidas nos níveis M1 e M2 em tempo de execução (Figura 2-7 b).

Os principais elementos considerados nos mapeamentos de meta-modelos MOF do nível M2 para a plataforma CORBA são mostrados na Tabela 2-2, juntamente com sua correspondência em termos de interfaces associadas e suas correspondentes descrições.

Tabela 2-2 - Elementos do mapeamento MOF para IDL

Elemento	Mapeamento para IDL
Package	Este elemento é mapeado em duas interfaces IDL: A interface *Package Factory* e a interface *Package*. A primeira é usada como fábrica de instâncias da interface *Package* e a segunda representa o conteúdo do pacote propriamente.
Class	Este elemento é também mapeado em duas interfaces IDL diferentes: a interface *Class Proxy* e a interface *Class*. A primeira serve de fábrica de instâncias para a interface *Class* e a segunda representa os meta-objetos do tipo *Class* com seus atributos (*Attributes*), operações (*Operations*) e referências (*References*).
Association	Este elemento é mapeado em uma única interface de nome *Association*. Ela representa todas as instâncias da associação como um conjunto de *links* derivados de uma associação.

Finalmente, temos o mapeamento de MOF para XML, que é apenas mencionado na especificação MOF, pois é definido em um outro documento devido a seu surgimento posterior. A especificação que descreve este relacionamento chama-se XML Meta-data Interchange format (XMI). Este tipo de acesso do MOF tem dois propósitos distintos:

a) Fornecer um padrão para sintaxe de transferência para meta-modelos definidos em MOF, permitindo a serialização e transporte de grandes quantidades de meta-informação entre ferramentas de modelagem, pois o mapeamento de MOF para IDL não é eficiente para este fim e

b) Permitir que ferramentas e aplicações, fora do ambiente CORBA, possam acessar meta-informação MOF.

Resumidamente, XMI define o mapeamento de MOF para XML em termos de um conjunto de DTD XML e regras de produção, usados para transformar meta-modelos em DTDs XML. Estas DTDs são então usadas para guiar a codificação de Meta-informação baseada em MOF na forma de documentos XML.

Essencialmente, a especificação MOF inclui definições do modelo MOF em termos de seus próprios artefatos, em termos de interfaces IDL, pela aplicação de mapeamento MOF-IDL e como DTD XML. Isto permite que a definição de meta-meta-modelo MOF seja usada para trocar meta-modelos, de forma similar à forma em que os meta-modelos são usados para trocar modelos.

2.4.1.4 Interfaces Reflexivas

Uma das vantagens dos meta-objetos é que eles permitem a um programa usar objetos sem o prévio conhecimento de suas interfaces. No contexto do MOF, o meta-objeto do nível M2 de um objeto M1 permite a um programa descobrir a natureza de qualquer objeto MOF do nível M1. Sendo assim, as interfaces reflexivas permitem a um programa: criar, atualizar, acessar, navegar e usar operações em objetos que são instâncias do nível M1; fazer buscas e atualizar os *links* usando objetos de associação do nível M1 e navegar na estrutura de um pacote do nível M1, sendo que toda esta funcionalidade é obtida sem fazer uso das interfaces específicas dos meta-modelos mapeados para IDL.

São quatro as interfaces reflexivas da especificação, descritas de forma abstrata para que suas implementações não sejam dependentes do mapeamento específico que será feito. As interfaces devem ser herdadas pelas interfaces específicas do nível M1 quando o mapeamento de MOF para IDL for feito.

Na Figura 2-8 é mostrado um exemplo da hierarquia de interfaces reflexivas e um exemplo conceitual de como estas são herdadas pelas interfaces específicas de um meta-modelo.

A principal interface reflexiva é a *RefBaseObject*. Ela fornece operações comuns para todos os objetos e meta-objetos MOF; RefObject fornece operações para os objetos do nível M1 e objetos *Class Proxy* associados com os mesmos; A *RefAssociation* fornece operações comuns para objetos de Association do nível M1; *RefPackage* fornece operações comuns para objetos Package do nível M1.

Embora seja correto que existam características de reflexão nas interfaces reflexivas, elas não são totalmente reflexivas, pois o MOF não define interfaces para modificação do comportamento reflexivo das interfaces.

Se compararmos esta funcionalidade com aquela oferecida pelo Repositório de Interfaces CORBA, quando usado em conjunto com a interface de invocação dinâmica DII (*Dynamic Invocation Interface*), observamos que há uma certa

semelhança no contexto de objetos de interface CORBA. No entanto, MOF é mais abrangente e tem uma riqueza semântica maior, pois o modelo MOF é poderoso em termos de modelagem e a sua arquitetura permite vários meta-modelos no M2.

A Figura 2-8 mostra a relação que existe entre um pacote P1 e as interfaces reflexivas definidas na especificação MOF. O Pacote P1 possui duas classes (C1 e C2) e uma associação (A1) entre elas. Após o mapeamento para CORBA, o pacote P1 dará origem às seguintes interfaces:

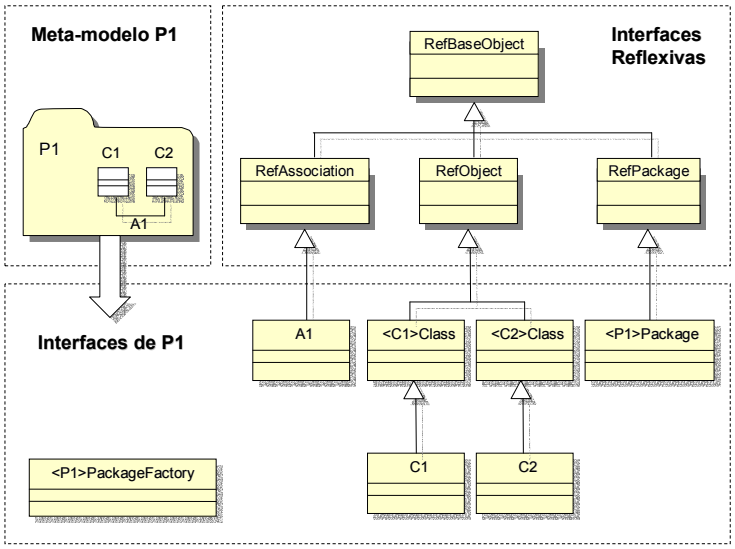

Figura 2-8 - Hierarquia de Interfaces Reflexivas

Tabela 2-3 - Relação entre Elementos de um meta-modelo e suas Interfaces

Elemento do Modelo	Interfaces Geradas	Interface Especializada	Descrição
P1	<P1>PackageFactory	---	Interface para fábrica de objetos.
	<P1>Package	RefPackage	Interface para o pacote de classes e associações.
C1	<C1>Class	RefObject	Interface para fábrica de classe C1.
	<C1>	<C1>Class	Interface para meta-objeto C1.
C2	<C2>Class	RefObject	Interface para fábrica de classe C1.
	<C2>	<C2>Class	Interface para meta-objeto C2.
A1	<A1>	RefAssociation	Interface para associação A1.

2.4.1.5 Outras especificações relacionadas

A especificação MOF deixou alguns pontos em aberto para que fossem mais bem estudados em outras especificações que viriam a seguir. Entre estes pontos estão a definição de um formato de intercâmbio de metadados via XML e a definição de um segundo mapeamento tecnológico para definir interfaces de

acesso baseadas em Java RMI. Estes pontos são tratados, respectivamente, pelas especificações XMI e JMI.

A especificação XMI

O XMI (XML Metadata Interchange) é um padrão de metadados criado pela OMG com o objetivo de prover interoperabilidade, no contexto de orientação a objetos, entre ferramentas CASE, repositórios de metadados e ferramentas de desenvolvimento, através da troca de metadados armazenados em sistemas de arquivos tradicionais ou no formato de fluxo *(stream*) de dados baseados no padrão XML. O padrão XMI também é uma iniciativa da comunidade industrial, envolvendo empresas líderes de mercado como a IBM e a Unisys, e foi adotado como um padrão OMG em março de 1999 [OMG00e].

O padrão XMI foi criado para permitir a troca de qualquer modelo de metadados especificado segundo o meta-modelo MOF e consiste em dois principais componentes: um conjunto de regras de produção de *Document Type Definitions* (DTDs) XML, que expressam como produzir DTDs para metadados codificados em XMI e um conjunto de regras de produção de documentos XML, que expressam como codificar metadados em documentos XML válidos e bem formados. O padrão XMI foi criado para ser utilizado como uma ponte universal entre as ferramentas de desenvolvimento orientadas a objeto, evitando desta forma a criação de uma variedade de formatos proprietários, cada um específico de cada fornecedor, à medida que cada ferramenta passe a importar e exportar metadados no formato XMI [OMG02e].

A especificação JMI

Um novo padrão para metadados que teve sua primeira versão concluída em 2001 é o JMI (*Java Metadata Interface*) [SUN01]. O JMI foi desenvolvido dentro da comunidade Java através do JCP (*Java Community Process*) da *Sun Microsystems* em seu grupo de trabalho JSR-40 (*Java Specification Request - 40*). O principal objetivo é definir uma infra-estrutura neutra que permita a criação, armazenamento, acesso, descoberta e troca de metadados usando interfaces Java e que seja baseado na especificação MOF da OMG.

Apesar da idéia da infra-estrutura neutra não ser nova, pois a especificação MOF já apontava para o XMI como principal meio de viabilizá-la, a experiência adquirida das implementações que se seguiram mostra que só os XML DTDs do XMI não eram suficientemente ricos semanticamente, para os objetivos para os quais MOF se propunha, pois tudo no MOF é orientado a objetos e o XML é apenas um sistema de tipos. Sendo assim as DTDs definem apenas formatos de troca de informações e não objetos. Apesar dos *Schemas* XML possuírem melhores características que um DTD, eles também não são modelos de objetos e não permitem múltiplos níveis de abstração como se precisa no MOF. Por estas razões

a especificação JMI vem se tornando cada vez mais, uma grande promessa de futuro para a gerência de metadados em ambientes heterogêneos como a Internet.

A especificação apresenta: uma arquitetura de meta-modelagem baseada em quatro níveis, nos quais o último deles implementa o modelo MOF (M3); um novo mapeamento específico, de MOF para Java; um conjunto de APIs para troca de informações entre serviços JMI chamada de *Stream based information Exchange* [OMG02e]; e uma nova hierarquia de interfaces reflexivas.

O mapeamento para Java é semelhante ao mapeamento para IDL e leva em consideração as regras de mapeamento abstrato do MOF. A diferença é que ao invés de gerar no final um arquivo de interfaces IDL CORBA, este novo mapeamento gera interfaces Java RMI (*Remote Method Invocation*).

A hierarquia de interfaces reflexivas mudou bastante no JMI. Agora são nove interfaces reflexivas (Figura 2-9) e uma classe reflexiva para as exceções chamada *RefException*.

Na Figura 2-9, pode-se observar um exemplo de como as interfaces reflexivas se relacionam com as interfaces relacionadas com um meta-modelo MOF que contém uma instância do elemento *Package* (P1), duas instâncias do elemento *Class* (C1 e C2) e uma instância do elemento *Association* (A). No processo de mapeamento de MOF para Java, os objetos gerados no modelo correspondente herdarão as interfaces definidas, como mostra a Figura 2-9. Ressalta-se a mudança com relação à adição das interfaces *RefFeatured* e *RefClass*. No mapeamento de MOF para IDL, *RefObject* permitia acesso a informações tanto para os *proxys* quanto para as instâncias das classes. Em JMI houve uma reorganização da hierarquia reflexiva e a inclusão de novas interfaces e classes foram incluídas para permitir acesso reflexivo também para os *DataTypes*.

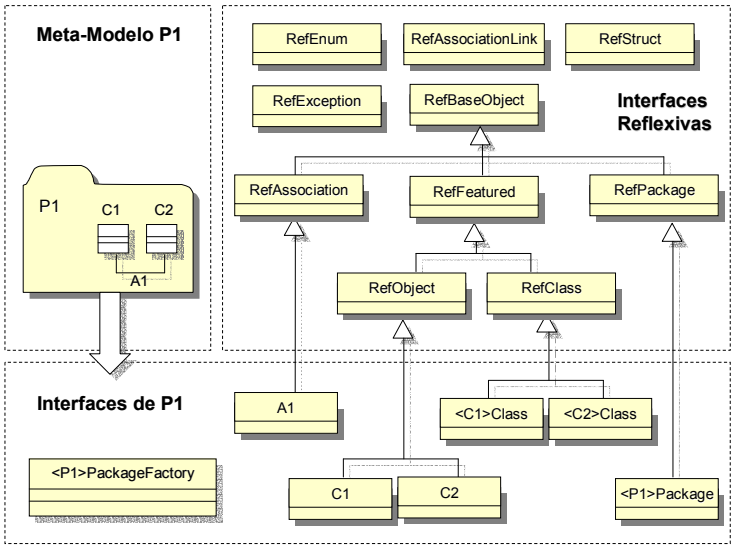

Figura 2-9 - Interfaces Reflexivas no JMI

A definição da API foi feita em junho de 2002 e deve alavancar o uso de MOF em ambientes Java e conseqüentemente na Internet como um todo, permitindo o uso em larga escala de Java para a gerência de meta-informações.

2.4.2. A especificação UML

A especificação UML (*Unified Modeling Language*) foi adotada pela OMG em 1997 como uma linguagem gráfica para visualizar, especificar, construir e documentar artefatos de software [OMG00b]. Apesar de contribuir sensivelmente para o processo de desenvolvimento de software, esta especificação não define nenhuma metodologia ou processo para desenvolver software. Em sua versão 1.4 [OMG01f], a especificação apresenta em sua estrutura: a semântica da linguagem UML, um guia da notação (sintaxe da linguagem UML), alguns exemplos de Perfis UML (mecanismo de extensão), Intercâmbio de Modelos (via XMI e IDL) e uma Linguagem de restrições em Objetos (OCL – *Object Constraint Language*).

2.4.2.1 Semântica da Linguagem

A descrição dos aspectos semânticos da linguagem está intimamente relacionada com a descrição da arquitetura de meta-modelagem em que o meta-modelo UML (Figura 2-10) está inserido e as formalizações necessárias para descrevê-lo.

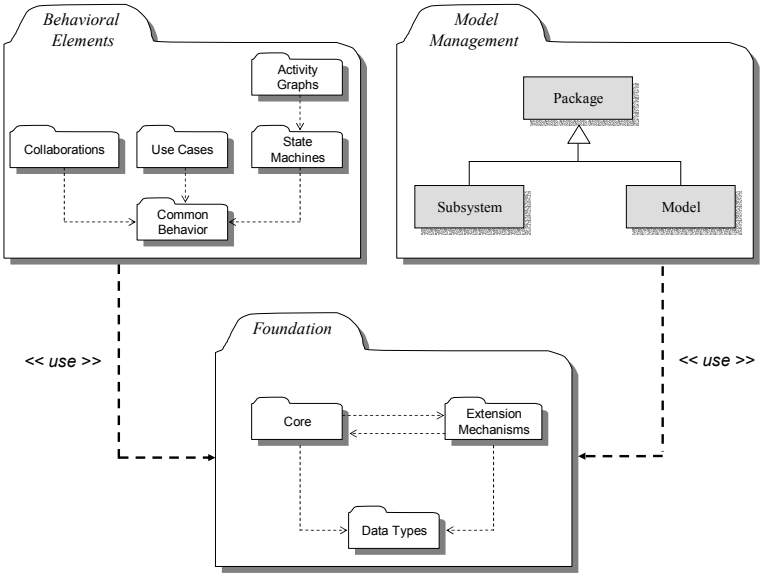

Figura 2-10 - Meta-modelo UML [OMG00b].

A arquitetura de meta-modelagem é a mesma apresentada no item sobre a especificação MOF e o nível em que o meta-modelo UML está inserido é o M2. No entanto, a seguinte consideração cabe neste ponto: como a MOF e a UML têm diferentes escopos e níveis de abstração, eles estão relacionados pelo que se conhece como meta-modelagem não-estrita (*loose*) ao invés de meta-modelagem estrita (*strict*)[4], que é o normalmente observado com meta-modelos puramente MOF com relação aos níveis adjacentes ao M2.

2.4.2.2 Guia de Notação da Linguagem

O guia de notação descreve a sintaxe da linguagem através da definição das notações utilizadas nas várias representações visuais da linguagem que compõem seus diagramas. Existem vários tipos de diagramas UML, cada um deles fazendo parte de um dos tipos de modelagem considerados em UML: modelagem estrutural, modelagem de casos de uso, modelagem comportamental e modelagem avançada [KOB01a][KOB01b[KOB01c].

- **A modelagem estrutural** - define uma visão de um sistema que dá ênfase nas estruturas de seus objetos (classificadores, relacionamentos, atributos e operações). Seus principais diagramas são: os diagramas estruturas estáticos (diagrama de classes e diagrama de objetos) e os diagramas de implementação (diagrama de componentes e diagrama de distribuição (*deployments*)) .

- **A modelagem de casos de uso** - define uma visão para um sistema, do ponto de vista de seus usuários externos. Esta visão é normalmente particionada em modelos de casos de uso que representam uma parcela da funcionalidade do sistema. Os casos de uso são considerados como transações que são importantes do ponto de vista dos usuários (atores). Seus principais diagramas são: os diagramas e as descrições de casos de uso.

- **A modelagem comportamental** - define uma visão comportamental para um sistema que está sendo especificado. Divide-se em três partes: as interações e colaborações, as trasições de estados (*statecharts*) e os diagramas de atividades. Seus principais diagramas são: o diagrama de seqüência, o diagrama de colaborações (ao nível da especificação e ao nível das instâncias), o diagrama de estados e o diagrama de atividades.

[4] *Na meta-modelagem estrita, cada elemento de um modelo do nível M_n é uma instância de, exatamente, um elemento de um modelo no nível M_{n+1}. Já na meta-modelagem não-estrita, um modelo do nível M_n é uma instância de um outro modelo no nível M_{n+1} sem a correspondência um a um de seus elementos internos.*

- **A modelagem avançada –** considera quatro aspectos adicionais para complementar as anteriores: a gerência de modelos, mecanismos de extensão, os perfis e a linguagens de restrições em objetos (OCL).

    - **gerência de modelos** - está relacionada com a forma de agrupar e organizar os artefatos UML em pacotes (agrupamentos genéricos de elementos de um modelo), modelos (tipo de pacote para agrupar uma visão de um sistema físico) e sub-sistemas (tipo de pacote para agrupar unidades comportamentais de um sistema físico).

    - **mecanismos de extensão –** permitem que os projetistas de modelos refinem a semântica da UML para um domínio específico. Refinar a semântica significa refinar o meta-modelo UML e seus artefatos para especializá-los dentro destes domínios. Estes refinamentos, entretanto, não podem violar algumas regras. Por exemplo, elementos novos devem ser criados, necessariamente, a partir de elementos já existentes. Desta forma, o meta-modelo UML é usado como uma meta-linguagem, mesmo fazendo parte da arquitetura MOF (onde o meta-meta-modelo MOF é a meta-linguagem). Os mecanismos básicos são: os estereótipos (*stereotypes*), as restrições (*constraints*) e os valores etiquetados (*tagged values*).

    - **Perfís UML –** um perfil UML é um pacote de elementos relacionados com extensibilidade que capturam padrões de variação e uso para um domínio específico. São exemplos de perfis UML: o perfil UML EDOC, o perfil UML para Real time, etc.

    - **Linguagem OCL –** é uma linguagem textual para descrever restrições que é usada na especificação UML. Trata-se de uma linguagem baseada em expressões que estão limitadas ao escopo do modelo ao qual fazem referência.

Como exemplo de uso conjunto do meta-modelo UML e do meta-meta-modelo MOF, a Figura 2-11 mostra como ambos podem se relacionar para a criação de um elemento de um modelo UML.

Na figura, imagine que um projetista faça uso de uma ferramenta de modelagem gráfica UML para criar um pacote de definições (*User's model*) com alguns elementos de sua preferência. Considere apenas um destes elementos, o que descreve um objeto chamado *Account*. Observe-se que os elementos do meta-modelo UML (artefatos do meta-modelo) são representados, dentro de um repositório, por artefatos do meta-meta-modelo MOF, provavelmente em tempo de compilação. E uma vez disponibilizada a ferramenta, os usuários podem fazer uso de ícones (de uma interface gráfica) representando os artefatos de modelagem para

criar instâncias dos componentes do meta-modelo UML dentro de um novo modelo definido pelo usuário projetista. Dentro da instância de artefato *Account*, novas instâncias de outros elementos são criadas e associadas a *Account* para representar uma classe com um atributo(*name*) e dois métodos (*balance() e getName()*). Quando este modelo é transformado em um programa para fazer parte do nível de informações, ele é representado por um objeto com valores inicializados internamente.

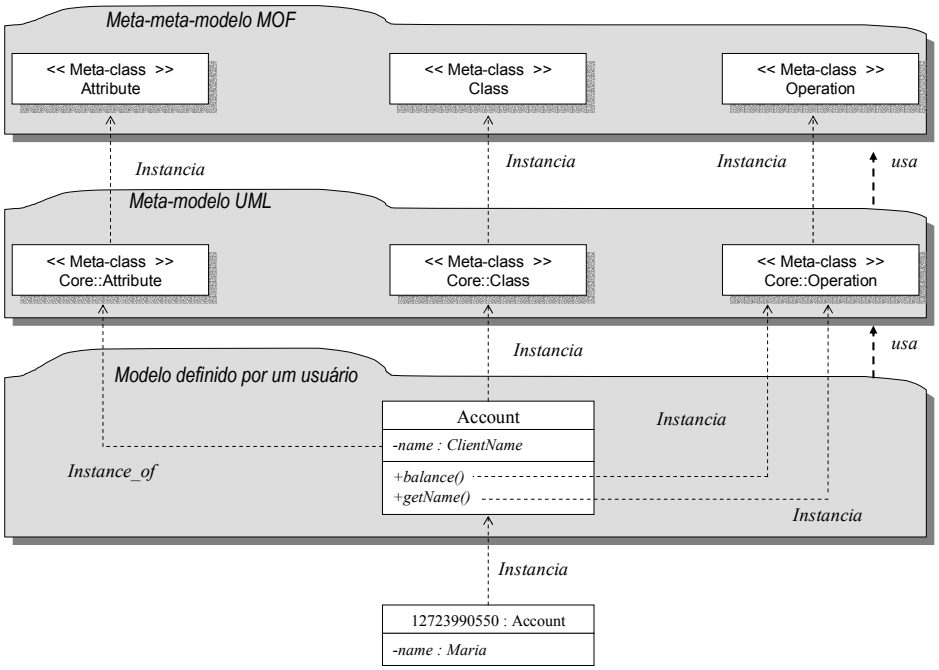

Figura 2-11 - Meta-modelo UML na arquitetura MOF

2.4.3. A arquitetura MDA

Desde sua criação, em 1989, a OMG tem tido como principal objetivo contribuir para a solução de problemas de integração através da definição de especificações voltadas para interoperabilidade, que sejam abertas e independentes de fornecedores ou fabricantes específicos. A base arquitetural para as especificações que têm sido definidas desde então, é a arquitetura OMA (*Object Management Architecture*). O principal componente da OMA, a ORB, foi especificado logo em seguida, através da especificação CORBA (*Common Object Request Broker Architecture*) e forneceu o contexto no qual deveriam ser tratadas todas as questões relacionadas com interoperabilidade.

No entanto, a partir de 1997, uma série de questões levaram à necessidade de especificações que, por definição, não estavam restritas ou não precisavam estar restritas ao contexto definido pela arquitetura OMA e seu padrão de ORB

(CORBA). Entre as especificações que emergiriam destas questões, estavam: as especificações *Unified Modeling Language* (UML) e a *Meta-Object Facility* (MOF), e posteriormente, a *XML Metadata Interchange* (XMI) e a *Common Warehouse Metamodel* (CWM).

Com a posterior adoção destas especificações como padrões formais da OMG e o conseqüente surgimento de implementações que serviram de provas de conceitos de sucesso [DST00], ficou evidente a necessidade de expansão do contexto de interoperabilidade usado pela OMG [OMG00f]. Passou-se também a abordar a questão da integração do ponto de vista de todo ciclo de vida dos sistemas: da modelagem de negócios para o projeto de sistemas, para construção de componentes, para montagem, integração, evolução e gerenciamento. Este novo contexto de interoperabilidade é que a OMG denominou de *Model-driven Architecture* (MDA) [OMG01k] [OMG03d].

A arquitetura MDA apresentada em seu principal documento na OMG [OMG01k] os conceitos, princípios, modelos e os relacionamentos existentes entre a arquitetura e os demais padrões da OMG.

A arquitetura MDA aborda o problema da interoperabilidade em sistemas IT através da separação da especificação da funcionalidade de um sistema da especificação da implementação desta funcionalidade em uma plataforma específica. Desta forma, o mesmo modelo funcional poderia ser usado em várias plataformas de implementações diferentes através de mapeamentos apropriados que relacionassem conceitos em diferentes modelos.

A Tabela 2-4 mostra alguns destes conceitos fundamentais, bem como suas definições, tendo em vista o amplo uso que faremos destas definições ao longo dos capítulos da tese.

Tabela 2-4 - Alguns conceitos básicos na visão da arquitetura MDA

Conceito	Interpretação em MDA
Modelo	É a representação de uma parte da funcionalidade, estrutura e/ou comportamento de um sistema. Deve ter uma descrição formal, no sentido de que cada elemento tenha uma semântica bem definida e esteja definido em conformidade com a especificação MOF.
Abstração	É a descrição de algo que omite alguns detalhes irrelevantes para um contexto específico.
Refinamento	É uma descrição detalhada que está em conformidade com uma abstração correspondente.
Plataforma	É uma infra-estrutura de software implementada com uma tecnologia específica (Unix, CORBA, Windows, etc) sobre uma tecnologia de hardware específica.
Infra-estrutura	É um conjunto de partes de software e/ou hardware assumidos como presentes por usuários quando estes desenvolvem um artefato de software.
Visão[5]	É uma representação de um sistema como um todo, a partir da perspectiva de um conjunto relacionado de interesses (*concerns*).
Ponto de Vista	É a especificação de convenções para a construção e uso de uma *Visão*, ou seja,

[5] Segundo IEEE STD 1471-2000

	um *template,* a partir do qual se desenvolvem visões pelo estabelecimento de propósitos e audiência, bem como as técnicas para sua criação e análise.
Mapeamento	É um conjunto de regras e técnicas usados para modificar um modelo e criar um outro modelo a partir do anterior.
Traceability	É a capacidade de navegar através de links que descrevem o histórico de artefatos de software ao longo de diferentes níveis de modelagem (análise, projeto, implementação).
Modelo Independente de Plataforma	Em inglês: PIM (*Platform Independent Model*). Fornece especificações formais da estrutura e funcionalidade de um sistema se abstraindo de detalhes técnicos.
Modelo Específico de Plataforma	Em inglês: PSM (*Platform Specific Model*). É expresso em termos do modelo de especificação da plataforma específica. Um PSM usa os conceitos próprios de uma plataforma específica, tais como: modelo de componentes, mecanismos de exceção, tipos de parâmetros, etc.

2.4.3.1 A MDA e as outras especificações da OMG

A arquitetura MDA compreende três grupos de especificações: o núcleo, os serviços essenciais e as especificações de domínio (Figura 2-12). O núcleo (*Core*) é baseado nos padrões de modelagem da OMG (UML, MOF e CWM) e é formado por vários modelos (*Core models*) que são, na verdade, perfis UML que especializam o meta-modelo UML para algumas áreas.

- Sistemas distribuídos Empresariais — Perfil UML para EDOC [OMG02b];
- Sistemas de Tempo Real — Perfil UML para RT [OMG02k];

Inicialmente, apenas os modelos acima faziam parte do conjunto de modelos do núcleo, posteriormente, outros perfis UML foram acrescentados [IYE01] :

- Sistemas Integrados Empresariais — Perfil UML para EAI [OMG02f];
- Engenharia de Processos de Software — Perfil UML para SPEM [OMG02j];

Cada um dos perfis UML acima, representa as características comuns de todas as plataformas de *middleware* apropriadas para sua categoria, mas será independente de qualquer plataforma específica. Desta forma, o primeiro passo para se construir uma aplicação MDA seria a criação de um modelo PIM da aplicação, expresso em UML através dos artefatos definidos no perfil UML apropriado. Uma vez definido e acabado o processo de definição do PIM, especialistas nas plataformas específicas poderiam transformar este modelo em outro que fosse específico para uma plataforma como: CCM, EJB ou COM+.

Os serviços essenciais (*pervasive services*) são mostrados no anel mais externo da Figura 2-12. Todas as aplicações, independente de seus contextos dependem de algum, ou alguns, serviços essenciais. Entre os vários que devem estar disponíveis

temos: Serviço de diretórios, de tratamento de eventos, de persistência, de transações, de segurança, etc.

Quando estes serviços são definidos e construídos sobre uma plataforma em particular, eles necessariamente levam em consideração as características que os restringem para a plataforma escolhida. Se por um lado isto pode implicar em otimizar alguns aspectos dos serviços, por outro lado, entretanto limitam sua utilização em plataformas diferentes. Pensando nisto, a OMG colocou os serviços essenciais como parte da MDA, o que implica na definição de PIMs UML para descrevê-los de forma independente das implementações.

No nível de abstração de um modelo PIM de componente de negócio , os serviços são apenas visíveis no nível mais alto . Quando o modelo é mapeado para uma plataforma em particular, o código que será gerado automaticamente fará uso dos serviços essenciais de forma transparente para o desenvolvedor.

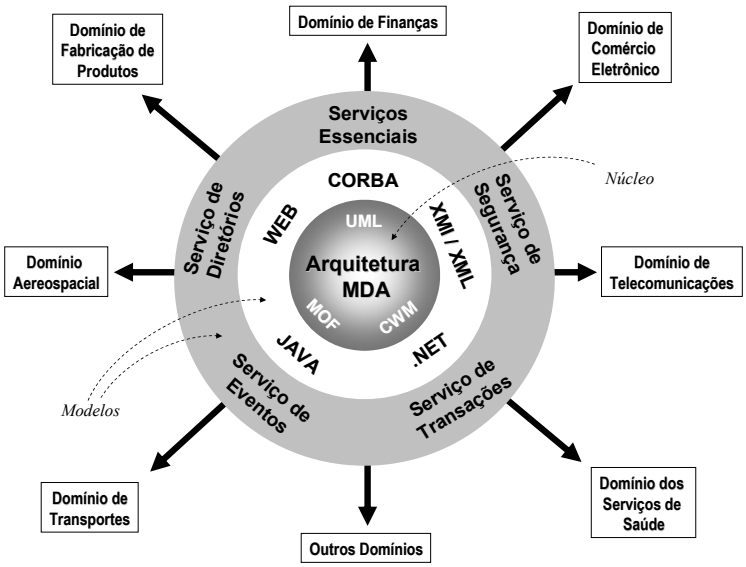

Figura 2-12 - A arquitetura MDA.

Na Figura 2-12, uma série de setas representam a aplicação da arquitetura MDA nos vários mercados verticais tratados pelas DTFs (*Domain Task Forces*) da OMG. Inicialmente, as especificações de serviços e facilidades nestes domínios eram feitas através da definição de interfaces em IDL e das descrições complementares da semântica envolvida e linguagem natural escrita. Esta forma de especificação é útil até certo ponto, mas é também inferior a descrição de um serviço através de perfil UML associado com um PIM, pois ao se usar UML, pode-se formalizar a semântica de uma forma mais consistente, bem como fazer uso dos recursos de diagramação para melhor definir interfaces e suas interações.

2.4.3.2 O conceito de Perfis UML e a Arquitetura MDA

Como já foi apresentado anteriormente, um perfil UML [OMG00g] é uma especificação que especializa o meta-modelo UML para um domínio específico de referência. Por exemplo: para o domínio de objetos de negócios temos o perfil EDOC. Esta especialização é feita através dos mecanismos de extensão da especificação UML tais como: estereótipos (*stereotypes), valores etiquetados (tagged values), definições de etiquetas (tagdefinitions) e restrições (constraints)*. Além disso, todo perfil UML é intercambiável através do uso da especificação XMI que define um subconjunto de meta-classes UML e pode ser especializado e composto, bem como associado a pacotes UML.

A Figura 2-13 mostra uma visão geral do meta-modelo UML [OMG00b] (parte a) e um exemplo de perfil UML, onde o artefato: *Core::Class*, do meta-modelo UML (em seu pacote interno *Core)* é estereotipado para dar origem a dois novos artefatos: *CORBAUserDefinedType* e *CORBAType* (parte b). Estes estereótipos são posteriormente especializados para criar outros artefatos dentro do perfil: *CORBAObjectType*, *CORBAValue*, *CORBACustomValue* e *CORBAInterface*. Neste exemplo, os estereótipos criados devem ser interpretados, também como artefatos *Core::Class*, só que agora, com semântica especificamente definida dentro de seu domínio e eventualmente com ícone próprio.

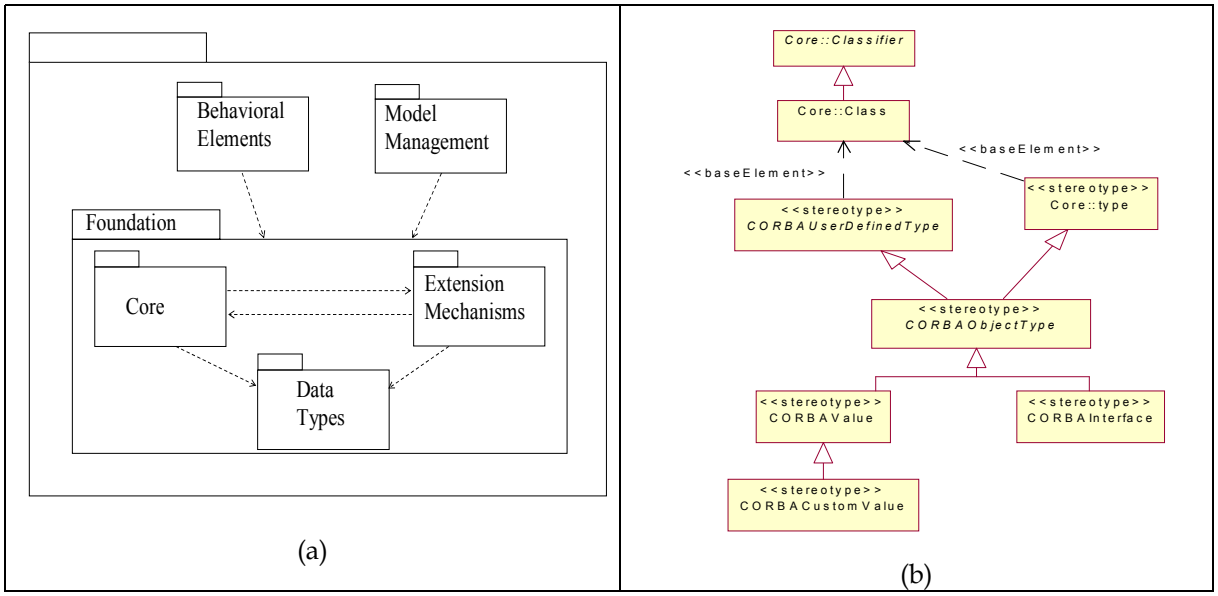

Figura 2-13 –Meta-modelo UML [OMG01f] (a) e Exemplo de Perfil UML [OMG01o] (b)

2.4.3.3 A MDA e o Ciclo de Vida dos Sistemas

Os sistemas de informação têm sido historicamente desenvolvidos, gerenciados e integrados através de inúmeras metodologias, ferramentas e

middlewares. Nos últimos anos, entretanto, têm-se observado uma convergência em torno de especificações e implementações relacionadas que apresentem um modelo semântico mais completo, bem como, para que disponham de padrões de intercâmbio de metadados. Certamente que esta convergência foi influenciada por organismos como a OMG e o W3C, pois as especificações: CORBA, UML, MOF, XMI e CWM, que são da OMG e XML e *XML-Schema,* que são da W3C têm desempenhado um papel importante neste contexto.

Um aspecto importante da MDA é que ela aborda todo o ciclo de vida de um software: análise, projeto, programação e gerenciamento. O núcleo do MDA é baseado em tecnologias da OMG que são usadas para descrever PIMs. Um PIM pode ser refinado diversas vezes até que o nível de descrição adequado seja atingido. Só então, a infra-estrutura é levada em consideração e o PIM é transformado em um PSM. O PSM por sua vez, também é refinado até atingir um nível desejado de refinamento em sua descrição. As linguagens de modelagem e intercâmbio têm papeis importantes neste ciclo de vida em que PIMs, PSMs e mapeamentos são desenvolvidos e usados para construir uma solução final completa. Em termos gerais pode-se descrever estes papéis da seguinte maneira:

- **UML (*Unified Modeling Language*) –** desempenha o importante papel de modelar arquiteturas, objetos, interações entre objetos, aspectos de modelagem de dados no ciclo de vida, bem como os aspectos de projeto de desenvolvimentos baseados em componentes, como construção e montagem de componentes.

- **XMI (*XML Metadata Interchange*) –** XMI é o mecanismo de intercâmbio padrão usado entre várias ferramentas, repositórios e *middleware*. XMI pode também ser usado para automaticamente produzir DTDs a partir de modelos UML e MOF, fornecendo um mecanismo de serialização XML para os artefatos dos modelos. Para cada conversão diferente, é necessário um diferente arquivo DTD. Por exemplo, para ser capaz de converter modelos UML para XML, é necessário dispor da DTD UML-XMI e para ser capaz de converter modelos MOF para XML, é necessário dispor da DTD MOF-XMI, etc.

- **MOF (***Meta-Object Facility***) –** MOF fornece a base comum de modelagem entre UML e CWM. Este fundamento comum permite o intercâmbio e interoperabilidade entre estes dois padrões, e é o mecanismo através do qual os modelos são analisados em XMI. MOF também define interfaces de gerência dos modelos armazenados e suas instâncias ao longo do ciclo de vida dos sistemas.

- **CWM (***Common Warehouse Metamodel***) –** é o padrão da OMG para *data warehouses*. Ele cobre o ciclo de vida inteiro do processo de projeto,

construção e gerência de aplicações para *data warehouses*. É o melhor exemplo até o momento, de aplicação do paradigma de MDA em uma área específica.

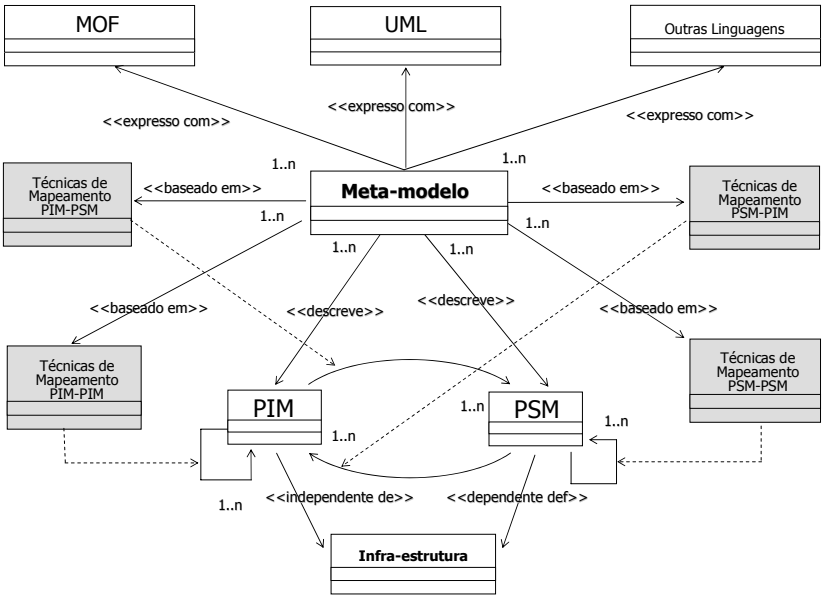

Figura 2-14 – Meta-modelo para a arquitetura MDA.

A Figura 2-14 apresenta um meta-modelo para ilustrar os principais conceitos e relacionamentos em MDA. No centro da figura temos um meta-modelo que é definido em termos destes conceitos. Ele é expresso através de artefatos definidos em uma ou mais das linguagens de modelagem (UML, MOF, etc) para se constituir em um PIM ou em um PSM. PIMs e PSMs são perfis UML que podem ser refinados ou transformados pelos mapeamentos disponíveis na arquitetura. Os mapeamentos são:

- **PIM para PIM** – São mapeamentos usados quando não há necessidade para a descrição de informações específicas de plataforma. Ex.: especializações de domínio, geração de modelos de projeto à partir de modelos que representam os requisitos da fase de análise, etc.

- **PIM para PSM** – São mapeamentos usados quando já se atingiu um nível de refinamento desejado para um modelo. Servem para gerar perfis UML que estão relacionados com as informações dependentes de plataforma. Ex.: modelos de componentes mapeados para EJB, CCM, DCOM, etc.

- **PSM para PSM –** São mapeamentos usados para refinar modelos específicos para implementações concretas de plataformas. Por exemplo,

da especificação EJB 1.1 para uma implementação concreta no produto Weblogic 6.1.

- **PSM para PIM -** São mapeamentos usados para criar modelos abstratos a partir de implementações existentes em uma tecnologia particular.

Os conceitos de PIM e PSM, juntamente com o conceito de CIM [OMG02l] são denominados de perspectivas MDA (MDA *Viewpoints*). Estas perspectivas são abstrações que separam áreas de interesse no âmbito da arquitetura. A seção seguinte apresenta em detalhes o conceito de CIM e o contexto em que está inserido.

2.4.3.4 O Conceito de CIM

Um conceito que vem sendo foco de atenção crescente no processo de amadurecimento da arquitetura MDA é o conceito de CIM (*Computational Independent Business Model*). Um CIM é um modelo que captura os requisitos de negócio de um sistema em uma linguagem própria do negócio e voltada para os usuários do sistema. Isto fornece uma base para o projeto funcional e um meio importante para as pessoas do negócio validarem um projeto [OMG02l].

Antes de continuar, vamos relembrar, neste ponto, alguns conceitos que dizem respeito a RM-ODP e a MOF. O modelo de referência RM-ODP descreve uma arquitetura de modelagem de sistemas com cinco perspectivas (empresarial, informacional, computacional, de engenharia e técnica). Estas perspectivas representam diferentes áreas de conhecimento necessárias para desenvolver um modelo de sistema de software segundo o modelo RM-ODP. Elas também representam diferentes abstrações para o sistema modelado em áreas de interesses distintas. Cada uma das abstrações permite aos projetistas do sistema darem ênfase aos elementos e relacionamentos que estão dentro da área de conhecimento da perspectiva em questão. A Figura 2-15 ilustra esta discussão.

Um projetista pode examinar um modelo considerando também diferentes visualizações (*views*). Estas visualizações são representações gráficas associadas a elementos de modelos. A forma gráfica e os relacionamentos são descritos como notação. O projetista pode utilizar múltiplas visualizações para criar e modificar elementos de modelo para abordar a área de interesse ou conhecimento deste projetista. Basicamente, as perspectivas RM-ODP são diferentes modelos do sistema que capturam a maneira com a qual os projetistas vêem o sistema. Ao mesmo tempo, é essencial que o modelo do sistema, como um todo, seja consistente. Isto requer que certas correspondências entre os modelos das perspectivas tenham que ser estabelecidas e impostas. Conseqüentemente, o ambiente de modelagem tem que integrar estes modelos de perspectivas, de forma

que as mudanças em uma delas possam ser refletidas nas demais, quando isto for necessário.

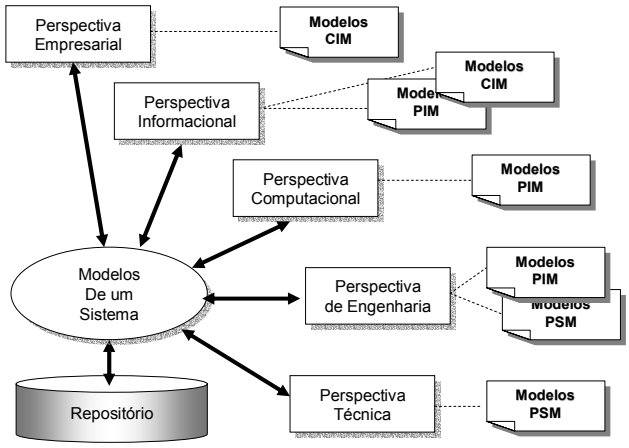

Figura 2-15 - Perspectivas do RM-ODP versus Modelos MDA

Neste contexto, os repositórios (como os baseados na especificação MOF) podem desempenhar o importante papel de armazenar os modelos criados nestas diferentes perspectivas e também para viabilizar os diversos tipos de manipulações que são feitas sobre os mesmos, tais como: mapeamentos, transformações, alterações na estrutura, etc. Estas manipulações, via de regra, são parte da infra-estrutura que concretiza o meta-modelo conceitual para MDA.

Esta Infra-estrutura deve permitir a integração de modelos, a armazenagem destes em repositórios compartilhados e o intercâmbio de modelos e meta-modelos com diferentes ferramentas baseadas em MOF e/ou XMI. Deste modo, os usuários de modelos e ferramentas MDA não ficam restritos ao uso de um único conjunto de ferramentas integradas, mas podem usar livremente outras alternativas, à medida que novos recursos sejam oferecidos nas mesmas.

A arquitetura MDA tem se norteado pela visão de que modelos possam ser usados para especificar sistemas de forma tecnologicamente independente para depois serem transformados para uma plataforma tecnológica especifica quando necessário, sem mencionar explicitamente atividades como as verificações de consistência de modelos. O modelo de referência RM-ODP reconhece a necessidade da consistência entre as várias perspectivas e suas linguagens de definição. A consistência semântica é obtida através do compartilhamento de abstrações de projeto de um sistema. Por exemplo, três diferentes linguagens de modelagem definidas pelos pontos de vista de engenharia, computacional e tecnológico poderiam compartilhar o conceito de interface, com o mesmo significado em todas as três linguagens. O projeto de todas elas incorporaria o mesmo elemento de projeto. Ao mesmo tempo, este elemento de projeto da linguagem pode ter diferentes relacionamentos com outros elementos de modelo

nos diferentes pontos de vista. Ele pode também ser expresso em diferentes visualizações dos modelos, com diferentes ícones ou expressões textuais. Em alguns casos, ele seria até mesmo identificado com diferentes termos (*aliases*) que melhor se adequariam a área de interesse.

O fato destas linguagens das perspectivas compartilharem os mesmos conceitos abstratos garante que os modelos produzidos por elas incorporem a mesma semântica, mesmo que os usuários possam vê-las de formas diferentes.

2.4.3.5 Transformações em Modelos

Como se pode observar das seções anteriores, os mapeamentos representam um dos conceitos básicos das arquiteturas baseadas em modelos. Eles permitem refinamentos em PIMs e PSMs, mapeamentos tecnológicos e processos de engenharia reversa em PSMs. No entanto, apesar de serem de fácil entendimento na essência, são na verdade um dos conceitos onde se observa o maior número de mal entendidos na bibliografia estudada. O primeiro mal entendido que se observa é quando procuramos diferençar o conceito de mapeamento de outro conceito semelhante: as transformações em modelos. Segundo a definição apresentada na Tabela 2-4, um mapeamento é: "*um conjunto de regras e técnicas usado para modificar um modelo para criar um outro modelo*." [OMG01k], enquanto que uma transformação é um conceito mais abrangente, que além dos mapeamentos, envolve transformações de modelos em texto e transformações genéricas em metadados. No entanto, muitos são os casos na literatura em que os termos são tratados indistintamente.

Há basicamente três grupos de transformações: as transformações em meta-modelos, as transformações em tipos de modelos e as transformações em instâncias de modelos, cada uma com as seguintes características:

- **Transformações em meta-modelos –** são baseadas em especificações que relacionam duas formas de representação de linguagens no plano da meta-linguagem MOF. A idéia é alterar a estrutura e/ou comportamento descrito na primeira (que já está no repositório) para gerar a segunda que também virá a ser um meta-.modelo descrito em MOF. Este grupo de transformações tem como variantes as transformações que alteram um meta-modelo MOF para a representação textual do mesmo, tais como: DTD XML (XMI), HUTN, etc. Ver Figura 2-17.

- **Transformações em tipos de modelos –** são baseados em especificações que relacionam tipos de modelos descritos em uma linguagem PIM, com tipos de modelos PSMs, descritos com uma linguagem PSM. São aplicadas em modelos PIMs para que passem a ser modelos PSM, descritos por tipos da linguagem PSM escolhida. São também conhecidas com *marcas* ou

anotações. Este grupo de transformações tem como variantes as transformações que alteram um modelo PIM ou PSM para a representação textual do mesmo, tais como: arquivo XML, etc.

- **Transformações em instâncias de modelos** – são baseadas em especificações que relacionam elementos de modelos de um PIM, que devem ser transformados de uma forma em particular diferente, dependendo do PSM que será usado em uma transformação em tipos de modelos.

Um dos primeiros trabalhos em que o conceito de transformações em modelos foi colocado a prova foi o da ferramenta dMOF [DST00]. No dMOF os projetistas podiam definir qualquer tipo de meta-modelo usando a linguagem MODL em um arquivo texto, submeter este arquivo a um compilador que o transformava em meta-objetos MOF (*modl2mof*) e a partir deste instante, várias ferramentas poderiam ser usadas para transformar os meta-objetos MOF em elementos de um arquivo CORBA IDL (*mof2idl*), em elementos de um arquivo XML DTD ou em elementos de arquivos fonte Java que implementariam um servidor de repositório para os meta-objetos especificados (*mof2moflets*). Estes arquivos fontes Java seriam as implementações das interfaces mostradas na Figura 2-8 e na Figura 2-9 e, no linguajar próprio da ferramenta, são denominados de *Moflets*. Ver Figura 2-16.

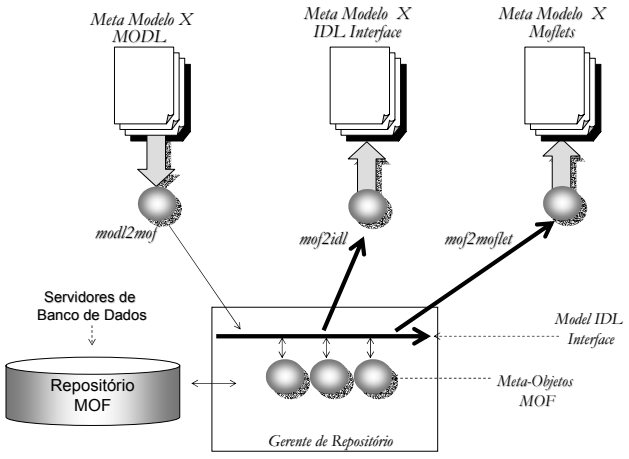

Figura 2-16 - Exemplos de Transformações no DSTC dMOF

Transformações como as mostrados na Figura 2-16, são chamadas de Modelo-Texto e normalmente são usadas no processo de geração de código fonte final e se baseiam em *templates,* ou coleções de *templates*, para nortear as transformações.

Existem várias abordagens para as implementações de transformações. [OMG00c], por exemplo, define um modelo para descrever transformações e dois

tipos de transformações: Transformações via *caixas pretas* (*black-boxes*) e via *caixas brancas* (white boxes). As primeiras associam as entradas e saídas de uma transformação sem descrever como uma é obtida a partir das demais, enquanto que nas outras, caixas-brancas, existe uma descrição detalhada das alterações sofridas nas entradas para se obter as saídas. Infelizmente, [OMG00c] apenas apresenta um modelo genérico, em UML, para as transformações sem oferecer um mecanismo real para a implementação destas transformações.

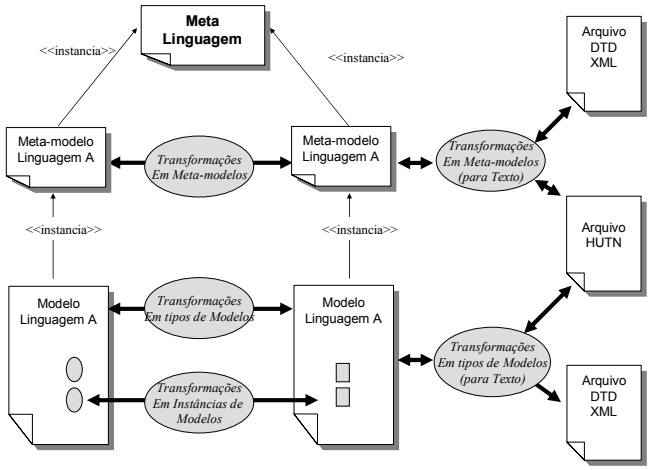

Figura 2-17 - Tipos de Transformações

A questão das Transformações em modelos e meta-modelos é um tema ainda em definição dentro da OMG. Apesar de mapeamentos manuais já serem uma realidade, muito ainda há para se definir com relação às automatizações destes e de outras transformações importantes.

2.5 Resumo

Este capítulo apresentou os principais conceitos e tecnologias necessárias para o entendimento do conteúdo dos demais capítulos. Alguns dos itens apresentam informações genéricas e introdutórias, como é o caso dos sistemas reflexivos e as técnicas de meta-modelagem, que têm seus fundamentos comparados para confirmar o que já havia sido observado em alguns trabalhos [CRA97][COS01]. Outros itens, por sua vez, apresentam um conteúdo mais abrangente e um papel de maior destaque na proposta da tese, pois fazem parte da essência do problema abordado na tese, como é o caso dos padrões da OMG relacionados com metadados, modelagem e meta-modelagem (MOF, JMI, XMI e UML) e a própria arquitetura MDA.

Capítulo 3
Sistemas de Gerência de Workflows

3.1 Introdução

Este capítulo apresenta um resumo dos fundamentos da tecnologia de gerência de workflows que são necessários para completar a descrição do contexto iniciada no capítulo anterior. O objetivo é fornecer a segunda parte da terminologia necessária para o entendimento dos demais capítulos da tese.

Na seção 3.2, os principais conceitos relacionados com a tecnologia de workflows são apresentados, bem como os sistemas de gerência de workflows, seus principais componentes e aplicações; na seção 3.3, os primeiros padrões e propostas são discutidos, bem como uma breve comparação entre eles; na seção 3.4, os fundamentos da tecnologia são analisados à luz das novas idéias apresentadas pela arquitetura MDA e pela especificação EDOC; e por fim, na seção 3.5, um breve resumo é feito para adicionar algumas considerações finais relacionadas com o capítulo.

3.2 Conceitos Básicos

3.2.1. Processo de Negócio

O termo *Processo de Negócio* pode ser definido como um conjunto de procedimentos ou atividades que coletivamente realizam um objetivo de negócio dentro do contexto de uma estrutura organizacional que define papéis e relacionamentos funcionais para os participantes do processo [WFM99]. Os processos de negócios podem ser vistos como descrições das atividades de uma organização através de jargão próprio do negócio em que estão inseridos. Estas descrições via de regra são implementadas através de processos organizacionais que agregam valor a serviços ou produtos de uma empresa. Apesar do conceito de processo de negócio envolver o conceito de processo organizacional, ele é mais

abrangente por considerar aspectos relacionados com o contrato de negócio entre cliente e a empresa e com a satisfação do cliente neste contrato [GEO95].

Automatizar um processo de negócio significa basicamente utilizar sistemas de software especializados para projetar, executar e acompanhar este processo (normalmente um processo manual ou parcialmente manual)com o objetivo de reduzir custos e aumentar sua eficiência e consistência. Existem várias formas e tecnologias para a automatização de processos de negócios. As primeiras soluções que surgiram faziam uso de sistemas monolíticos que embutiam os fluxos de dados e de controle na própria lógica de seus programas. Apesar de obterem bons resultados inicialmente, em comparação com processos manuais, possuíam o inconveniente de serem de difícil manutenção e pouco flexíveis uma vez que, sempre que um processo de negócio era modificado, havia a necessidade de mudanças na lógica do sistema de software usado. Com o tempo novas tecnologias surgiram e entre elas, está a tecnologia de workflows que procura modelar os processos de negócios como tipos de dados especiais que são definidos de forma independente dos sistemas que os utilizavam permitindo assim, maior flexibilidade para as soluções que procuravam automatizar os processos de negócios de uma organização.

3.2.2. Workflow

O termo *Workflow* é freqüentemente usado para fazer referência a uma série de áreas de trabalho relacionadas, tais como: automação de processos de negócios, reengenharia de processos, especificação, implementação, modelagem ou gerência de processos de negócios. No entanto, a rigor, o termo representa apenas um conceito utilizado por todas essas áreas. Uma das primeiras definições para *workflow* foi dada pela WfMC (*Workflow Management Coalition*) em seu glossário de termos. Na ocasião, o termo era visto como:

"*A automação de um processo de negócio, no todo ou em parte, durante a qual documentos, informações ou tarefas são passados de um participante para outro, para que ações sejam tomadas de acordo com um conjunto de regras procedimentais*" [WfMC99].

No início, o contexto de aplicação para os workflows estava praticamente limitado à coordenação de atividades envolvendo pessoas (participantes). Dessa forma, o trabalho relacionado com um processo de negócio era passado de um participante a outro até que todos pudessem acrescentar sua parcela de trabalho ao todo. O principal benefício, na ocasião, era a automação da entrega do trabalho. Como conseqüência do processo de amadurecimento da tecnologia, níveis maiores de automação foram considerados e passaram a incluir sistemas dedicados para a execução de workflows. Estes sistemas foram denominados de *Sistemas de gerência de workflows*.

3.2.3. Sistemas de Gerência de Workflows

O passo seguinte no processo de maturação da tecnologia foi a automatização dos próprios processos de negócios e a utilização de sistemas de gerência dedicados para controlá-los. Estes sistemas foram denominados de *Sistemas de Gerência de Workflows*. Esta automatização implicou na criação de especificações formais que pudessem representar um processo de negócio através de um conceito denominado de *Definição de Processo de Negócio*. Uma definição de processo é a representação de um processo de negócio em uma forma adequada para permitir manipulações automatizadas por um sistema de gerência. Ela consiste de uma rede de atividades, seus relacionamentos, seus critérios de início e término e informações sobre as atividades individuais que compõem o processo (participantes, aplicações e dados de contexto).

Um sistema de gerência de workflows (WFMS – *Workflow Management System*) é um sistema de software que implementa um esquema de gerência de workflows para permitir que os usuários do mesmo possam definir, criar e gerenciar a execução de workflows, através de entidades capazes de interpretar uma definição de processo formal, de interagir com participantes definidos para o processo em execução e fazer uso de ferramentas e aplicações disponíveis para o sistema.

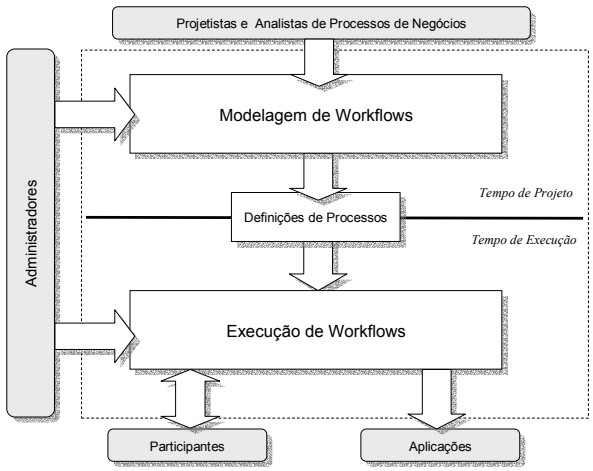

Figura 3-1 - Principais componentes de um sistema de gerência de workflows.

Um WFMS possui dois grupos de componentes funcionais básicos: os componentes de modelagem e os componentes de execução (Figura 3-1). Os primeiros estão relacionados com a criação de modelos que representam as definições de processos, os participantes, os dados, as aplicações e os aspectos organizacionais necessários para a correta definição de um modelo de processo de workflow. São utilizados por usuários do WFMS que projetam, analisam e aperfeiçoam estes modelos; os segundos, por sua vez, dão apoio à correta execução de um processo de workflow segundo as definições de tipos definidos nos

modelos. São utilizados por entidades que criam instâncias de execução para os processos e atividades definidos no modelo criado através do primeiro grupo de componentes. Estes componentes de execução são usados pelos usuários que inicializam uma determinada definição de processo, para que esta defina os passos necessários para a execução de atividades manuais ou automatizadas. No caso das atividades envolvendo participantes, são necessárias listas de trabalhos (worklists) para armazenar os itens de trabalhos[6] que representam uma tarefa a ser executada pelo participante em questão dentro da atividade que lhe está associada.

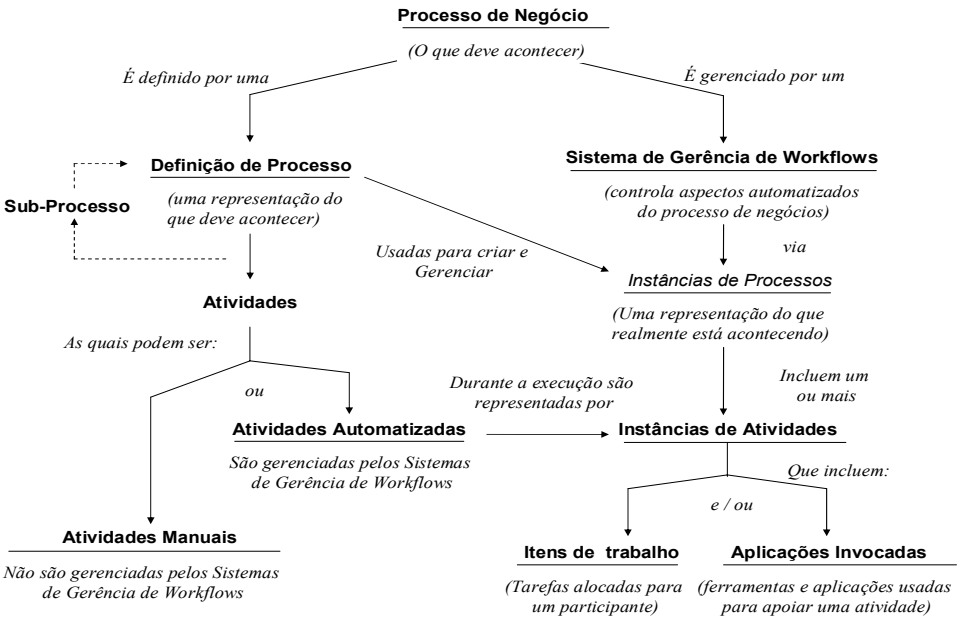

Figura 3-2 - Relacionamentos entre conceitos Básicos [WFM99].

A Figura 3-2 mostra os principais relacionamentos entre os conceitos necessários para representar um processo de negócio através de uma definição de processo que é interpretada e executada pelo WFMS.

3.3 Os Primeiros Padrões

Os primeiros padrões relacionados com workflows surgiram na WfMC (*Workflow Management Coalition*) e na OMG (*Object Management Group*). As sub-seções seguintes apresentam as principais contribuições e especificações dos trabalhos de padronização desenvolvidos nestas duas instituições.

[6] *Item de trabalho – representação do trabalho a ser processado por um participante no contexto de uma atividade dentro de uma instância de processo.*

3.3.1. WfMC

A primeira iniciativa para criar padrões relacionados com workflow surgiu em meados dos anos noventa, com a proposta da WFMC que definia um modelo de referência para workflows. Este modelo descreve uma arquitetura com os principais componentes, interfaces e formatos de dados comumente observados na estrutura genérica de um sistema de gerência de workflows (Figura 3-2).

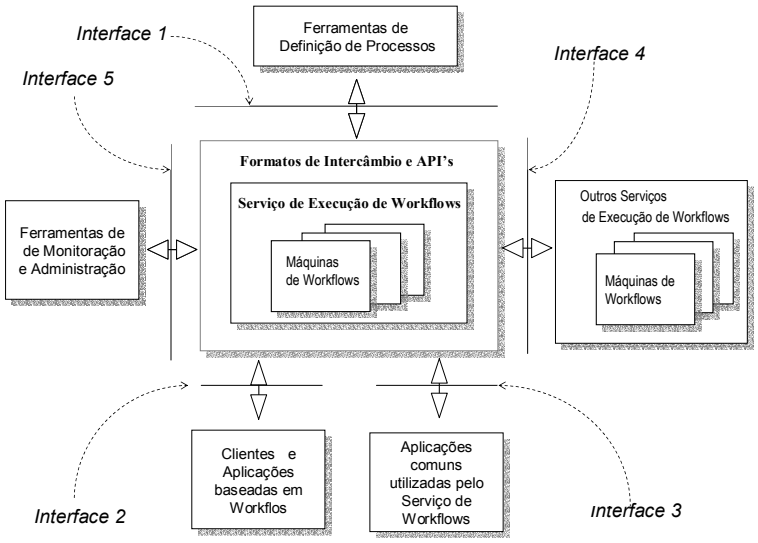

Figura 3-3 - Modelo de referência para *workflows* [WFM95].

A arquitetura da WFMC propõe um serviço de execução (*Workflow Enactment Service*) no qual os procedimentos de instanciação e ativação de processos são conduzidos através de máquinas de workflows (*Workflow Engines*) que são responsáveis pela interpretação e ativação de definições de processos específicas e suas interações com recursos externos. Estes recursos externos podem ser aplicações de um servidor de aplicações (*Invoked Applications*) ou participantes que interagem com a arquitetura através de aplicações clientes capazes de manipular itens de trabalhos produzidos pelas atividades que são executadas pelas máquinas de workflows (*Workflows Applications*). Além disso, a arquitetura prevê a interação do serviço de execução com ferramentas de administração e monitoração de funções e operações vitais do ambiente de execução; a interação com ferramentas de criação de definições de processos de workflows; e interoperação com outros serviços de execução para permitir a execução coordenada de processos complexos que levem em consideração serviços de execuções distintos para as diferentes partes de seus sub-processos.

O conjunto de cinco interfaces do modelo é denominado de WAPI (*Workflow Application Program Interface*) e constitui um conjunto de serviços pelos quais o

sistema de execução pode ser acessado para conduzir as interações com os componentes externos do modelo. As interfaces são:

- **Interface 1 –** é a interface entre o ambiente de modelagem (constituído por modelos de processos de negócios que estão sendo criados, analisados ou alterados por ferramentas de modelagem apropriadas) e o serviço de execução da arquitetura. A idéia é usar um meta-modelo para representar as definições de processos e informações relacionadas (participantes, aplicações, dados, estrutura organizacional), separar o ambiente de modelagem do ambiente de execução (para que diferentes implementações possam compartilhar a mesma forma de representar processos) e padronizar a forma de acessar as definições a partir do serviço de execução que vai utilizá-las [WFM99b].

- **Interface 2 –** é a interface entre o serviço de execução e os participantes que interagem dentro de um processo de negócio. Ela determina que as interações entre os dois lados sejam feitas através de listas de trabalho (*worklists*) que recebem os dados necessários para os participantes na forma de itens de trabalho (*work itens*), que são, por sua vez, tratados por aplicações especializadas na manipulação destas listas de trabalho (*worklist handlers*) ([WFM95b] e [WFM98a]).

- **Interface 3 –** esta interface define as interações entre o serviço de execução e as aplicações que são utilizadas sem a intervenção de participantes humanos. Ela leva em consideração a utilização de diferentes agentes de invocação que permitem a utilização de diferentes métodos de invocação dependendo do tipo de aplicação a ser utilizado (processos locais, *shell scripts*, objetos CORBA, chamada RPC, etc) ([WFM95b] e [WFM98a]).

- **Interface 4 –** esta interface define as diversas formas de interação entre dois serviços de execução distintos que devem interoperar para executar partes de processos de workflow complexas que envolvam vários domínios de execução. Cada domínio está associado ao ambiente de execução definido por um serviço de execução em conformidade com a arquitetura ([WFM99c], [WFM00a] e [WFM00b]).

- **Interface 5 –** é a interface entre o serviço de execução e o ambiente de administração deste serviço. Esta interface envolve a utilização de diversas ferramentas capazes de gerenciar e/ou monitorar áreas funcionais tais como: gerência de usuários, gerência de papéis, gerência de informações de auditoria, controle de recursos, supervisão de processos e etc [WFM98b].

Outro aspecto importante do modelo de referência da WFMC diz respeito ao intercâmbio de dados entre os componentes do mesmo. Os seguintes tipos de dados são identificados: dados de controle, dados relevantes do workflow e os dados de aplicação. O primeiro tipo compreende dados gerenciados internamente no sistema de execução para auxiliar no controle das várias instâncias de máquinas de workflows e seus estados (não são normalmente intercambiáveis); o segundo tipo compreende dados usados para determinar condições de transição e que podem afetar a escolha da próxima atividade a ser executada dentro do processo (são usados nas interfaces 2 e 3); e o terceiro tipo compreende dados específicos de aplicações invocadas que não estão acessíveis ao sistema de execução. Por exemplo, um determinado tipo de dado relevante de workflow que seja necessário para a execução de uma aplicação, pode necessitar que seu formato seja alterado para se adaptar aos requisitos de parâmetros de entrada exigidos por uma aplicação. Neste caso, os dados transformados seriam os dados de aplicação usados na interface 3.

O ponto forte do modelo de referência é o ganho de interoperabilidade advindo da separação do ambiente de modelagem do ambiente de execução e a padronização de interfaces entre os vários componentes que constituem este ambiente de execução. Seus pontos fracos são a falta de uma definição precisa da semântica das interfaces, nenhum suporte para transações e a ausência de especificações para federações de serviços dos ambientes de execução em diferentes domínios.

3.3.2. OMG

Os pontos fracos do modelo de referência da WFMC e a necessidade de se adequar a teoria dos sistemas de gerência de workflows ao contexto da arquitetura OMA (*Object Management Architecture*) foram determinantes para que a OMG publicasse em meados de 97 um *Request for Proposals* (RFP) com objetivo de especificar um *facility* para gerência de workflows baseado em CORBA (*Workflow Management Facility*) [OMG98a]. Entre os requisitos obrigatórios estavam:

- A definição de interfaces para o ambiente de execução – as interfaces deveriam ser especificadas em CORBA IDL (de forma precisa e funcionalmente completa);
- A definição de um meta-modelo para as interfaces – este modelo deve considerar os relacionamentos e ciclos de vida das instâncias das interfaces descritas;
- A definição de interfaces para monitoração – são interfaces para recuperar informações de status dos processos workflows, cujas instâncias estão em execução.

- A definição de interfaces para auditoria – são interfaces que permitem o acesso ao histórico de execução das instâncias dos processos e
- Suporte para aninhamento de workflows – este requisito implica na necessidade de suporte a características de recursão nas interfaces do sistema de gerência.

Os demais aspectos foram descritos em requisitos opcionais ou em partes que definem questões a serem resolvidas nas propostas submetidas, como é o caso do suporte a transações através da utilização de serviços CORBA. A primeira versão da especificação foi criada a partir da proposta de um consórcio de empresas chamado jFlow [OMG97a] (com algumas alterações para incorporar definições de outra proposta [OMG97b]) .

A especificação do modelo de interfaces para um sistema de gerência de workflow definida pela OMG [OMG00a] tem como principal ponto de sua abordagem o estudo das interações dos *objetos de execução de workflows*. Estes objetos nada mais são do que as instâncias de processos ou atividades criadas pelos sistemas de gerência de workflows. Em termos gerais, as interações destes objetos de execução se dão com os dados relevantes de workflow (no momento de definir os seus contextos de execução e os resultados esperados); com os recursos utilizados pelo objeto (no momento em que faz uso de aplicações e listas de trabalho); com o seu ambiente interno (no momento de manter seu estado de execução atualizado) e com seu controlador (nos momentos em que eventos de auditoria ou de mudanças são enviados). Este estudo é base para a definição do modelo de interfaces CORBA proposto (Figura 3-4).

As interfaces mostradas na figura definem o núcleo básico de interfaces para o ambiente de execução de workflows e são descritas a seguir:

- **WfRequester –** representa uma solicitação de trabalho a ser feito pelo objeto que implementa a interface WfProcess. A interface WfRequester pode ser implementada por uma aplicação de controle ou por uma atividade que faz uso de um sub-processo aninhado. Ao longo da execução do trabalho, eventos de mudanças de estado podem ser fornecidos ao solicitador do trabalho (ex. conclusão do trabalho solicitado).

- **WfExecutionObject –** é uma interface abstrata que representa a funcionalidade básica de um objeto de execução. Esta interface é herdada pelas interfaces WfProcess e WfActivity e fornece atributos, estados e operações comuns para um objeto em execução genérico. Os principais aspectos relacionados com esta interface dizem respeito ao controle do estado de execução do objeto (que é diferente do estado do processo como um todo, que é definido pelo objeto de execução no topo do aninhamento), ao monitoramento da execução do objeto através de eventos que

representam o histórico de sua execução e às operações de definição de contexto com os dados relevantes de workflows.

- **WfProcessMgr** – Esta interface representa uma fábrica de instâncias de processos de workflow. O objeto que implementa esta interface permite o acesso a meta-informações sobre o contexto de execução e para os resultados que um processo deve retornar. Além disso, para que o objeto que implementa esta interface seja capaz de criar instâncias de uma definição de processo é necessário que este acesse o mesmo repositório de definições usado pelos componentes de modelagem dos workflows ou utilize uma representação fixa do processo.

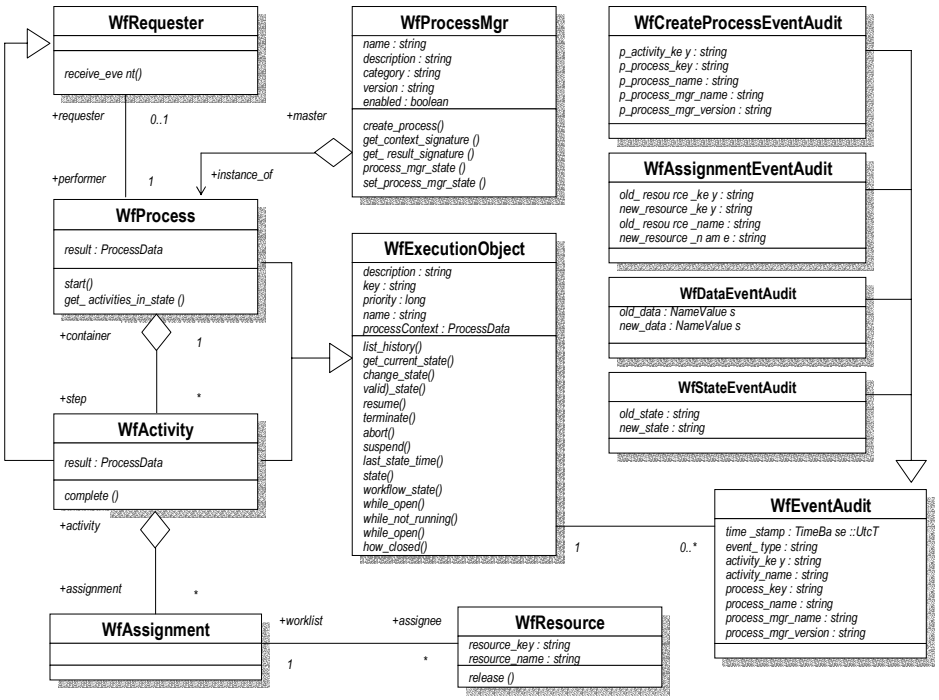

Figura 3-4 - Modelo de interfaces CORBA para Workflows [OMG00a].

- **WfProcess –** esta interface representa o responsável pelo atendimento das solicitações feitas pelos objetos que implementam a interface WfRequester. O objeto que implementa WfProcess é uma instância de processo de workflow. Esta interface especializa a interface WfExecutionObject pela adição da operação que inicia a execução do processo, da operação que obtém o resultado da execução do mesmo e relacionamentos com o solicitador do trabalho (WfRequester) e com as atividades que compõem o processo em execução (WfActivities).

- **WfActivity –** esta interface é implementada pelos objetos que implementam as atividades que compõem um processo de workflow. Ela implica em uma solicitação de trabalho no contexto do processo do qual faz parte. Assim como a interface WfProcess, WfActivity também é uma especialização da interface WfExecutionObject para incluir uma operação para sinalizar que um passo do workflow foi executado, uma operação para receber o resultado da atividade e os relacionamentos com o WfProcess que a criou e com as alocações de recursos necessários para a atividade em execução (WfAssignment).

- **WfAssignment –** esta interface representa uma alocação real de trabalho, que se dá entre um objeto que implementa uma atividade (WfActivity) e um outro que implementa, ou representa, um recurso (WfResource). Esta interface pode ser especializada por uma outra que representa um gerente de recursos. Este por sua vez interpretaria os dados de contexto de uma determinada atividade e, a partir destes, negociaria os recursos mais apropriados para serem alocados.

- **WfResource –** é uma interface que representa um recurso (abstração para pessoal ou coisa capaz de receber solicitações de alocações). É usada para implementar adaptadores para objetos que representam as pessoas, papeis e outros elementos de um modelo organizacional. Sua especificação não faz parte do escopo da especificação do sistema de gerência de workflows proposto, devendo ser considerada em outras especificações especializadas.

- **WfEventAudit –** esta interface define um conjunto de propriedades comuns para todos os eventos de workflows. As informações contidas em um evento definem a fonte do evento e uma série de dados específicos para cada evento. Na versão utilizada neste trabalho [OMG00a], quatro tipos de eventos de workflow são considerados:

    - WfCreateProcessEventAudit – notifica a criação de um processo por parte do objeto que implementa a interface WfProcessMgr;

    - WfStateEventAudit – notifica uma mudança de estado do objeto de workflow em execução por parte do objeto que implementa uma das interfaces WfActivity ou WfProcess.

    - WfDataEventAudit – notifica uma inicialização ou mudança nos dados que compõem o contexto de um objeto que implementa a interface WfExecutionObject. Este evento é gerado também quando

os resultados esperados para uma atividade (WfActivity) são inicializados ou modificados.

- WfAssignmentEventAudit – Esta interface especializa a interface abstrata de eventos para notificar mudanças relacionadas com as alocações de recursos. Por exemplo, a notificação é feita quando as alocações (WfAssignments) para uma atividade são criadas, quando o status desta alocação é modificado ou quando uma alocação já existente é re-alocada para outro recurso.

A especificação do sistema de gerência para workflows na arquitetura OMA e a definição mais precisa e formal das interfaces usadas no ambiente de execução são pontos fortes desta proposta da OMG, pois permite níveis maiores de interoperabilidade, escalabilidade e funcionalidade na medida que o *facility* (facilidade que implementa o sistema de gerência de workflows) descrito pode ser usado em conjunto com diversos serviços disponíveis da arquitetura, tais como: o serviço de nomes, de *trader*, de transações [OMG98c], etc. Além disso, uma definição mais precisa do ambiente de execução permitiu que diferentes implementações fossem mais facilmente conformes entre si e com a especificação em que se baseiam.

Seus principais pontos fracos são: a ausência de um modelo para as definições de processos que são gerenciados por um objeto que implemente a interface WfProcessMgr; a ausência de definições que mostrassem onde e como uma definição de processo (externa ou proprietária) deveria ser manipulada; a ausência de um modelo organizacional e da especificação do processo de alocação de recursos pelas atividades em execução.

Alguns destes pontos fracos foram abordados em especificações posteriores, tais como: modelo organizacional [OMG00m], alocação de recursos [OMG01t]. No entanto, o modelo para as definições de processos teve um tratamento diferenciado, na medida em que foi definido através de um perfil UML que fazia parte da especificação EDOC [OMG02b], o que já sinalizava uma tendência na OMG de passar a definir padrões de forma independente de sua plataforma base (OMA) e na direção da arquitetura MDA.

3.4 Workflows e a arquitetura MDA: A Especificação EDOC

O Perfil UML para EDOC, ou simplesmente, EDOC (*Enterprise Distributed Object Computing*) é uma especificação da OMG para a modelagem de sistemas empresariais distribuídos baseados em componentes. Embora seus fundamentos tenham surgido no final de 1998 [OMG98b], só se tornou padrão dentro da OMG, em novembro de 2001 ([OMG01a] e [OMG01b). Atualmente, a especificação EDOC [OMG02b] é um dos principais pilares para a especificação de sistemas de software

através de modelos, tal qual definida pela arquitetura MDA (*Model-driven Architecture*) da OMG [OMG01k].

A especificação descreve um framework de modelagem para sistemas de computação na Internet, integração de *web services*, sistemas baseados em mensagens, ebXML, .NET e outras tecnologias que possam dar suporte ao modelo independente de tecnologia fornecido na especificação EDOC. Este framework tem as seguintes características:

- Modelagem independente de plataforma e baseada em colaborações recursivas com diferentes níveis de granularidade e diferentes graus de acoplamento;
- Conceitos de modelagem para descrever claramente os processos de negócios e regras associadas que os sistemas aceitam, a estrutura da aplicação e o uso da infra-estrutura de serviço e organização do sistema em componentes configuráveis;
- Uma abordagem arquitetural que permite a integração de modelos de processos e modelos de informações;
- Uma abordagem de desenvolvimento que permite o acompanhamento bidirecional entre o desenvolvimento (especificação, implementação e operação) de sistemas computacionais empresariais e as funções de negócios para as quais eles são projetados e
- Suporte para a evolução e especificação de colaborações entre sistemas.

O *framework* é basicamente uma extensão de alguns elementos do meta-modelo UML da versão 1.4 [OMG01f] e é fornecido através da definição de um conjunto de elementos dos perfis UML e uma estrutura para a aplicação dos elementos dos perfís na especificação de sistemas EDOC em conformidade com MDA. Entre os elementos dos perfis, temos: um perfil independente de tecnologia, chamado de arquitetura de colaboração – ECA (*Enterprise Collaboration Architecture*); um perfil de padrões (*Patterns*); e um conjunto de modelos específicos para várias tecnologias.

A arquitetura de colaboração ECA, por sua vez, é composta por cinco perfis UML: a arquitetura de colaboração de componentes – CCA (*Component Collaboration Architecture*); um perfil para entidades; um perfil para eventos; um perfil para Processos de Negócios; e um perfil para relacionamentos.

- **A arquitetura de colaboração de componentes** – detalha como os conceitos UML de classes, colaborações e grafos de atividades podem ser usados para modelar, em níveis variados de granularidade, a estrutura e comportamento dos componentes que compõem um sistema;

- **O perfil UML de entidades** – descreve um conjunto de extensões UML que podem ser usadas para modelar objetos de entidades que são

representações de conceitos no domínio do problema da aplicação e os define como elementos que podem fazer parte de composições de vários outros elementos;

- **O perfil UML de eventos –** descreve um conjunto de extensões UML que podem ser usadas em si mesmas ou em combinação com outros elementos EDOC, para modelar sistemas dirigidos a eventos;

- **O perfil UML de processos de negócios –** especializa o CCA e descreve um conjunto de extensões UML que podem ser usadas em si mesmas ou em combinação com outros elementos EDOC, para modelar o comportamento de um sistema no contexto do negócio que está sendo suportado;

- **O perfil UML de relacionamentos –** descreve as extensões ao núcleo de facilidades UML para atender as necessidades de especificação de relacionamentos em geral e na modelagem de negócios e de software em particular.

- **O perfil UML para MOF –** descreve como representar modelos MOF em UML e vice-versa. É usado pela especificação EDOC para permitir duas formas de representação para os perfis descritos e consiste de um conjunto de tabelas que relacionam os elementos das duas especificações.

Cada perfil consiste de um conjunto de extensões UML que representa conceitos necessários para modelar aspectos específicos dos sistemas EDOC. Os conceitos são descritos em termos de perfis UML. As semânticas dos perfis (com exceção do perfil de relacionamentos) são também expressas em um meta-modelo MOF que é independente do UML. Os perfis UML ECA são independentes de tecnologias e são usados juntos para definir modelos de sistemas EDOC independentes de plataformas em conformidade com a arquitetura MDA.

A Figura 3-5 mostra os relacionamentos entre o modelo de referência RM-ODP ([ITU95c] e [ITU96]) e os perfis que fazem parte da especificação EDOC. A especificação EDOC usa as perspectivas (*viewpoints*) do modelo de referência como framework conceitual para a especificação de sistemas EDOC considerando as seguintes correspondências:

- **Especificação Empresarial –** Define uma perspectiva empresarial para um sistema EDOC. Ela é formada por instâncias de artefatos definidos nos perfis: CCA, perfil de eventos, perfil de entidades, perfil de processos de negócios e/ou perfil de relacionamentos.

- **Especificação Computacional –** Define uma perspectiva computacional para um sistema EDOC. Ela é formada por instâncias de artefatos definidos nos perfis: CCA, perfil de eventos e perfil de entidades.

- **Especificação Informacional –** Define uma perspectiva informacional para um sistema EDOC. Ela é formada por instâncias de artefatos definidos nos perfis: perfil de entidades e relacionamentos.

- **Especificação de Engenharia –** Define uma perspectiva de engenharia para um sistema EDOC. Ela faz uso de modelos de abstração tecnológica como o modelo de composição de fluxo [OMG02b].

- **Especificação Tecnológica –** Define uma perspectiva tecnológica para um sistema EDOC. Ela é formada pelos mapeamentos tecnológicos que podem ser aplicados a um modelo independente de plataforma (PIM) para torná-lo um modelo específico de plataforma (PSM).

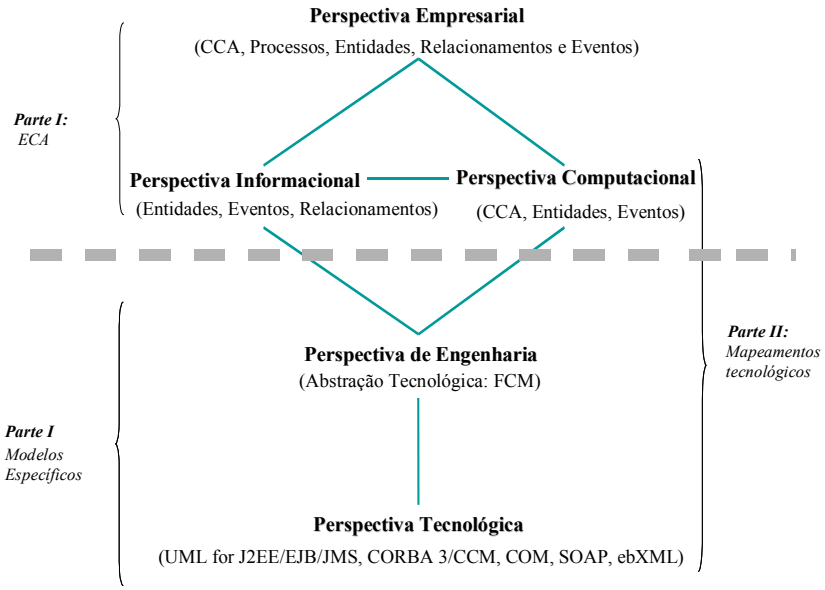

Figura 3-5 - Perspectivas RM-ODP e os perfis EDOC [OMG02b]

3.4.1. O Perfil UML para Eventos

O perfil UML para eventos descreve um conjunto de extensões UML que pode ser usado para modelar sistemas baseados em eventos. A especificação EDOC [OMG02b] define os sistemas baseados em eventos, como aqueles em que o

paradigma de computação utiliza interações que se baseiam na troca de notificações entre os componentes do sistema ao invés de considerar simplesmente instruções do que deve ser feito. O perfil UML para eventos além de poder ser usado para sistemas gerais, pode também ser usado para definir processos de negócios que fazem uso deste paradigma, considerando eventos, próprios do negócio que automatiza, para nortear os fluxos de controle e de dados destes processos.

A idéia básica por trás desta forma de lidar com os processos de negócios usando eventos é mostrada na figura a seguir (Figura 3-6):

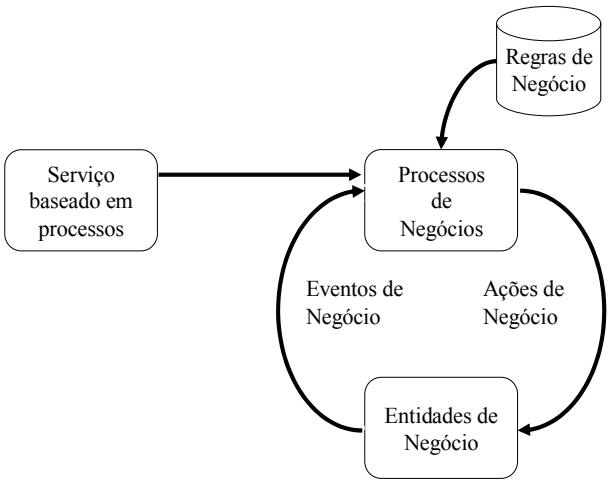

Figura 3-6 – Processos de negócios baseados em eventos [OMG02b].

A Figura 3-6 mostra os principais elementos da modelagem de processos de negócios através de eventos. Os serviços baseados em processos são aplicações que fazem uso dos processos de negócio, para usufruírem dos benefícios de coordenação de atividades oferecidos por eles. Quando o paradigma usado é baseado em eventos a ordem de execução das atividades é definida por regras de negócio que relacionam diversos tipos de eventos com as atividades definidas para o processo. As atividades de um processo permitem a utilização de entidades de negócios que representam os recursos necessários para a realização da atividade que o solicitou. É neste relacionamento que estão a maioria dos eventos usados nas regras de negócio. Por exemplo, as ações de negócios (*Business Actions*) são atividades de negócio que modificam os estados de uma ou mais entidades de negócio, enquanto os eventos de negócio (*Business Events*) representam mudanças de estados importantes ao longo da execução de uma entidade de negócio, que representa um recurso que está em uso por uma determinada atividade definida no processo.

Assim como todos os perfis EDOC, o perfil de eventos também possui um conjunto de artefatos que definem uma extensão do meta-modelo UML. O meta-

modelo de eventos é formado por dois grupos de artefatos de modelagem que representam seus dois principais aspectos: a publicação/subscrição de eventos e os eventos propriamente ditos.

Cada um dos artefatos do meta-modelo é mostrado graficamente na Figura 3-7 e tem sua própria semântica que é descrita, sucintamente, na Tabela 3-2.

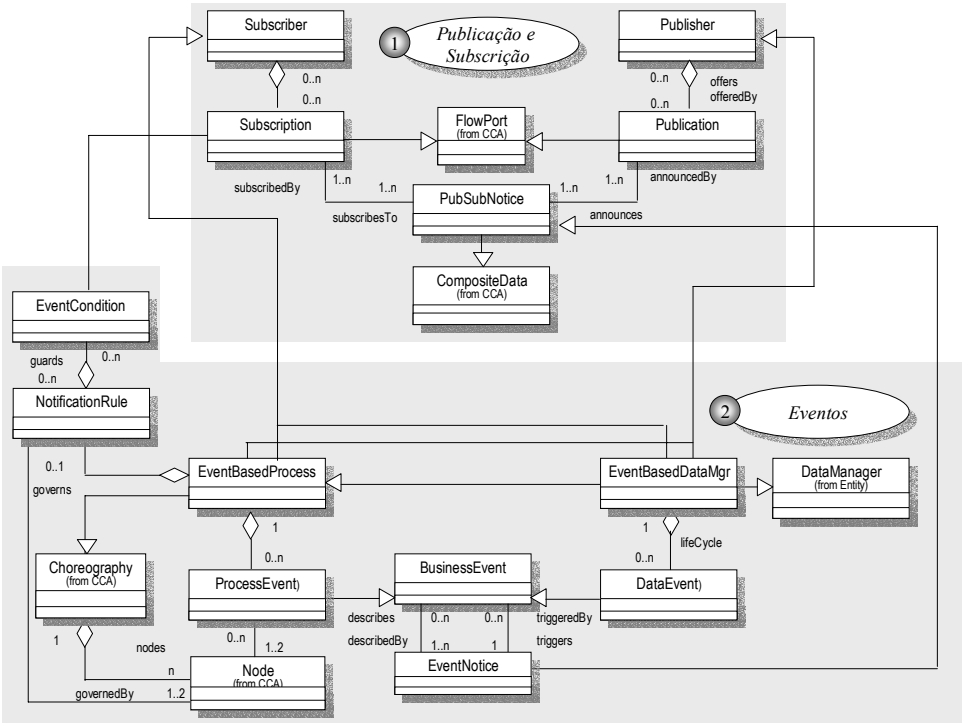

Figura 3-7 - Meta-modelo UML para eventos EDOC.

Tabela 3-1 - Principais artefatos do perfil UML para eventos

Artefato	Descrição
DataEvent	Representa um evento de negócio que reflete uma mudança em um ou mais dados gerenciados por um *EventBasedDataManager*.
EventBasedDataManager	Representa um gerente de dados, ou seja, uma entidade de negócio capaz de publicar ou se subscrever com um *PubSubNotice*.
EventBasedProcess	Representa um conjunto de atividades capazes de alterar o estado de uma entidade de negócio, gerando eventos de negócios como conseqüência.
PubSubNotice	Representa qualquer estrutura de dados que seja anunciada por uma *Publication*.
EventNotice	Representa uma *PubSubNotice* que foi disparada por um evento de negócio.

NotificationRule	Representa uma regra que governa a execução de um *EventBasedProcess* (ou uma parte desta).
ProcessEvent	Representa um evento de negócio que reflete uma mudança dentro de um processo (entradas ou saídas de *Nodes* dentro de uma coreografia[7] representada por *EventBasedProcess).*
Publication	Representa uma declaração de capacidade ou intenção de produzir um *PubSubNotice*.
Publisher	Representa um componente que forncece um *PubSubNotice*.
EventCondition	Identifica uma subscrição e especifica um subconjunto de instâncias de *PubSubNotice* que satisfazem a uma condição especificada neste artefato.
Subscriber	Representa um componente que recebe um *PubSubNotice*.
Subscription	Representa a expressão do interesse em receber um *PubSubNotice*.

Embora possa ser usado independentemente, o perfil de eventos é normalmente usado em conjunto com outros perfis para implementar formas de acoplamento mais flexíveis (*loose coupled*) entre os componentes que se relacionam.

3.4.2. O Perfil UML para Processos de Negócios

O perfil UML para processos de negócios (Perfil EDOC-BP) fornece artefatos para modelar os principais conceitos relacionados com a descrição de processos de negócios em termos de composições de atividades e de seus relacionamentos (coordenações e comunicações). Este perfil permite que dependências complexas de diversos tipos (de dados ou temporais) possam ser expressas formalmente através de recursos visuais e artefatos de modelagem associados.

O perfil permite também que os fluxos de dados e de controle sejam modelados através de conexões entre as várias portas (das atividades) e que diferentes papéis (*performer, artifact, responsibleParty*) sejam associados a estas atividades para maior flexibilidade no momento de alocar os recursos necessários para sua execução. Por exemplo, ao invés de relacionar uma determinada atividade (parte de uma definição de processo representada por uma composição de atividades - *CompoundTask*) diretamente com um recurso específico, a atividade é associada com um papel (*ProcessRole*) que é fornecido a um gerente de recursos em tempo de execução, para que este forneça o melhor recurso para a atividade

[7] *Coreografia - é um conceito abstrato, próprio do perfil CCA, que está relacionado com a ordem com que as portas são executados em um protocolo ou processo [OMG02b].*

com base nos critérios de seleção ou criação, associados com o papel (*selectionRule ou CreationRule*). A Figura 3-8 mostra o meta-modelo para processos de negócios.

O meta-modelo para processos de negócios é formado por três grupos de artefatos de modelagem que representam diferentes contextos de utilização (*UsageContexts*). São eles: a composição de processos (circunferência "1"), os papéis (circunferência "2") e as portas de entradas e saídas para atividades ou tarefas compostas (circunferência "3"). Cada um dos artefatos tem sua própria semântica que é descrita, sucintamente, na Tabela 3-2.

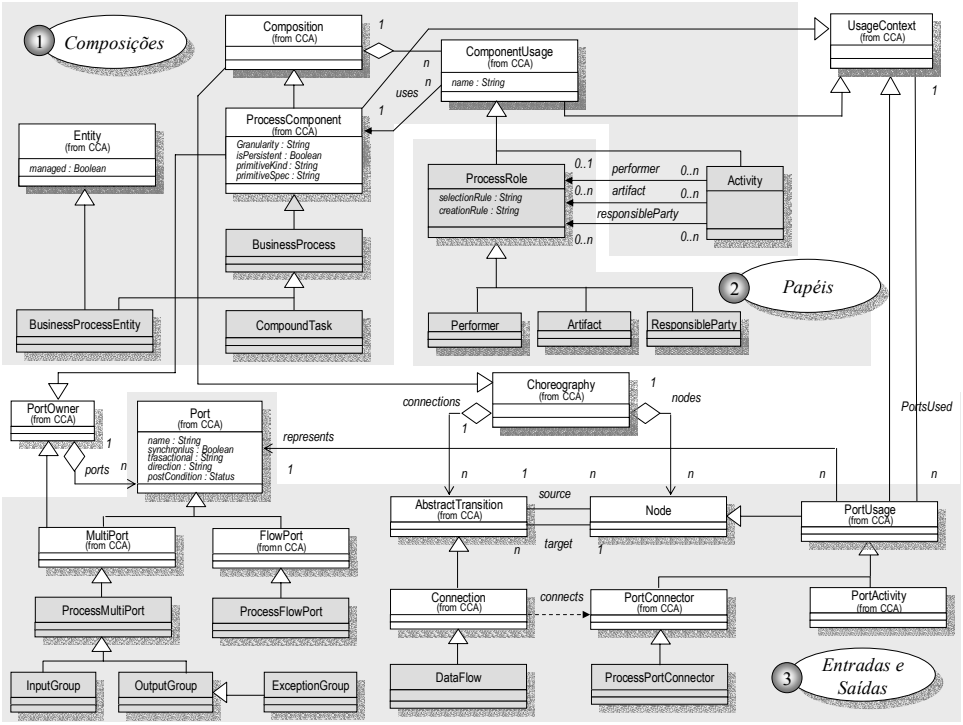

Figura 3-8 - Meta-modelo para Processos de Negócios.

Tabela 3-2 - Principais Artefatos do Perfil UML para Processos de Negócios

Artefato	Descrição
Composition	É um artefato importado do perfil EDOC-CCA que representa uma abstração usada para *ProcessComponents* e para *community processes*. Uma composição é usada para mostrar como um conjunto de components pode ser usados para definir (ou implementar) um processo.
ProcessComponent	É um artefato importado do perfil EDOC-CCA que representa uma unidade de processamento ativa. Ele pode conceber um conjunto de portas (*Ports*) para interações com outros *ProcessComponents* e pode ser configurado com a definição de propriedades que podem ser usadas em sua configuração.

ComponentUsage	É um artefato importado do perfil EDOC-CCA que representa o uso de um componente dentro de uma composição (*composition*) do tipo*ProcessComponent* ou *community process*.
BusinessProcess	É um artefato que define a visão que um *ProcessComponent* tem de uma definição de processo que coordena um conjunto de atividades relacionadas. Ele define um processo de negócio completo.
Entity	É um artefato importado do perfil EDOC-ENTITY. Ele é um objeto que representa algo no mundo real do domínio da aplicação.
BusinessProcessEntity	É um BusinessProcess que também é uma Entity com identidade. É usado para modelar processos de longa duração que podem requerer gerenciamento e ou interações durante seu ciclo de vida.
Activity	É um artefato que representa uma parte do processo de negócio usando um dos mecanismos abaixo: • A criação de uma composição de atividades aninhadas, ProcessRoles e DataFlows descritos em um CompoundTask que a atividade referencia, ou • A execução de alguma feature de um objeto bound na instância de ProcessRole referenciado pela associação performedBy.
Port	Um artefato *Port* representa uma conversação de um *ProcessComponent* ou Protocolo. Todas as interações com um *ProcessComponent* são feitas via um de seus artefatos *Ports*.
MultiPort	Este artefato combina um conjunto de *ports* com os quais são comportamentalmente relacionados. Cada port dentro de um MultiPort armazena informação enviada para um *port* até que todas as *ports* tenham recebido seus dados. Neste momento, todas as portas enviarão seus dados.
FlowPort	É um *Port* que define um fluxo de dados de entrada ou de saída com relação ao componente ou protocolo que o contém.
ProcessMultiPort	*ProcessMultiPort* representa um conjunto de *ProcessFlowPorts* relacionados que são usados para descrever as entradas e saídas de *CompoundTasks*. Eles agem como um tipo de correlacionadores para *DataFlows*.
ProcessFlowPort	Um artefato *ProcessFlowPort* representa dados usados nas entradas/saídas de uma *CompoundTask*
PortUsage	É o uso de um *port* como parte de uma coreografia.
PortConnector	Um artefato do tipo *PortConnector* representa um ponto de

	conexão para *ComponentUsages* dentro de uma *composition* e expõe as *ports* definidas dentro da *composition*. As conexões entre *PortConnectors* são feitas com Connections.
Node	Um Node é um elemento abstrato que especifica algo que pode ser origem e/ou destino de uma conexão ou transição e que faz parte de um processo coreografado.
AbstractTransition	É o fluxo de dados e/ou controle entre dois nodes.
Connection	Uma *Connection* conecta dois *PortConnectors* dentro de uma *composition*. Cada *port* pode gerar e/ou consumir eventos de mensagens.
DataFlow	Um DataFlow representa um relacionamento causal em um processo de negócio. A origem do DataFlow tem que estar pronta antes do destino.
ProcessPortConnector	Um ProcessPortConnector representa o uso de um ProcessFlowPort no contexto de uma CompoundTask.
InputGroup	Um InputGroup é uma especialização de um ProcessMultiPort. É um container para um número de ProcessFlowPorts que são entradas para uma CompoundTask.
OutputGroup	Um OutputGroup representa possíveis resultados de uma CompoundTask, fornecendo os valores dos dados associados com o resultado. Um OutputGroup modela uma coleção de valores de dados produzidos por uma CompoundTask.
ExceptionGroup	Uma ExceptionGroup representa o resultado de uma CompoundTask que falhou em sua operação normal.
ProcessRole	Este artefato é usado como uma espécie de stub para um ProcessComponent concreto que realiza uma Activity ou que é usado na realização de uma Activity. Um ProcessRole é um subtipo de ComponentUsage com alguns atributos qualificadores. Um ProcessRole pertence a uma CompoundTask e seu comportamento se torna parte do comportamento das activities com a qual está associado.
Performer	É um artefato que representa uma especialização de um tipo ProcessRole. Um Performer identifica uma Entity que pode realizar a atividade com o qual está associado.
Artifact	É um artefato que representa uma especialização de um tipo ProcessRole. Um Artifact identifica uma Entity que é necessária para uma Activity como recurso.
ResponsibleParty	É um artefato que representa uma especialização de um tipo ProcessRole. Um Artifact identifica uma Entity que é responsável por uma Activity com a qual está associado.

Um aspecto interessante a ser observado nos perfis UML que fazem parte da especificação EDOC (entre eles o perfil para processos de negócios) é a utilização de duas formas de apresentação dos meta-modelos: uma baseada em MOF e a outra baseada em UML e seus mecanismos de extensão. Por exemplo, o artefato *Port* pode ser visto através da ótica de uma arquitetura de meta-modelagem (como é o caso das ferramentas que utilizam MOF) ou da ótica de uma arquitetura de modelagem UML (como é o caso das ferramentas UML que estão preparadas para utilizar perfis). Todos os meta-modelos apresentados na especificação EDOC possuem as duas formas de apresentação. A utilização dos artefatos de extensão UML (*stereotypes, tagDefinitions, Constraints e TaggedValues*) só aparecem na segunda forma de representação. A Figura 3-9 mostra um exemplo destas duas formas de apresentar os elementos de um perfil EDOC. Na primeira (a), os mecanismos de extensão são usados para mostrar que a semântica do meta-modelo não é alterada quando da adição de artefatos de extensão; na segunda (b), os artefatos de um perfil são considerados como um pacote de meta-modelo MOF que importam definições de um pacote de meta-modelo UML(parcialmente representado em MOF) com o objetivo de estendê-lo semanticamente sem afetar o meta-modelo inicial.

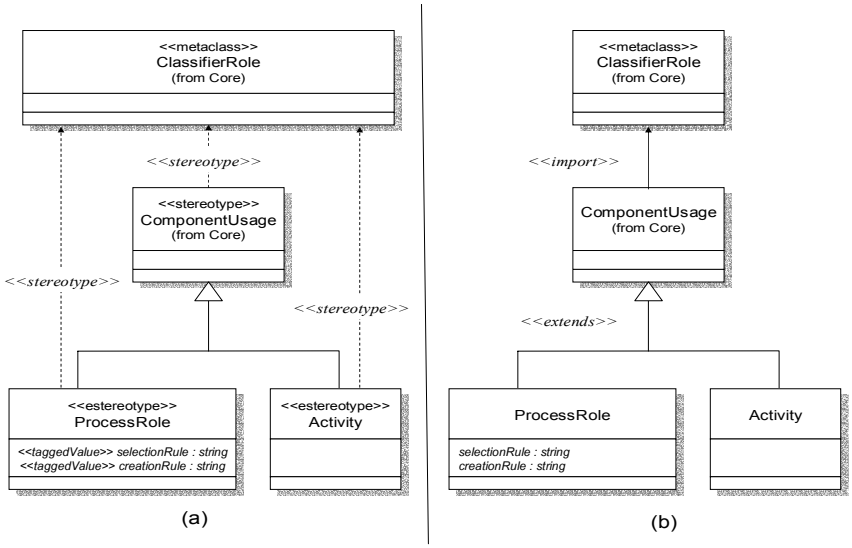

Figura 3-9 - As duas formas de apresentação de artefatos da Especificação EDOC

Outra característica interessante dos artefatos de modelagem EDOC é a disponibilização de uma notação gráfica para cada um deles. Para dar uma idéia desta funcionalidade, considere-se a utilização dos artefatos do meta-modelo do perfil de processos de negócios na representação de definições de processos genéricas. A Figura 3-10 mostra uma representação gráfica de um processo fictício (CT) com três atividades (A, B e C) e suas várias portas de entrada e saídas, bem

como seus papéis associados. Para tanto, a figura faz uso da notação associada com cada um dos artefatos que são descritos na especificação [OMG02b].

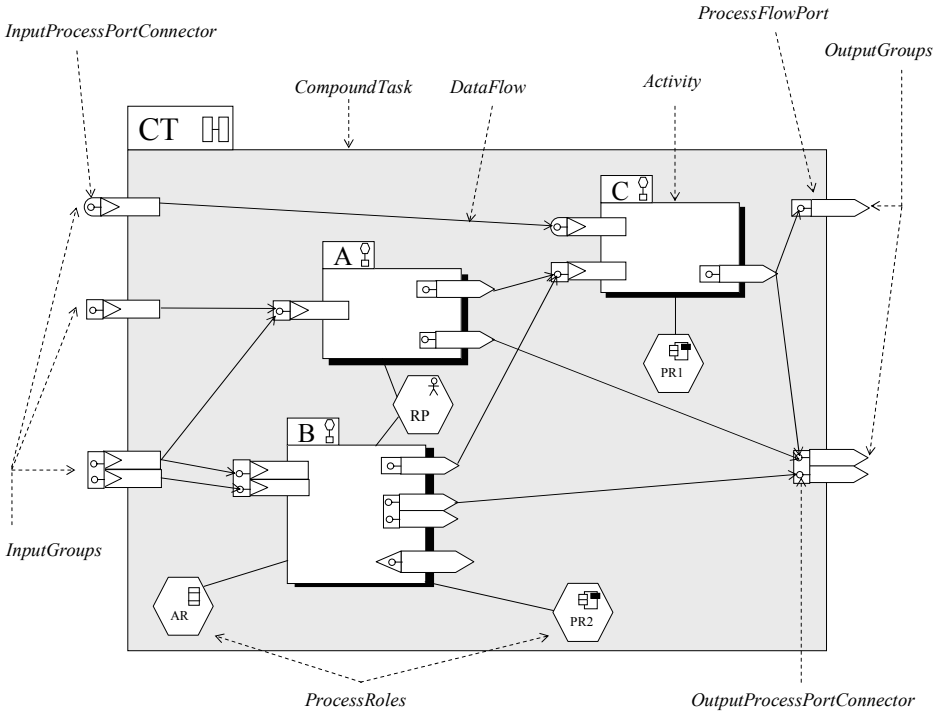

Figura 3-10 - Elementos do modelo e respectivas notações [OMG02b].

A definição de processo mostrada (*CompoundTask*) possui em sua entrada três *InputGroups* (sendo um deles do tipo *MultiPort*), que estão associados a quatro *ProcessFlowPorts* (cada um com um tipo de dado associado). As atividades internas ("A","B" e "C") também possuem portas de entrada e saída relacionadas com os *InputGroups* e *OutGroups* mostrados e os relacionamentos entre as várias portas internas e externas são representados por *DataFlows* que conectam duas portas distintas. Cada uma das atividades e a própria definição de processo, como um todo, podem ter diferentes papéis associados. Por exemplo, a atividade "C" está associada ao papel "PR 1" (*ProcessRole* "1"). A cada ProcessRole devem estar definidos componentes (ou Aplicações) que serão usados pelas atividades para realizarem seus objetivos dentro de um processo de negócio.

Apesar dos perfis EDOC permitirem uma abordagem integrada para a criação de sistemas e processos de negócios baseada nos princípios da arquitetura MDA, nenhum dos perfis considerados (*CCA, Entities, Events, Business Process*) descreve como deve ser interpretada uma definição de processo modelada segundo um CompoundTask. Isto contribui, sobremaneira, para problemas de interoperação, no momento em que diferentes sistemas de gerência de workflows (ou processos de negócios) necessitem interoperar entre si.

3.5 Resumo

Este capítulo apresentou os fundamentos da teoria relacionada com a tecnologia de workflows, seus principais padrões e a evolução da tecnologia até os dias de hoje. A primeira sub-seção apresentou a terminologia e os principais conceitos associados; a segunda sub-seção descreveu sucintamente os primeiros padrões, bem como seus pontos fortes e pontos fracos; a terceira sub-seção introduziu a especificação EDOC e o seu papel na arquitetura MDA, o perfil EDOC para processos de negócios e sua relação com a modelagem de processos de negócios. Por fim, apresentou as principais lacunas ainda existentes na modelagem de negócios MDA que motivam trabalhos adicionais, como a definição de um modelo PIM para descrever componentes de execução de workflows que sejam mais interoperáveis entre si.

Capítulo 4
Processos de Negócios na Arquitetura MDA

4.1 Introdução

Este capítulo dá seqüência à metodologia discutida no primeiro capítulo da tese, apresentando o novo contexto para os processos de negócios, seu papel na arquitetura MDA, seus cenários e principais requisitos. Nesta direção, procura resumir os fundamentos das arquiteturas baseadas em modelos e identificar os principais motivadores do novo contexto dos processos de negócios, mostrando a necessidade de um modelo independente de plataforma que possa representar ambientes de execução de processos de negócios e complementar a especificação EDOC em seu perfil para processos de negócios.

A seguinte organização foi adotada para apresentar os principais pontos do assunto tratado: a seção 4.2 apresenta uma visão geral dos principais aspectos das arquiteturas baseadas em modelos; a seção 4.3 discute o novo contexto para processo de negócios e mostra a necessidade de um modelo independente de plataformas para componentes de processos de negócios gerados segundo a metodologia MDA; a seção 4.4 analisa e compara duas diferentes abordagens para a criação de infra-estruturas que possam servir de base para a formalização de perfis UML EDOC e para os modelo PIM e PSM definidos a partir destes; a seção 4.5 comenta os principais trabalhos relacionados com a proposta desta tese; e por fim, a seção 4.6 faz um breve resumo do que foi exposto no capítulo e acrescentar algumas considerações finais pertinentes ao capítulo.

4.2 Arquiteturas baseadas em modelos

Como vimos no capítulo um, com o surgimento das plataformas de *middleware* em meados dos anos 80, pôde-se observar que a indústria de software introduziu, em média, uma nova plataforma de *middleware* a cada ano que passou, cada uma dessas plataformas prometendo, de certa forma, substituir as existentes ao oferecer maior simplicidade de uso ou universalidade de aplicação. Algumas delas eram voltadas para certos nichos de rede, onde poderiam alegar algum tipo

de vantagem sobre as demais soluções. No entanto, as novas plataformas nunca chegaram a prevalecer decisivamente sobre as concorrentes e sendo assim, cada nova introdução, na verdade, aumentou o número de plataformas que uma determinada organização deveria suportar.

A situação se tornou mais complexa com a introdução de novos tipos de equipamentos, como os PDAs (*Portable Digital Assistant*), *hand-held*s e outros dispositivos de comunicação associados a celulares, pois estes se tornaram justificativas para a introdução de *middlewares* especiais com características próprias, via de regra, diferentes dos demais. Não há nenhum sinal de que esta corrida por novas plataformas de *middleware* esteja perto de seu fim. O que torna impraticável para a maioria das organizações padronizarem uma única plataforma de *middleware* e continuar a interoperar com todos os seus clientes e fornecedores.

Além da própria questão da grande diversidade de soluções de *middleware* está também a questão da complexidade crescente dos sistemas de software atuais. Estes normalmente são complexos, não só com relação ao tamanho e complexidade do negócio que implementam, mas também com relação à forma como combinam várias tecnologias para executarem suas funcionalidades. Devido a estas características, especificar grandes sistemas atualmente, envolve a utilização de várias linguagens diferentes, cada qual com seus modelos e métodos, que em conjunto, especificam um sistema como um todo. Neste caminho, a sobreposição de linguagens é um dos principais desafios a ser vencido, pois permite muitas vezes que um mesmo elemento de um sistema seja representado em múltiplas especificações através de diferentes linguagens, o que normalmente leva a redundâncias desnecessárias. Este fator, aliado aos problemas relacionados com a heterogeneidade de soluções em *middleware*, faz com que o entendimento dos sistemas como um todo seja uma tarefa difícil e que a duração da especificação concluída tenha vida curta, uma vez que esta, quase sempre, está atrelada a um tipo de plataforma de *middleware*.

4.2.1. A questão da complexidade dos sistemas de software

Vamos considerar primeiramente a questão da complexidade dos sistemas de software. Como cada linguagem de especificação é normalmente concebida e projetada de seu próprio ponto de vista, não há forma de entender ou fazer referências aos pontos comuns de outras linguagens que participem das especificações de grandes sistemas. Sendo assim, a falta de integração por si só, já é uma grande fonte de erros e redundâncias.

Há três conceitos básicos que podem ser usados para abordar esta questão. São os conceitos de *abstração*, *semântica* e *derivação*. *Abstração* é o processo de mostrar o que é relevante a partir de um ponto de vista específico, escondendo ou ignorando os demais detalhes relacionados. *Semântica* vem do grego, *semaino*, e corresponde a *significado*. Semântica é também uma parte de uma disciplina mais

abrangente, a Semiótica, que é o estudo dos signos[8]. Dentro da Semiótica, a semântica é a parte que lida com os signos, na condição de referentes, e com as linguagens, na condição de sistemas de referentes.

No contexto deste trabalho, o interesse maior está no que se entende por *semântica compartilhada* entre linguagens de modelagem, ou seja, linguagens de modelagem com diferentes níveis de abstração compartilhando os significados de alguns elementos. Por exemplo, a semântica do termo *interface* pode ser compartilhada entre as especificações UML, MOF e Java, mas cada uma das especificações interpreta o termo do seu próprio ponto de vista, ou seja, em UML uma interface é um artefato de modelagem, em MOF a interface é uma instância de um artefato que a descreve e em Java ela é uma instância do artefato definido em UML. O terceiro conceito, *Derivação*, é o ato de originar elementos ou conceitos a partir de outros. A idéia de *Derivação* é importante porque introduz a noção de *transformação* no contexto das linguagens que possuem semântica compartilhada, permitindo que uma especificação em uma linguagem X possa ser derivada de uma outra em uma linguagem Y. Por exemplo, um conjunto de diagramas de classes UML pode ser transformado em um conjunto de classes Java. Desta forma, poderíamos dizer que a especificação em Java seria derivada da especificação em UML.

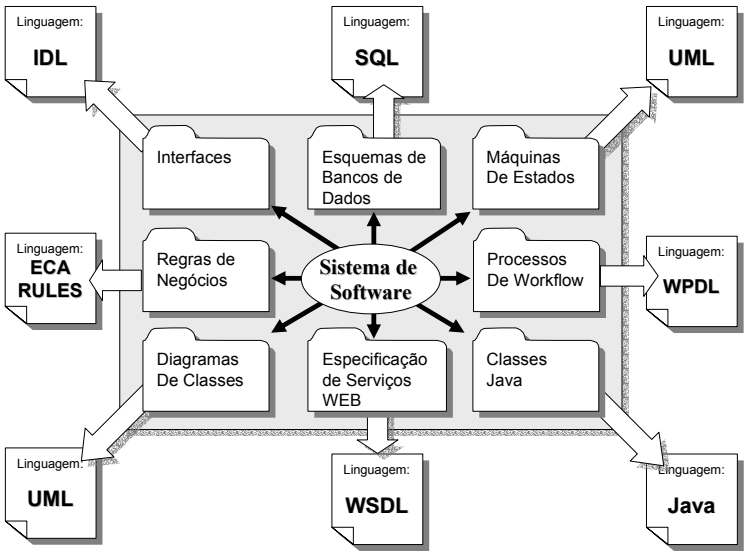

Figura 4-1 - Exemplo de sistema especificado por diferentes linguagens.

A Figura 4-1 ilustra como um conjunto de especificações em várias linguagens pode ser usado para especificar um sistema genérico. A especificação completa de um sistema de software pode dar origem, algumas vezes, ao que conhecemos por *arquitetura de software*. Uma arquitetura de software pode ser definida como uma

[8] *Signo – é uma fonte de informações que dá origem a uma interpretação de um objeto associado, ou seja, o reconhecimento da representação do mesmo [GUD96].*

estrutura (ou estruturas) que compreende(m) os componentes de software, suas propriedades externas visíveis e seus relacionamentos dentro desta mesma estrutura [KAZ99]. As arquiteturas de software baseadas em modelos possuem algumas peculiaridades que as tornam especialmente interessantes, pois ao apresentarem abordagens neutras, independentes de fabricante, fazem uso do conceito de *meta-modelos* para criar modelos que descrevem os aspectos semânticos e estruturais de um sistema de forma unificada e em um nível de abstração mais alto, ao invés de usarem uma abordagem convencional, como a mostrada na Figura 4-1. Desta forma, sempre que for necessário diminuir o nível de abstração na especificação, basta fazer uso de derivações nos elementos dos modelos para obter uma representação mais próxima das convencionais. Por exemplo, derivar um conjunto de classes C++, a partir de um diagrama de classes UML.

A Figura 4-2 mostra uma ilustração dos três conceitos (abstração, semântica compartilhada e derivação) no contexto de engenharia de software em uma arquitetura baseada em modelos. Observe que o papel da meta-linguagem pode ser desempenhado por meta-meta-modelos como o definido pelo Modelo MOF que foi apresentado no capítulo 2. Os meta-modelos são linguagens de modelagem e portanto, possuem uma sintaxe concreta, uma sintaxe abstrata e uma definição semântica bem definida. Um exemplo de linguagem de modelagem é a linguagem UML.

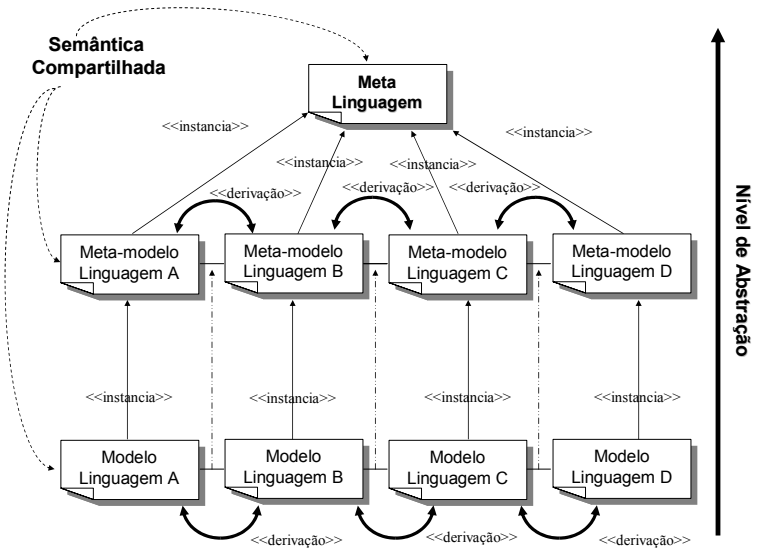

Figura 4-2 – Os conceitos de abstração, semântica e derivação de linguagens.

4.2.2. A questão da Heterogeneidade dos *Middlewares*

Outra questão interessante abordada pelas arquiteturas baseadas em modelos é a que diz respeito à forma de lidar com as mudanças que afetam a vida útil de

uma especificação, seja esta especificação abstrata ou não. Isto porque não basta apenas aumentar o nível de abstração de uma especificação de sistema, se esta não dispuser de mecanismos que permitam fácil adequação às mudanças de requisitos, aos novos *middlewares* ou às novas ferramentas de manipulação que venham a surgir ao longo do tempo. Sendo assim, é comum neste contexto a preocupação com a separação de aspectos funcionais de um sistema de seus aspectos implementacionais. A abordagem geral mais comum é a que separa os aspectos funcionais em meta-modelos ditos independentes de implementação e os aspectos implementacionais em meta-modelos que descrevem plataformas de *middleware*. A Figura 4-3 mostra um esboço deste processo que é conhecido como *O Ciclo "Y"* [BEZ01]. No Ciclo "Y", a idéia é desenvolver as etapas de análise de requisitos e projeto de um sistema através do uso de modelos independentes de plataformas (PIMs) até que um certo nível de refinamento dos mesmos seja obtido. A partir deste momento, um modelo dependente de implementação (PSM) é derivado a partir destes PIMs e do PDM[9] que descreve a plataforma de *middleware* desejada. O modelo resultante, por sua vez, pode passar por outro processo de refinamento até estar pronto para gerar código em uma linguagem específica.

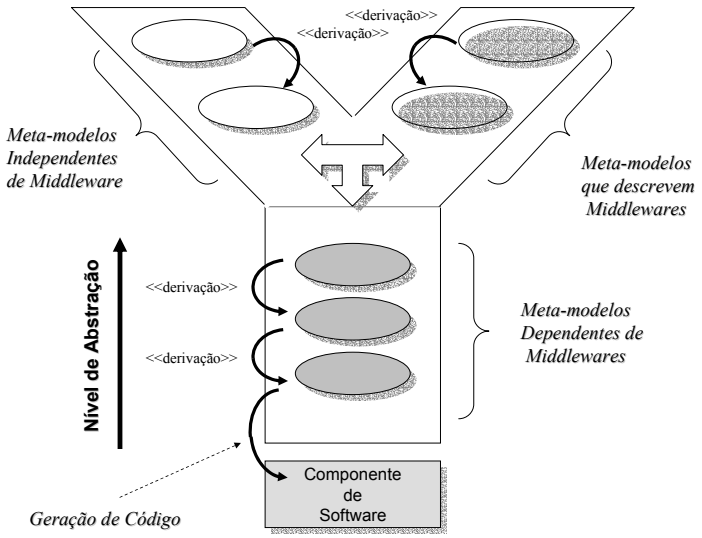

Figura 4-3 - O Ciclo "Y" [BEZ01].

É interessante observar que os conceitos de abstração, semântica compartilhada e derivação continuam presentes também neste ciclo, apesar de termos agora a derivação de um código final a partir de um modelo. Uma das propostas de maior sucesso, neste contexto, é a arquitetura MDA da OMG. Ela oferece uma interpretação do conceito de arquitetura baseada em modelos que é

[9] *PDM (Platform Definition Model) – Perfil UML que define uma plataforma de middleware específica [BEZ01].*

feita à luz das especificações de modelagem e de intercâmbio de meta-informações já existentes na OMG. Estas linguagens que formam os fundamentos da arquitetura MDA foram apresentadas no capítulo anterior (MOF, UML e XMI).

4.2.3. Principais Cenários

O ciclo "Y" da Figura 4-3 está intimamente relacionado com o ciclo de vida esperado para uma especificação baseada em modelos independentes de plataforma (PIM). Neste ciclo de vida das especificações, os três cenários apresentados no capítulo 1 são facilmente identificados como sendo cenários básicos da engenharia de software baseada em modelos.

- **Cenário 1 –** Consiste na definição de linguagens de modelagem através da meta-linguagem adotada pela arquitetura baseada em modelos;

- **Cenário 2 –** Consiste na definição de modelos que representam sistemas, através do uso das linguagens de modelagem definidas previamente no cenário 1. No caso da Figura 4-3, estes modelos seriam os modelos independentes de plataformas (PIMs) e os modelos que descrevem as plataformas (PDMs); e o

- **Cenário 3 –** Consiste na utilização de mapeamentos, refinamentos e transformações aplicados sobre os modelos desenvolvidos no cenário 2, para permitir tarefas como a geração de código final.

Um quarto cenário que pode ser considerado diz respeito a componentes gerados a partir de modelos, segundo uma metodologia MDA, e que fazem uso do ambiente de modelagem para se re-configurar dinamicamente. Este é um caso típico de gerência de meta-informações em sistemas distribuídos, que pode ser estendido aos sistemas de gerência de processos de negócios (ou workflows), quando estes representam suas definições de processos nos próprios repositórios de meta-informações que os geraram.

4.3 O Novo Contexto para Processos de Negócios

Os sistemas de gerência de processos de negócios têm sido, tradicionalmente, chamados de sistemas de gerência de *workflows* e têm dado ênfase na coordenação de atividades de pessoas (apesar de poderem também coordenar atividades automatizadas sem intervenção humana). Recentemente, o conceito de gerência de *workflows* foi estendido para incluir conceitos como: gerência de sessões na Web, integração de aplicações e colaborações empresariais. Este escopo expandido tem

sido chamado por alguns de *e-process management* [WAN00] e por outros de *gerência de processos de negócios*, expandindo o conceito anterior que se tinha para o termo [OMG02c].

O contexto das aplicações de *workflow* tradicionais estava limitado a funções de negócios específicas e a gerência de processos de negócio dentro de cada uma destas funções de negócio. Vários produtos foram desenvolvidos, implementados e evoluíram independentemente para abordar estes segmentos de automação. A aplicação destes conceitos às áreas de integração empresarial, colaborações entre negócios e gerência de sessões na Web requer que estes vários processos de negócios interoperem. A principal razão desta necessidade é que para monitorar, avaliar e alterar processos de negócios é preciso que seja possível também examinar o fluxo de trabalho através de diferentes implementações de sistemas de execução de processos de negócios, de forma que os gargalos e oportunidades de melhoria possam ser identificados. Além disso, a medida em que os participantes dos processos de negócios dependem cada vez mais de entradas de dados via web, os serviços que fazem uso de processos de negócio evoluem do modelo baseado em processamento batch para o modelo baseado em transações, em que solicitações e eventos originados em portais de informações dão origem a ações em diferentes partes dentro da empresa. Considerando-se que estas partes normalmente correspondem a diferentes funções de negócio que são executadas por sistemas de gerência de processos de negócios, isto leva a necessidade de que estes sistemas tenham conformidade com uma especificação precisa que permita sua perfeita interoperação.

4.3.1. A necessidade do PIM para Processos de Negócios

Como foi mostrado no capítulo 3, a questão da interoperabilidade entre sistemas de gerência de processos de negócios já havia sido tratada no contexto da OMG através da especificação de um sistema de gerência de workflows baseado em CORBA e nos demais serviços disponíveis na arquitetura OMA [OMG00a]. Esta especificação cumpriu bem o seu papel quando havia a orientação de direcionar as especificações da OMG apenas para o contexto da arquitetura OMA.

No entanto, com o surgimento da arquitetura MDA e com a necessidade de implementar as funções de negócios com diferentes produtos e tecnologias, a especificação OMG-WF [OMG00a] não é mais adequada e se abre espaço para a definição de um modelo independente de plataforma (PIM) para descrever o ambiente de execução destes processos de negócios.

A especificação EDOC apresentada anteriormente contribui para este contexto com a definição de um framework de modelagem para a criação de modelos PIM e PSM e entre os vários perfis propostos, inclui o perfil UML para processos de negócios (perfil EDOC-BP) que permite que artefatos UML possam ser usados para descrever a lógica de um processo de negócio de forma

independente de plataforma. Este foi um passo importante, mas não suficiente para o novo contexto dos processos de negócios ([OMG01c], [OMG01d] e [OMG01e]), pois a definição da lógica do processo não garante a interoperabilidade das implementações que dela fazem uso, pois os detalhes de como o processo deve ser acessado e executado não são abordados neste momento. Sendo assim há a necessidade também de um modelo independente de plataforma (PIM) que seja capaz de representar os aspectos internos do ambiente de execução dos sistemas de gerência para processos de negócios, de forma que as implementações de sistemas de gerência de processos de negócios (conformes com MDA) possam tratar as definições de processos da mesma maneira. Este PIM seria construído a partir do framework de modelagem EDOC para estender a especificação EDOC no que diz respeito a especificação de componentes capazes de implementar estes ambientes de execução de processos de negócios.

Em meados de 2002, a OMG oficializou a necessidade de completar e estender o perfil UML EDOC-BP com novas especificações que abordassem as questões pendentes mencionadas anteriormente ([OMG02d], [OMG02h] e [OMG02g]). A base para a nova especificação deveria ser um modelo que representasse um conjunto de interfaces de forma independente de plataforma semelhante ao que é solicitado na RFP [OMG02d] (**Figura 4-4**).

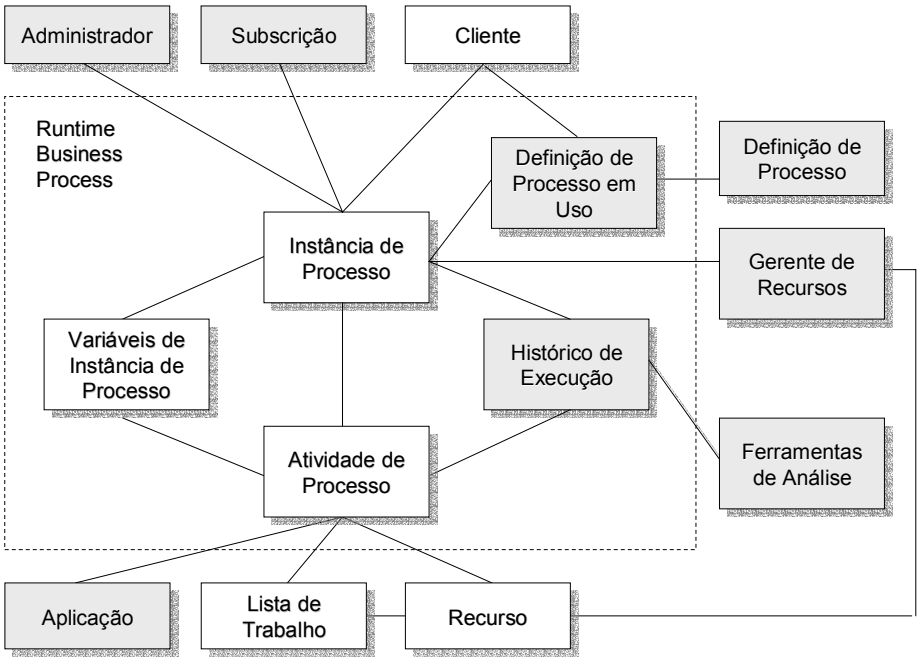

Figura 4-4 - Novas Interfaces para o ambiente de execução

O diagrama da Figura 4-4, mostra os elementos desejados para o modelo de ambiente de execução para processos de negócios. Na figura, os elementos

representados por retângulos na cor branca correspondem aos elementos que já estavam disponíveis na especificação do modelos de sistema de gerência de workflows da OMG [OMG00a], enquanto que os retângulos na cor cinza correspondem aos novos elementos desejados para o modelo independente de plataforma. Estes elementos representam as interfaces que devem ser abordadas para definir os relacionamentos com outros processos e serviços externos. Em termos gerais estas interfaces podem ser descritas da seguinte maneira:

- **Interface com Clientes** (*Requesters)* – Um cliente tem que ser capaz de solicitar a execução de um processo; fornecer as variáveis de processos iniciais que definam o contexto de execução; solicitar o status de um processo e receber notificações ao final da execução do mesmo.

- **Interface de Subscrição** (*Subscriber)* – Um *subscritor* tem que ser capaz de solicitar uma notificação de eventos de processos que possam ser de seu interesse.

- **Interface com Administrador** (*Administrator)* – Um Administrador tem que ser capaz de identificar instâncias de processos ativos; obter status dos processos; alterar a distribuição de recursos e alterar a seqüência de atividades sempre que for necessário.

- **Interface com Definição de Processos** (*Process Definition)* – Esta interface permite que uma definição de processo seja importada de um repositório para um sistema de gerência de processo de negócios.

- **Interface para Definição de Processo em Uso** – Esta interface permite que uma definição de processo seja manipulada durante a execução de uma instância de processo de negócio para facilitar a implementação de migrações dinâmicas. Uma migração de instância de processo é um conjunto de operações que agem sobre uma instância de processo para adaptá-la a uma nova versão do processo de negócio que lhe é correspondente, sem que seja necessária a reinicialização da instância em questão.

- **Interface para Instância de Processo** – Esta interface permite que clientes administradores acompanhem a execução de uma instância de processo, monitorando cada um dos passos que compõem uma instância de processo de negócio em execução.

- **Interface para Instância de Atividade** – Esta interface permite que clientes administradores acompanhem a execução de uma instância de

atividade, monitorando cada um dos passos que compõem a evolução desta atividade (alocação de recursos, execução e conclusão).

- **Interface para Variáveis de Instâncias** – Esta interface permite que clientes administradores possam alterar variáveis internas de uma instância de processo ou instância de atividade para fazer intervenções que possam ser necessárias para tratar exceções que tenham surgindo durante a execução da instância em questão. Por exemplo, alterar o estado da instância para que esta saia de um estado de suspensão e retorne a executar.

- **Interface para Histórico de execução** – Esta interface permite que todos os eventos gerados ao longo da execução de uma instância sejam acessados por administradores e/ou ferramentas de análise de processos.

- **Interface de Análise de Processos** (*Analysis Tools)* – Dados associados com o histórico de execução de um processo têm que ser disponibilizados para análises de performance e outros tipos de análises para permitir uma melhoria no processo ao longo do tempo.

- **Interface de Recursos** (*Resources)* – Uma vez atribuído um recurso, um processo pode interagir com o mesmo através desta interface. A liberação do recurso após sua utilização seria feita também através de um dos métodos da interface.

- **Interface para Lista de Itens de Trabalho** (*Worklist)* – Atribuições de recursos (particularmente, pessoas) devem ser gerenciadas através de listas de itens de trabalho (*worklists*), de forma que, todas as atribuições de um recurso possam ser observadas e gerenciadas. Instâncias de processos têm que atualizar as *worklists*.

- **Interface com Aplicações** (*Application)* – Processos podem executar aplicações, fornecendo parâmetros de execução e obtendo resultados como uma base para a execução de outros processos que venham posteriormente.

4.3.2. Os Principais Requisitos

Na RFP [OMG02d], além da funcionalidade básica que está associada aos elementos básicos e suas interfaces, são definidos requisitos adicionais que são necessários para complementar às interfaces já existentes anteriormente na

especificação do sistema de gerência de workflows da OMG [OMG00a]. Entre os principais requisitos estão:

- Criação de um PIM expresso em UML;

- Definição de um mapeamento entre o PIM criado e as interfaces propostas na especificação do sistema de gerência de workflows da OMG [OMG00a] ;

- O PIM deve ser estruturado de forma a permitir que diferentes mapeamentos possam estar associados a diferentes graus de acoplamentos, ou seja, o modelo deve ser capaz de se transformar em um componente que opera como um sistema de gerência fortemente acoplado ou como um componente onde há acoplamento fraco entre clientes, componentes e recursos;

- Definição de um modelo de máquina de estados para os processos do sistema modelado no PIM;

- Definição de uma interface de solicitações capaz de acessar informações sobre processos disponíveis, seus nomes, descrições e variáveis de contexto que permitam a execução e monitoração dos processos; operações para suspender ou terminar um processo ou modificar suas variáveis de processo; notificar a conclusão ou término de um processo com a apresentação das variáveis relevantes para seu resultado e o acesso aos estados de sub-processos relacionados com um determinado processo invocado;

- Definição de uma interface de acesso ao histórico de eventos de um processo e suas associações;

- Definição de uma interface para importar definições de processos a partir de repositórios, onde foram modeladas para dentro do ambiente de execução definido pelo PIM .

- Definição de um escopo consistente para transações envolvendo processos de negócios. Por exemplo, o escopo de uma transação de um processo em execução não deve entrar em conflito com o escopo de um de seus sub-processos.

- Definição de tratamento de exceções dentro do modelo do processo ou outras funções e serviços com as quais o componente modelado venha a interagir.

Além dos requisitos listados acima, uma série de outros requisitos complementares é considerada para o modelo em questão, mas como a própria RFP não os considerou obrigatórios, estes não estão sendo considerados aqui. Por exemplo: suporte para assinaturas digitais, contabilidade, segurança, etc.

4.4 Modelos Independentes de Plataformas

Do ponto de vista da especificação EDOC, criar um modelo independente de plataforma consiste em se especificar o modelo do sistema em termos das três primeiras especificações relacionadas com o modelo de referência RM-ODP apresentado no capítulo 3 (Figura 3-5), ou seja, a especificação empresarial, a especificação computacional e a especificação informacional do sistema em questão.

A especificação empresarial modela a estrutura e o comportamento no contexto da organização da qual o sistema modelado faz parte, considerando os processos de negócios relacionados, objetos empresariais, os principais papéis envolvidos e os relacionamentos entre eles.

A especificação computacional descreve, em termos funcionais, a implementação do sistema EDOC que é necessária para cumprir com o contrato externo do sistema (ou componente) modelado. Neste sentido, considera a decomposição do sistema em objetos computacionais que interagem através de suas interfaces internas e externas.

A especificação informacional define a semântica das informações utilizadas pelos componentes internos, bem como o processamento necessário para esta utilização.

Existem duas abordagens diferentes que podem ser consideradas no momento de definir um modelo PIM a partir de artefatos de um conjunto de perfis UML EDOC [OMG02b] :

- As técnicas que se baseiam na utilização de modelagem UML através de ferramentas que implementam o meta-modelo UML sem considerar o uso de meta-linguagens para descrevê-lo e

- As técnicas que se baseiam na utilização de meta-modelagem MOF para representar um subconjunto do meta-modelo UML que possa ser estendido por outros meta-modelos representados pelos perfis UML, segundo o mapeamento definido pelo perfil UML para MOF proposto na especificação EDOC [OMG02b].

É importante observar que a duas abordagens acima são técnicas que permitem a utilização prática do conceito de perfis em implementações concretas

baseadas em MOF e/ou UML. Cada uma delas faz uso de uma das formas de definição dos artefatos de perfis descritos na especificação EDOC.

4.4.1. Abordagem sem Meta-modelagem MOF

Esta abordagem considera uma arquitetura de modelagem como a mostrada na Figura 4-5. Na figura, não há a utilização de meta-linguagem e o meta-modelo UML é definido de forma fixa na ferramenta que implementa a arquitetura. Um perfil UML é representado por um pacote de instâncias dos artefatos UML que compõem o pacote *Foundations::ExtensionsMecanism* apresentado anteriormente no capítulo 2 (Figura 2-10).

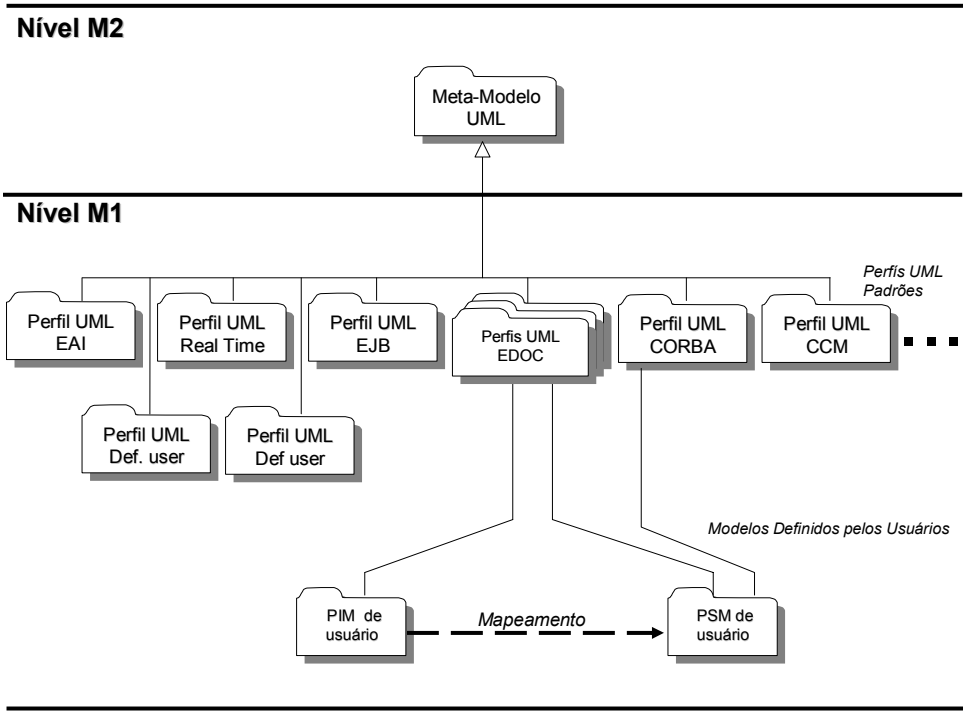

Figura 4-5 - Modelagem de perfis sem a utilização de MOF

O pacote *Foundations::ExtensionsMecanism* contém os artefatos de extensão do meta-modelo UML (*stereotypes*, *tagDefinitions*, *taggedValues* e *constraints*) que estendem o meta-modelo UML sem alterar a semântica do meta-modelo, ou seja, as extensões podem apenas criar estereótipos para as meta-classes UML já existentes sem poder adicionar novas meta-classes às já definidas inicialmente no meta-modelo. Esta forma de extensão é denominada de mecanismo de extensão *lightweight* [OMG03a] e tem como principal objetivo permitir que diversas ferramentas que implementam o meta-modelo de forma fixa, possam ser estendidas através de pacotes com estereótipos de meta-classes UML.

Observe-se que apesar dos artefatos de um perfil representarem conceitualmente elementos do nível M2, estes são representados nas ferramentas através de modelos UML que correspondem a meta-modelos virtuais. Um meta-modelo virtual é definido como um modelo formal de um pacote de extensões feitas em um meta-modelo UML através de seu mecanismo de extensão baseado em estereótipos [OMG02b].

4.4.2. Abordagem com Meta-modelagem MOF

Esta abordagem considera uma arquitetura de modelagem como a mostrada na Figura 4-6. Na figura, o meta-meta-modelo MOF é utilizado como meta-linguagem para descrever meta-modelos e um subconjunto do meta-modelo UML é definido através da meta-linguagem, para representar um meta-modelo MOF.

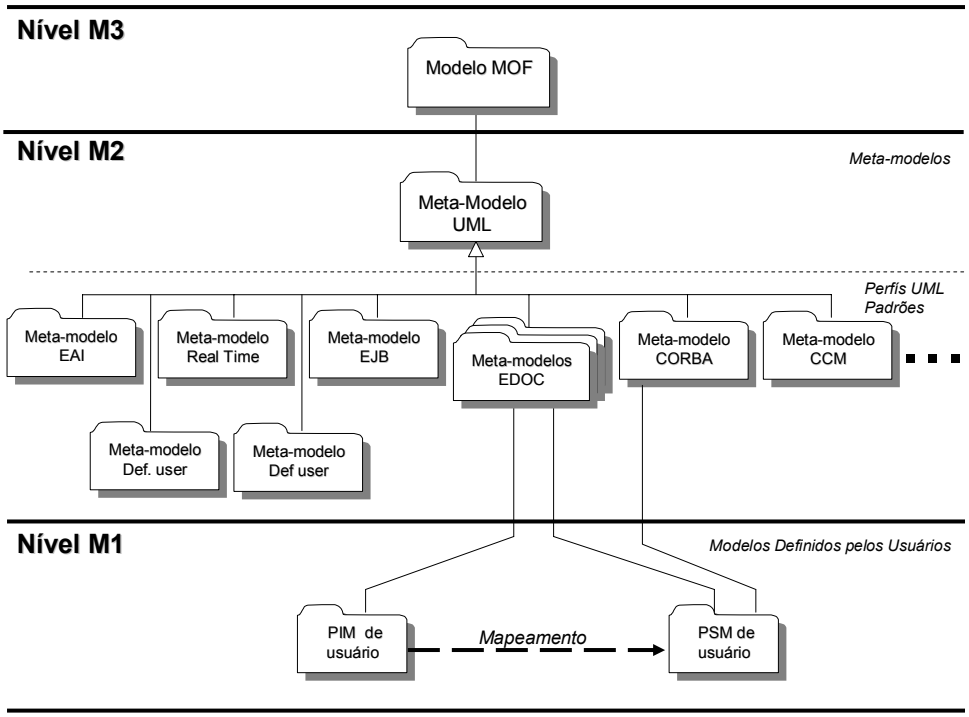

Figura 4-6 - modelagem de perfis com utilização de MOF

Um perfil UML é representado neste contexto através de meta-modelos MOF que lhe são semanticamente equivalentes. Esta equivalência é definida através de um dos perfis EDOC (o perfil UML para MOF [OMG02b]) que descreve as regras de equivalência entre um meta-modelo virtual e um meta-modelo MOF que o representa. Para manter a coerência no que diz respeito ao fato dos perfis UML estenderem a semântica das meta-classes UML, todos os meta-modelos MOF

correspondentes a um perfil UML importam o conjunto de definições disponível no pacote de meta-modelo UML que foi definido pela meta-linguagem MOF.

Esta abordagem tem a característica de não estar limitada a um único meta-modelo e de ser flexível o suficiente para implementar não só mecanismos *lightweight* descritos anteriormente, mas também os mecanismos *heavyweight* que consideram a extensão dos meta-modelos considerados através da adição de novas meta-classes. No contexto desta abordagem, os modelos PIMs e PSMs são posicionados no nível M1 e os seus relacionamentos com os perfis são relacionamentos de instanciação.

4.4.3. Análise Comparativa

Ambas as abordagens apresentadas nas subseções anteriores têm suas vantagens e desvantagens. Isto faz com que dependendo do contexto de aplicação a ser considerado, uma abordagem possa ser mais indicada que a outra. A abordagem apresentada na subseção 4.4.1 está normalmente associada às ferramentas ou aplicações baseadas em UML que surgiram antes da especificação MOF e procuraram se adaptar às mudanças na especificação UML sem se preocuparem em considerar aspectos de meta-modelagem próprios da arquitetura MOF. Estas ferramentas normalmente estão amadurecidas no que diz respeito a oferecer recursos gráficos de qualidade, próprios da notação UML. Além disso, uma vez que não há necessidade de considerar limitações relacionadas com faltas de alinhamentos entre MOF e UML, a abordagem pode fazer uso dos mecanismos de extensão para estereotipar qualquer meta-classe do meta-modelo UML e não apenas um subconjunto deste. No entanto, a natureza do mecanismo de extensão *lightweight* limita excessivamente a capacidade dos projetistas de estender o meta-modelo UML, pois não permite adição de semântica nova ao meta-modelo.

Já a abordagem apresentada na subseção 4.4.2, tem como ponto principal a flexibilidade e é teoricamente ilimitada no que diz respeito a recursos de extensão, uma vez que os meta-modelos são representados por uma meta-linguagem e não há limite para o número de meta-modelos que podem ser definidos, nem mesmo para a função que será atribuída a cada um deles. Além disso, possui um diferencial importante para os contextos em que os modelos definidos na arquitetura precisam ser utilizados pelos componentes gerados em tempo de execução, pois é comumente usada para gerenciar meta-informações a partir das interfaces de acesso MOF obtidas pelos mapeamentos tecnológicos apresentados no capítulo 2. Seus pontos fracos estão relacionados com a falta de alinhamento entres as versões 1.X das especificações UML e MOF e a relativa escassez de ferramentas gráficas amadurecidas para este tipo de arquitetura.

Como o PIM para o ambiente de execução considera a utilização de definições de processos (representadas na forma de modelos) armazenadas em repositórios de definições que implementam a arquitetura em questão, e a especificação usada

para definir os modelos relacionados com processos de negócios [OMG02b] fornece os meta-modelos MOF para todos os perfis UML apresentados na especificação EDOC, a segunda abordagem será a adotada por esta tese para definição do PIM para processos de negócios.

4.5 Trabalhos Relacionados

Esta tese se relaciona com uma série de trabalhos desenvolvidos em diferentes áreas de pesquisa, pois a definição de um modelo independente de plataforma MDA pressupõe a utilização de ambientes que permitam a utilização dos diferentes modelos PIM, PDM e PSM através do ciclo de vida discutido neste capítulo. Além disso, há vários trabalhos que procuraram definir modelos PIM para domínios de serviços diferentes do considerado aqui, cada um com sua interpretação própria e eventualmente, com seu próprio framework de modelagem. Por fim, já existem também algumas propostas de modelo PIM para a execução de processos de negócios conforme os requisitos definidos na RFP [OMG02d]. Esta seção discute os principais trabalhos destas áreas e suas relações com o trabalho desenvolvido aqui.

O primeiro deles [DST00] apresenta uma ferramenta de meta-modelagem que foi uma das primeiras provas de conceito para a especificação de um *Meta-Object Facility* nos moldes definidos pela OMG na especificação MOF [OMG00d]. Ela foi desenvolvida pelo DSTC (*Distributed System Technology Centre*) e é um sistema para criação de gerentes de repositórios de meta-informações baseados no paradigma de orientação a objetos e voltado para a arquitetura OMA. O dMOF permite que um meta-modelo seja descrito através da linguagem MODL em um arquivo texto, passe por um processo de compilação onde é representado através do modelo MOF e fique armazenado no repositório da ferramenta (usando a aplicação modl2mof). Uma série de aplicações são fornecidas para efetuar mapeamentos para o que está armazenado no dMOF. O dMOF, apesar de ser muito flexível para o ciclo de vida de um meta-modelo e seu gerente de repositório, tem algumas deficiências que limitariam em muito sua aplicabilidade no contexto de MDA, tais como: não tem integração de projeto com a especificação UML, ou seja, um modelo UML não pode ser representado diretamente em dMOF, a não ser que haja uma transformação de sua representação em UML para MODL ou XMI; não dispõe de gerência de coleções de meta-modelos, apenas das coleções relacionadas com os modelos de um dado meta-modelo; e possui mapeamento tecnológico de gerenciamento apenas para CORBA, apesar de permitir migração para XMI.

O segundo [COS01] procurou combinar técnicas de reflexão orientada a objetos com técnica de gerência de meta-informações para obter plataformas de *middleware* mas reconfiguráveis. Neste caminho, um meta-modelo foi criado para definir meta-informações próprias de um *middleware*, tais como: aspectos estruturais (componentes, interfaces, bindings, etc) e aspectos comportamentais (base de

atributos QoS, anotações QoS, etc); um ambiente reflexivo foi desenvolvido a partir de segundo ambiente reflexivo (*MMRF – Multi-model reflection framework*) [OKA95]; e um repositório das definições do MetaORB foi integrado ao seu ambiente. Este projeto foi um estudo importante dos princípios das duas técnicas de manipulação do plano de meta-informações, mas assim como o anterior ele é voltado para plataformas CORBA (com linguagem *Phyton*) e não para plataformas genéricas como se propõe a arquitetura MDA, além disso seu enfoque é na reconfiguração de *middleware* e apesar de fazer uso de modelagem e de ser até certo ponto, uma arquitetura baseada em modelos, não faz uso do framework conceitual de MDA, nem de princípios básicos como: mapeamento e transformações em modelos e meta-modelos.

O terceiro [BEL02a] utiliza o framework de modelagem EDOC para criar sistemas baseados em componentes para depois mapeá-los para componentes CCM (CORBA Component Model) através de uma linguagem MTRANS que faz uso de regras para inspecionar repositórios de definições EDOC. É um exemplo importante muito próximo do trabalho feito aqui, apesar de considerar apenas o perfil CCA e fazer uso de uma linguagem própria para implementar o mapeamento correspondente a geração de código final. Além disso, considera a geração de código para um PDM (para CCM) que ainda não está padronizado o que pode gerar incertezas a respeito da solução final proposta.

O quarto [ZIA02] e o quinto trabalho [TIL02] são semelhantes ao trabalho anterior, pois fazem uso da especificação EDOC para modelar componentes a partir do perfil CCA e depois utilizar um esquema de mapeamento proprietário para fazer a geração de código final na plataforma EJB. O que também implica em incertezas com relação a solução final proposta. Além disso, nenhum dos três trabalhos anteriores descreve a arquitetura usada para formalizar os perfis UML e definir os modelos PIM, PSM e PDM a partir destes.

O último grupo de trabalhos corresponde às primeiras propostas para a RFP para um modelo PIM para o ambiente de execução de processos de negócios [OMG02d]. A primeira proposta [OMG03b] é uma submissão preliminar parcial de um PIM para um ambiente de execução de processos de negócios geral. O único ponto abordado na proposta é uma descrição estrutural de uma arquitetura de execução baseada nos conceitos de *domínios de execução de processos de negócios* e *segmentos de processos*. Segundo a proposta, um *domínio de execução* corresponde ao conjunto de todas as instâncias de processos em conformidade com uma definição de processo, enquanto que os *segmentos de processos* correspondem ao conjunto de todas as instâncias de atividades que fazem parte deste processo. Cada um dos *segmentos de processos* de um domínio deve estar associado a um *nó* que aglomera em torno de si os vários recursos e artefatos necessários para as suas atividades. Apesar da proposta mostrar um esboço de estrutura para o ambiente de execução definido através de um diagrama UML, ela não define nenhum detalhe do comportamento necessário para a execução, nem mesmo os protocolos e interfaces envolvidos neste processo. Além disso, a proposta não faz uso de perfis UML,

nem da notação CCA necessária para definir um PIM nos moldes da especificação EDOC e da arquitetura MDA.

A outra proposta [OMG03c] toma por base o conjunto de interfaces definido no OMG-WF [OMG00a] e acrescenta cinco interfaces novas, com o objetivo de adaptar o meta-modelo de interfaces de um workflow management facility, para atuar em colaborações intermediadas por um ambiente de serviços web. Apesar de abordar a maioria dos requisitos mandatórios apontados no novo modelo de interfaces da OMG [OMG02d], a proposta falha por não definir o relacionamento do ambiente de execução com um repositório capaz de armazenar as definições para processos de negócios EDOC e na maneira de definir o PIM, pois não leva em consideração a representação dos elementos do PIM através de perfis EDOC (o que não é aceitável do ponto de vista de MDA), nem a notação gráfica necessária (notação CCA). Além disso, não mostra nenhum mapeamento do PIM proposto para uma plataforma final. O ponto forte desta proposta é a utilização de interfaces apropriadas para que o ambiente de execução possa participar de colaborações via um ambiente de serviços Web.

A impressão final após a leitura das duas propostas é a de que o conceito de PIM e as formas de descrevê-lo e usá-lo ainda não estão perfeitamente claras para a maioria dos desenvolvedores de aplicações. Além disso, nenhuma das propostas considerou um ambiente de execução de processos de negócio perfeitamente integrado com um repositório de definições de modelos MDA, no qual o próprio ambiente tenha sido gerado, o que limita consideravelmente a criação de aplicações flexíveis e interoperáveis a partir dos modelos propostos.

4.6 Resumo

Este capítulo apresentou os principais motivadores para o desenvolvimento de um modelo independente de plataforma para ambientes de processos de negócios, seus principais requisitos e as primeiras propostas apresentadas no contexto da OMG. Além disso, discutiu duas alternativas de representação para os perfis UML e seus respectivos modelos PIM e PSM, considerando suas principais vantagens e desvantagens. Por fim, justifica a escolha de uma destas alternativas para a representação e implementação que será feita no momento da validação do modelo PIM para o ambiente de execução de processos, dentro dos cenários básicos considerados na arquitetura MDA e no cenário adicional, relacionado com a gerência de meta-informações. A forma de representar o modelo PIM é importante no momento em que se define uma plataforma para formalizar as instâncias dos artefatos que representam um determinado modelo e no momento em que mapeamentos automatizados são usados para realizar transformações em modelos. Como este trabalho considera estes aspectos é importante definir previamente como os modelos PIM e PSMs devem ser representados.

Capítulo 5
Um Modelo PIM para a Execução de Processos de Negócios

5.1 Introdução

Este capítulo apresenta uma proposta de modelo independente de plataforma para representar as interfaces de um ambiente de execução de processos de negócios. Este modelo deve ser descrito a partir de perfis UML já definidos no capítulo 3 (Perfil CCA, Processos de negócios, Eventos, etc.) e deve complementar a especificação EDOC nos pontos de seu perfil para processos de negócios em que ela é omissa ou inadequada. A seguinte organização foi adotada para apresentar os principais pontos do assunto tratado. Na seção 5.2, o contexto externo para o ambiente de execução é apresentado; na seção 5.3, os principais aspectos internos do componente são modelados através dos artefatos definidos nos perfis EDOC; na seção 5.4, o modelo proposto é comparado com a funcionalidade solicitada no capítulo anterior; e por fim, na seção 5.4, um breve resumo é feito para apresentar algumas considerações finais relacionadas com o capítulo.

5.2 Perspectiva Externa

Um dos principais objetivos dos sistemas de informação de qualquer empresa informatizada e integrada é dar suporte aos seus processos de negócios. Este suporte pode ser feito de forma customizada e monolítica ou através do uso de sistemas de gerência de processos de negócios que fornecem maior flexibilidade para acomodar mudanças diversas relacionadas com estes processos.

Como foi mostrado no capítulo 3, os sistemas de gerência de processos de negócios (versão mais abrangente dos sistemas de gerência de workflows) foram criados para dar mais flexibilidade na manipulação dos processos de negócios, tanto no momento em que são definidos, quanto no momento em que são executados ou analisados. A idéia básica por trás destes sistemas é a de usar esquemas de bancos de dados para representar *definições de processos* (juntamente com outras informações relacionadas, tais como: dados, participantes, aplicações, papéis, etc); depois, criar instâncias destas definições nos bancos de dados e

repositórios associados; e por fim, permitir que os componentes em execução acessem a estes repositórios para fazerem uso das definições disponíveis e criarem instâncias de processos e atividades capazes de interagir com recursos externos para a execução do objetivo da definição de processo.

As empresas, de uma maneira geral, utilizam diferentes soluções para a modelagem e execução de seus processos de negócios. Infelizmente, nem todas são compatíveis com o mesmo padrão o que dificulta seriamente a interoperação das diferentes soluções para a realização de colaborações envolvendo processos de negócios em diferentes domínios, diferentes plataformas e diferentes sistemas de gerência. Muitos dos problemas resultantes destas dificuldades de interoperação têm sido amenizados com o aumento do nível de abstração com o qual os sistemas de software são descritos (conforme o que é proposto na arquitetura MDA).

No caso dos sistemas de processos de negócios, a especificação EDOC contribuiu sensivelmente para este contexto com a criação de artefatos para modelar definições de processos com diferentes graus de acoplamentos e por oferecer uma abordagem integrada com o projeto dos demais componentes de um sistema (capítulos 2 e 3). Infelizmente, o ambiente de execução foi esquecido na proposta da especificação. Sendo assim, apresentamos aqui uma proposta de modelo PIM para este ambiente, através dos próprios artefatos e notações usados nos perfis UML da especificação EDOC.

5.2.1. O Ambiente de Execução e seu Contexto Externo

Como discutido anteriormente, a especificação EDOC utiliza o conceito de perspectivas RM-ODP (ou *viewpoints*) para dividir o processo de especificação de um sistema em cinco etapas, que correspondem às especificações próprias de cada uma destas perspectivas (empresarial, computacional, informacional, de engenharia e tecnológica). Neste contexto, a definição de um modelo independente de plataforma (PIM) deve considerar os conceitos e notações relacionados com todas as perspectivas em que não haja nenhuma dependência associada com as plataformas finais do sistema. Conseqüentemente, a especificação de um PIM deve ser vista como uma composição das especificações que correspondem às três primeiras perspectivas do RM-ODP [OMG02b] [OMG01u]: a especificação empresarial, a especificação computacional e a especificação informacional.

No modelo PIM proposto para o ambiente de execução (ou BPRI PIM, de *Business Process Runtime Interface PIM*), a especificação empresarial modela a estrutura e comportamento do sistema BPRI no contexto da organização da qual este faz parte. Seu conceito central é o de *Comunidade*. Uma comunidade deve ser vista como uma coleção de entidades que interagem para um objetivo específico.

A Figura 5-1 mostra como a comunidade na qual o ambiente de execução de processos de negócios está inserido é representada através de um diagrama de comunidades próprio da notação CCA. Este diagrama representa uma colaboração

entre os componentes que representam os principais papéis utilizados (*roles*) no contexto em questão. Cada uma das interações entre duas ou mais portas é representada por um protocolo de comunicação que associa a cada uma destas portas, um papel de *iniciador* ou *respondedor* do protocolo em questão (o *iniciador* é uma porta de protocolo que começa um conjunto de interações e o *respondedor* é uma outra porta que responde a estas interações).

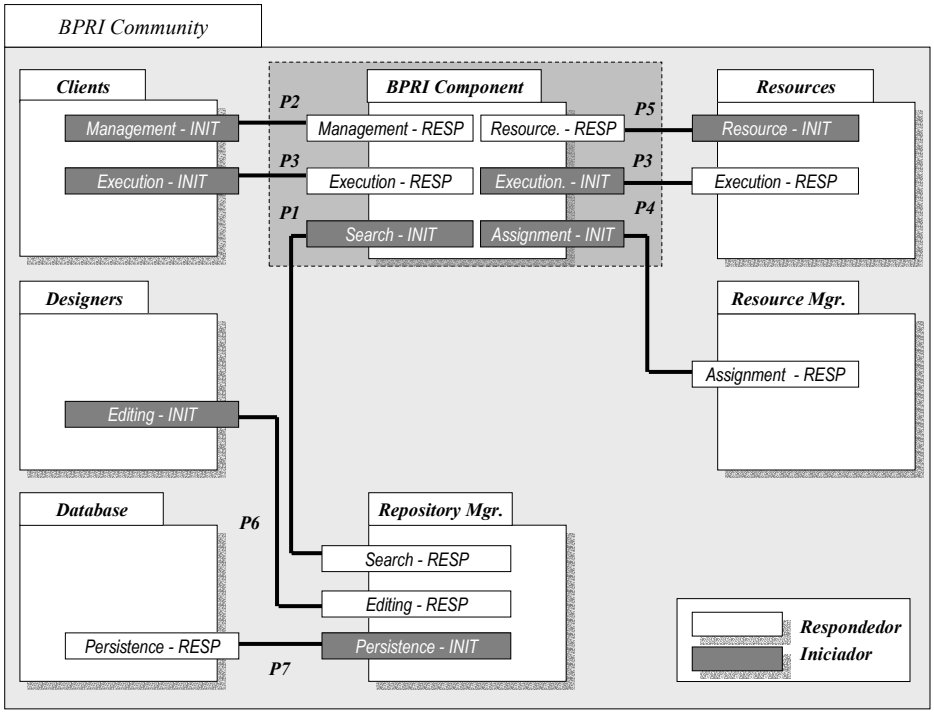

Figura 5-1 - Comunidade representando o contexto do ambiente de execução[10].

Na Figura 5-1, quatro dos componentes que reprsentam papéis importantes na comunidade mostrada estão diretamente relacionados com o componente BPRI (*BPRI Component*). São eles: um componente que representa os clientes do BPRI (*Clients*), um componente que representa o gerente de repositório que mantém representações de processos de negócios (*Repository Mgr*), um componente que representa um gerente de recursos que intermédia a seleção de recursos com base nas características fornecidas pelo BPRI (*Resource Mgr*) e um componente que representa um recurso selecionado pelo gerente de recursos (*Resources*). Estes componentes interagem com o BPRI através de cinco protocolos de comunicação representados por "Pj" (j=1 a 5) : *Search* (P1), *Management* (P2), *Execution* (P3), *Assignment* (P4) e *Resources* (P5). Os dois outros componentes mostrados não mencionados (*Designers* - Projetistas *e Database*- Banco de Dados) têm apenas relação indireta com o BPRI e implementam os protocolos de comunicação representados por "Pj" (j = 6 a 7): *Editing* (P6), *Persistence* (P2).

[10] *Os termos usados nas notações estão em inglês para facilitar a comparação com o arquivo do apêndice B.*

A comunidade pode ser descrita da seguinte maneira: o gerente de repositório (*Repository Mgr)* representa o papel de uma infra-estrutura de modelagem na qual vários sistemas são projetados, desenvolvidos e gerados segundo a metodologia definida na arquitetura MDA. As aplicações e ferramentas que fazem uso desta infra-estrutura representam o papel de Projetistas *(Designers)* de modelos e meta-modelos, que criam meta-modelos para representar os artefatos de perfis UML e fazem uso dos mesmos, para criarem modelos PIM e PSM para sistemas de software, definições de processos, regras de negócios, etc. Uma vez que um ambiente de execução de processos de negócios (representado por um BPRI PIM) for transformado em um componente executável, este poderá fazer uso do repositório da infra-estrutura para atualizar sua definição de processos sempre que esta for modificada devido a uma imposição qualquer do negócio. Lembrando que esta definição de processos representa o fluxo de controle e de dados do processo em questão, e sendo assim, é fundamental para o comportamento do componente em tempo de execução.

Para a execução de uma definição de processo, um componente BPRI deve ser acessado por clientes (*Clients)* que sejam capazes de usar seus serviços e acompanhar sua execução, à medida que os vários recursos relacionados com o mesmo (*Resources*) vão sendo usados pelo intermédio de um gerente de recursos (*Resource Mgr*). Por fim, há também o componente que faz o papel de um sistema de banco de dados que é usado pelo gerente de repositório para fazer persistência de seus modelos e meta-modelos através do *backend*[11] definido para a infra-estrutura.

Uma vez que um destes clientes tenha consultado a descrição do processo disponível e as meta-informações associadas a este (descrições dos dados de entrada e saída), poderá criar uma instância do mesmo e fornecer o contexto necessário para sua execução (dados de entrada compatíveis com as descrições disponíveis nas meta-informações).

A execução de uma instância de processo é o coração da funcionalidade de um componente BPRI, pois envolve os mecanismos necessários para interpretar uma definição de processo (fornecida a partir de um repositório) e para concretizar a coordenação do fluxo de controle e de dados embutido na definição, utilizando recursos reais associados com os *ProcessRoles* referenciados na definição de processo. Os *ProcessRoles* são elementos do perfil EDOC-BP que são usados para criar definições de processos. Eles foram apresentados inicialmente no capítulo 3.

A Figura 5-2 mostra uma outra forma de representar as interações de um componente BPRI e seu ambiente externo que foi apresentado no diagrama de comunidade. Ao invés de utilizar uma notação própria do perfil CCA, esta segunda forma de representação faz uso de diagramas de classes UML convencionais e pode ter sua semântica enriquecida com a adição de restrições descritas em OCL (*Object Constraint Language*) que representam condições

[11] *Backend – é um termo usado para referenciar serviços transparentes associados com um serviço que é utilizado diretamente por um cliente.*

invariantes, pré-condições e pós-condições e outras restrições (em linguagem natural) relacionadas com os elementos do diagrama. O diagrama da Figura 5-2 usa classes que correspondem a instâncias dos elementos do perfil EDOC CCA: *ProtocolPort*, *Protocol* e *ProcessComponent*. Cada papel da comunidade corresponde a um *ProcessComponent*, os protocolos são associados a instâncias do artefato *Protocol* e as portas que iniciam ou respondem às interações de um protocolo são representados pelas instâncias do elemento *ProtocolPort*.

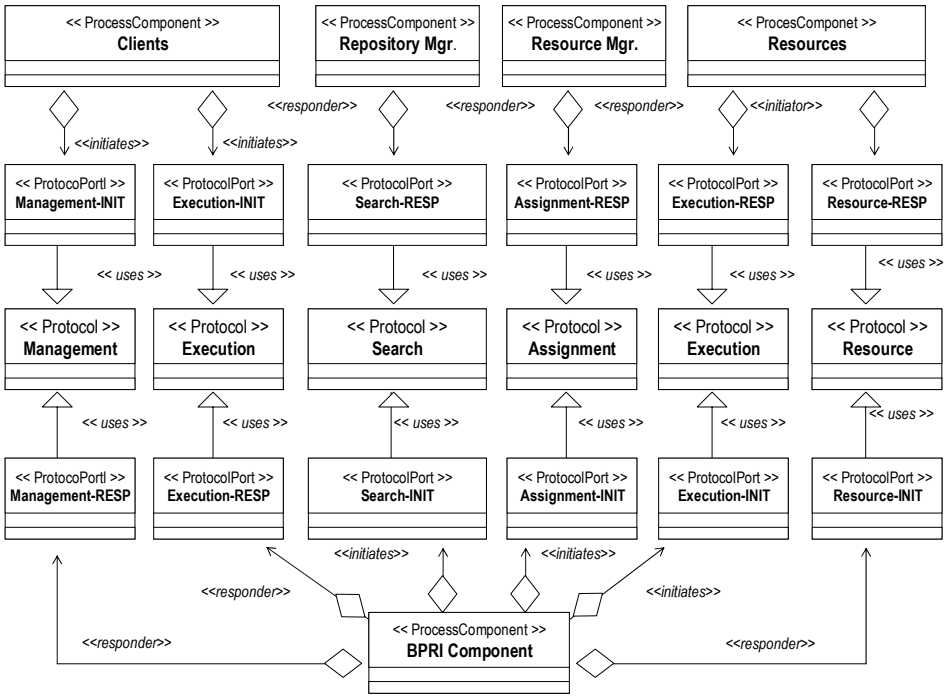

Figura 5-2 - Diagrama de classes para os protocolos externos.

As sub-seções seguintes descrevem cada um dos protocolos considerados na Figura 5-1 e na Figura 5-2.

5.2.2. Os Protocolos Externos

Um protocolo define um tipo de conversação entre dois papéis: o *iniciador* e o *respondedor*. As portas interligadas por um determinado protocolo são denominadas de *portas de protocolo,* e definem a forma de usar este protocolo, pois estão normalmente associadas também com as interfaces usadas no protocolo em questão. Várias sub-portas denominadas de *portas de fluxo* podem estar contidas em uma porta de protocolo. Estas portas de fluxo representam diferentes fluxos de dados que entram ou saem de uma porta de protocolo e são normalmente associadas a operações, métodos ou primitivas. As portas de um protocolo devem

ser sempre definidas com relação ao seu respectivo *respondedor*, pois este é o responsável pelas interfaces associadas à porta de protocolo referenciada. Cada um dos protocolos considerados nas interações externas do componente BPRI é descrito em termos de suas funcionalidades, de suas sub-portas de fluxo de dados (com as primitivas correspondentes) e da coreografia considerada para o mesmo.

5.2.2.1 Protocolo *Search (P1)*

O protocolo *Search (P1)* é um protocolo de busca que define o conjunto de regras que governa as interações entre um ambiente de execução de processos BPRI e um gerente de repositório associado a uma base de dados. Estas interações têm como principal objetivo o acesso do ambiente de execução a sua definição de processo que foi previamente armazenada no repositório. O BPRI faz uso de um conjunto de primitivas que está associado com as portas de fluxo que compõem a porta de protocolo *Search – RESP.* As primitivas são:

- **get_pd**(*DefProcessID*): Esta primitiva é iniciada por um componente que implementa um ambiente de execução BPRI e tem como objetivo acessar um modelo de definição de processo representado por *DefProcessID*. A primitiva retorna uma referência para um meta-objeto que corresponde à identificação fornecida. Este meta-objeto é uma referência para um modelo de definição de processo que foi criado pelos Projetistas para o componente que está em execução.

- **subscribe(***DefProcessID, ProcessMgrRef) :* Esta primitiva é iniciada por um componente que implementa um ambiente de execução BPRI e tem como objetivo subscrever o componente para que este receba notificações de atualização sempre que houver uma nova versão da definição de processo *DefProcessID* da qual depende sua execução. Para tanto, deve identificar também o seu ID em *ProcessMgrRef*. Não há valor retornado para esta primitiva.

- **unsubscribe(***DefProcessID, ProcMgrRef) :* Esta primitiva é iniciada por um componente que implementa um ambiente de execução BPRI e tem como objetivo cancelar uma subscrição de notificação feita anteriormente. Para tanto, deve identificar o ID da definição *DefProcessID* e o ID do componente BPRI que foi responsável pela subscrição. Não há valor retornado para esta primitiva.

- **update**() : Esta primitiva é iniciada por um repositório de definições de processos e é usada para notificar os componentes em execução de que estes devem atualizar as meta-informações anteriormente fornecidas pelo

repositório. Ela é sempre usada quando o protocolo de edição faz com que uma nova versão de um modelo de definição de processo passe a substituir uma versão anterior que está em uso por algum componente BPRI. Não há valor retornado para esta primitiva.

5.2.2.2 Protocolo *Management (P2)*

O protocolo *Management (P2)* é um protocolo de gerência que define o conjunto de regras que governam as interações entre um cliente administrador do um ambiente de execução e o componente que implementa este ambiente de execução. Seus principais objetivos são: permitir ao cliente consultar as meta-informações relacionadas com a definição de processo do ambiente; permitir a criação de uma instância de execução para o processo; permitir a definição do estado do ambiente (habilitado ou não); permitir a obtenção de informações sobre as instâncias de processos que já estão em execução.

O ambiente de execução faz uso de um conjunto de primitivas que está associado com as portas de fluxo que compõem a porta de protocolo *Management–RESP*. Como a lista de primitivas é muito grande, apenas as principais delas são consideradas nesta subseção. São elas:

- **create_process**(*ClientID*) : Esta primitiva é iniciada por um cliente do ambiente de execução para criar uma instância de processo capaz de executar a definição de processo selecionada. Ela retorna uma referência para a instância do processo que foi criada.

- **context_signature**() : Esta primitiva é iniciada por um cliente do ambiente de execução e serve para solicitar meta-informações sobre os tipos dos dados necessários para a execução de uma definição de processo. Esta primitiva retorna uma estrutura de dados denominada de *ProcessDataInfo*, ou seja, uma seqüência de pares (nome de atributo e nome do tipo) correspondendo aos nomes dos dados e seus tipos que são necessários para a execução de um determinado processo.

- **result_signature**() : Esta primitiva é iniciada por um cliente do ambiente de execução e serve para solicitar meta-informações sobre os tipos dos dados retornados após a execução de uma definição de processo. Esta primitiva retorna uma estrutura de dados denominada de *ProcessDataInfo*, ou seja, uma seqüência de pares (nome de atributo e nome do tipo) correspondendo aos nomes dos dados e tipos esperados como resultado para a execução de um determinado processo.

- **process_mgr_state**() : Esta primitiva é iniciada por um cliente do ambiente de execução e serve para verificar se o ambiente de execução está habilitado (*enabled*) ou não (*disabled*). Quando habilitado, o ambiente opera normalmente, caso contrário não responde a nenhum de seus protocolos externos.

- **receive_event(***EventID***)** : Esta primitiva é iniciada pelo ambiente de execução para confirmar a criação da instância de processo criada pela primitiva *create_process*(). *EventID* é usada para fornecer o ID de um evento correspondente à criação de um processo.

5.2.2.3 Protocolo *Execution (P3)*

O protocolo *Execution (P3)* define o conjunto de regras que governam as interações entre um cliente usuário de um processo de negócio e o ambiente utilizado por este para a execução do mesmo. Este protocolo permite também que um ambiente de execução faça uso de um outro ambiente de execução ou de um componente adaptador que represente um recurso associado a uma atividade de um processo de negócio. Sendo assim, ele é usado para definir o contexto de execução de um processo; iniciar a execução da instância criada no protocolo de *Gerência;* acompanhar a evolução da instância através de eventos que comunicam mudanças de estados, de dados, de alocações de recursos e outros eventos relevantes; e acessar componentes que representem recursos usados pelas tarefas de um processo executado localmente. O protocolo de execução (*Execution)* faz uso de um conjunto de primitivas que está associado com as portas de fluxo que compõem a porta de protocolo de *Execution–RESP*. Como a lista de primitivas é muito grande, apenas as principais delas são consideradas nesta subseção. São elas:

- **set_process_context(***ProcessData***)** : Esta primitiva é iniciada por um cliente do ambiente de execução para definir um conjunto de dados compatíveis com aqueles que definem o contexto de execução de uma instância de processo previamente criada. A estrutura de dados fornecida, *ProcessData*, é semelhante a *ProcessDataInfo*, pois também é uma seqüência de pares (nome do dado e valor do mesmo) correspondendo aos nomes dos dados e seus respectivos valores que são necessários para a execução de um determinado processo. Uma estrutura deste tipo corresponde ao que se denomina de contexto de execução para um processo.

- **start**() : Esta primitiva é iniciada por um cliente do ambiente de execução e permite que uma instância de processo previamente criada (e com contexto de execução já definido) entre em execução.

- **get_activities_in_state(***ExecutionState***)** : Esta primitiva é iniciada por um cliente do ambiente de execução para apanhar uma lista de atividades que estejam em um mesmo estado de execução. Seu único parâmetro é a indicação do estado desejado.

- **receive_event(***AuditEventRef***)** : Esta primitiva é iniciada pelo ambiente de execução para notificar a ocorrência de um evento de auditoria. Estes eventos representam mudanças de estado, de dados de contexto e de alocação de recursos utilizados pelas atividades de um processo. Seu parâmetro é uma estrutura que descreve um evento de auditoria.

- **event_type**() : Esta primitiva é iniciada pelo cliente do ambiente de execução que recebeu um evento para obter mais informações sobre o tipo do evento recebido.

- **suspend**() : Esta primitiva é iniciada pelo cliente do ambiente de execução que está acompanhando a evolução de uma instância de processo de negócio. Ela coloca a instância associada em um estado de *suspensão* no qual nenhuma atividade é executada dentro da instância de processo.

- **resume**() : Esta primitiva é iniciada pelo cliente do ambiente de execução que está acompanhando a evolução de uma instância de processo de negócio. Ela coloca uma instância que havia sido suspensa em estado de execução.

- **abort**() : Esta primitiva é iniciada pelo cliente do ambiente de execução que está acompanhando a evolução de uma instância de processo de negócio. Ela interrompe a execução da instância associada de forma brusca (não amigável). Deve ser usada quando um processo tiver que ser terminado antes de sua conclusão normal e não for possível finalizá-lo amigavelmente.

- **terminate**() : Esta primitiva é iniciada pelo cliente do ambiente de execução que está acompanhando a evolução de uma instância de processo de negócio. Ela interrompe amigavelmente a execução da instância associada com o cliente em questão.

- **subscribe_event**(*ClientID, EventID*) : Esta primitiva é iniciada por um cliente do ambiente de execução para se subscrever com relação a um tipo de evento que este deseja receber.

- **unsubscribe_event**(*ClientID, EventID*) : Esta primitiva é iniciada por um cliente do ambiente de execução para cancelar uma subscrição de notificação de evento anterior.

- **state**() : Esta primitiva é iniciada pelo cliente do ambiente de execução que está acompanhando a evolução de uma instância de processo de negócio. Ela retorna o estado de execução da instância de processo associada com o cliente que a solicitou.

- **change_state**(*ExecutionState*) : Esta primitiva é iniciada pelo cliente do ambiente de execução que está acompanhando a evolução de uma instância de processo de negócio. Ela permite que o estado de execução de uma instância de objeto em execução (Figura 5-3) seja alterado para o estado definido por *ExecutionState*. É importante notar-se, que simplesmente colocar uma instância em estado de suspensão ou conclusão forçada (amigável ou não), pode não ser suficiente para suspender ou terminar adequadamente a instância, pois alguns procedimentos podem ser necessários antes da mudança do estado propriamente dita. Por exemplo, para suspender um processo em execução é necessário que todas as suas atividades sejam também suspensas.

- **valid_states**() : Esta primitiva é iniciada pelo cliente do ambiente de execução que está acompanhando a evolução de uma instância de processo de negócio. Ela permite que o cliente possa solicitar a lista de estados válidos com relação ao estado atual em que se encontra uma determinada instância de objeto em execução.

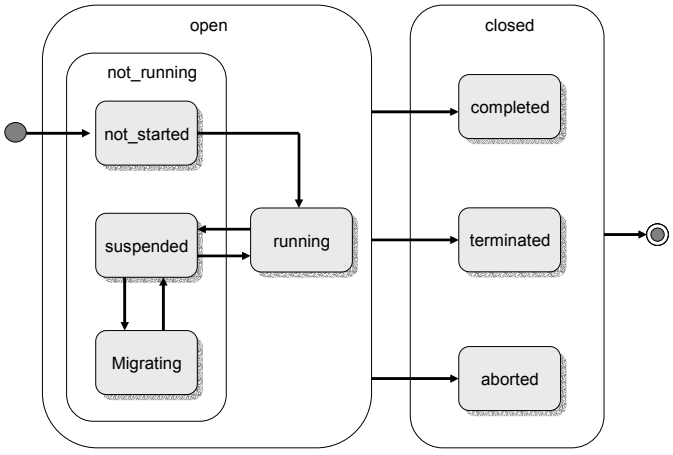

Figura 5-3 - Máquina de estados - protocolo de execução

A Figura 5-3 mostra o modelo hierárquico da máquina de estados usada para um objeto de execução do protocolo Execução. Nela estão definidos dois estados principais (*open* e *closed*) e uma série de sub-estados que fazem parte de cada um deles. O estado *open* significa que um processo está ativo e o estado *closed* significa que este foi finalizado. A tabela a seguir mostra as descrições dos sub-estados relacionados.

Tabela 5-1 - Estados de um objeto em execução

Estado	Sub-estado	
open	*not_running*	Objeto está ativo, mas não está em execução. Se o sub-estado for *not_started*, o objeto está aguardando o início da execução; se for *suspended*, o objeto estava em execução e foi paralisado temporariamente; se for *migrating*, o objeto está em processo de adaptação para uma nova definição de processo.
	running	Objeto está ativo e em execução normal.
closed	*completed*	Objeto finalizado normalmente.
	terminated	Objeto finalizado por iniciativa do usuário e antes de sua conclusão normal. Algumas das atividades relacionadas foram completadas e outras não.
	aborted	Objeto finalizado de forma brusca por iniciativa do usuário e antes de sua conclusão normal. Os estados das atividades relacionadas são indeterminados.

5.2.2.4 Protocolo *Assignment (P4)*

O protocolo *Assignment (P4)* define o conjunto de regras que governam as interações entre um ambiente de execução e um gerente de recursos externos, para a definição de reservas para os recursos que serão usados durante a execução das atividades de um processo que foi já criado e inicializado. O protocolo *Assignment* faz uso de um conjunto de primitivas que está associado com as portas de fluxo que compõem a porta de protocolo *Assignment–RESP.* Suas principais primitivas são:

- **get_assignment**(*Role*, *SelectionRule*, *CreationRule*) : Esta primitiva é iniciada por uma atividade do ambiente de execução para reservar um recurso compatível com um papel (role) associado com a atividade junto a um gerente de recursos. A primitiva em questão retorna uma estrutura (Assignment) que representa a concretização de uma alocação de recurso.

É normalmente usada após a definição de seu contexto junto ao processo que a executa.

- **get_assignment_state**() : Esta primitiva é iniciada por uma atividade do ambiente de execução para verificar o estado da alocação de um recurso. Quando o recurso é solicitado inicialmente seu estado é *Potential* (O recurso já foi localizado, mas ainda não confirmou a reserva), quando o estado é *Accepted* (reserva confirmada), a atividade do ambiente de execução responsável pela a alocação já pode utilizar o protocolo de execução para efetivar a execução do trabalho.

- **assignee**() : Esta primitiva fornece o recurso relacionado com um assignment, ou seja, retorna uma interface para o recurso previamente alocado.

5.2.2.5 Protocolo *Resource (P5)*

O protocolo *Resource (P5)* define o conjunto de regras que governam as interações entre um ambiente de execução e um conjunto de recursos que seja necessário para a execução de uma atividade. Estes recursos são localizados pelo gerente de recursos através dos dados fornecidos na primitiva get_assignment(). O principal dado usado nesse processo é o *ProcessRole* que está associado a um recurso solicitado. Este protocolo permite também o acesso a objetos adaptadores[12] que fazem com que pessoas possam responder por alguns *ProcessRoles*. O protocolo de *Recursos* faz uso de um conjunto de primitivas que está associado com as portas de fluxo que compõem a porta de protocolo de *Resource–RESP*. Suas principais primitivas são:

- **release**() : Esta primitiva é iniciada pelo ambiente de execução (uma de suas atividades) para liberar um recurso que foi previamente alocado.

- **resource_key**() : Esta primitiva retorna o id associado com um determinado recurso que foi alocada para uma tarefa.

- **resource_name**() : Esta primitiva retorna o nome associado com um determinado recurso que foi alocada para uma tarefa.

- **createRelationship**(*ActivityId*) : Esta primitiva é iniciada pelo gerente de recursos para um recurso que se qualifica para desempenhar uma função definida no papel (*ProcessRole*) fornecido pela atividade. Seu principal

[12] *Adaptadores (adapters) – São objetos de software que implementam os mecanismos necessários para que pessoas ou outros objetos possam interagir com uma atividade do ambiente de execução [OMG00a].*

objetivo é informar ao recurso que a atividade *ActivityId* vai fazer uso de seus serviços.

5.2.2.6 Coreografias

Um diagrama de classes de um protocolo mostra o que o protocolo envia e recebe sem especificar quando ocorrem estes envios e recebimentos de dados (os fluxos de dados nas duas direções). Um diagrama de atividades de um protocolo complementa o diagrama de classes fornecendo informações adicionais que mostram quando cada porta é usada com relação às demais portas do mesmo protocolo. Nas figuras a seguir (Figura 5-4 e Figura 5-5), as coreografias dos protocolos externos são mostradas através de diagramas de atividades UML.

- **Protocolo *Search*** (Figura 5-4 a) – Na coreografia deste protocolo, o iniciador (ambiente de execução) se subscreve para receber notificações de atualização relacionadas com uma definição de processo específica (*subscribe()*); logo a seguir, o iniciador acessa a versão disponível e prossegue em sua operação normal definida nos demais protocolos; Quando uma definição de processo que está em uso sofrer uma alteração que tenha surgido por uma necessidade de negócio, o gerente de repositório enviará uma notificação a todos componentes que se subscreveram para esta definição de processo; ao receber esta notificação o ambiente de execução acessa novamente o repositório para receber a versão mais nova disponível. Este processo continua até que a subscrição seja cancelada através de uma primitiva *unsubscribe()*.

- **Protocolo *Management*** (Figura 5-4 b) – Na coreografia deste protocolo, o iniciador (cliente do ambiente de execução) utiliza algumas portas para colher informações sobre o ambiente de execução e suas definições de processos. Uma vez escolhida uma definição de processo, o cliente pode solicitar a criação de uma instância deste para coordenar sua execução. Ao ser criada com sucesso, esta instância envia um evento sinalizando o sucesso da operação.

- **Protocolo *Execution*** (Figura 5-5 c) – Na coreografia deste protocolo, o iniciador (cliente do ambiente de execução) fornece um contexto de execução para a instância de processo criada anteriormente e acompanha a evolução da execução recebendo eventos de auditoria que notificam o cliente a respeito das etapas que são concluídas (mudanças de estado, de dados, etc). Ao longo deste processo o cliente pode excepcionalmente suspender a instância em execução para realizar algumas intervenções que julgue necessárias, como por exemplo, uma possível mudança de estados

visando a solução de uma situação de exceção ou algum problema com recursos que sejam necessários em atividades futuras.

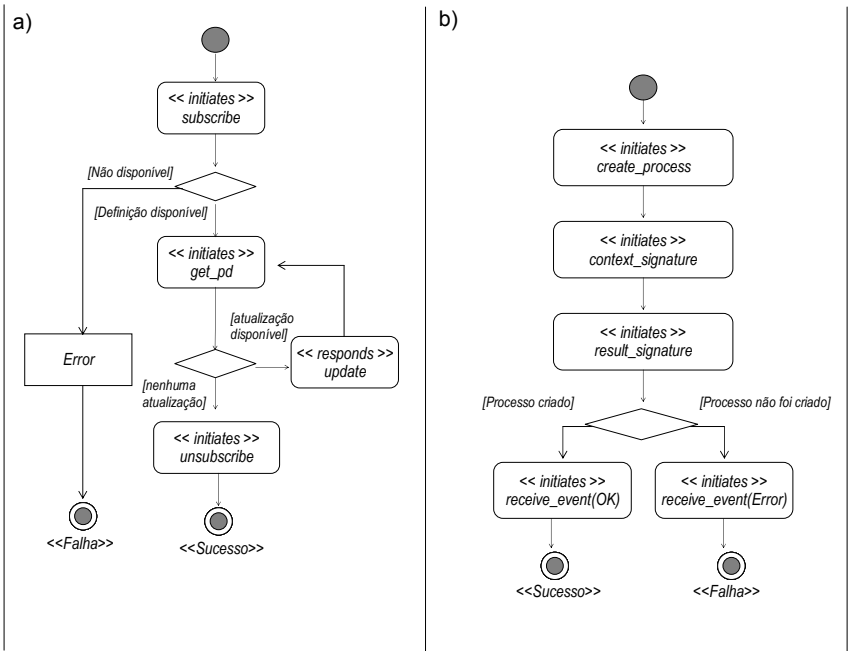

Figura 5-4 - Coreografias dos Protocolos (Parte 1)

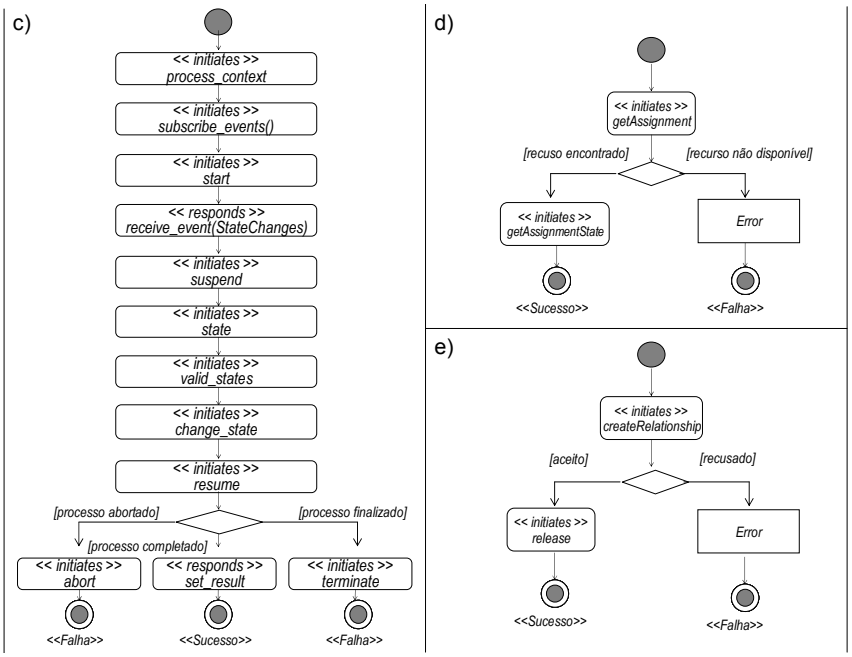

Figura 5-5 - Coreografia dos Protocolos (Parte 2)

- **Protocolo *Assignment*** (Figura 5-5 d) – Na coreografia deste protocolo, o iniciador (instância de atividade dentro de um processo em execução)

utiliza *get_assignment*() para fazer uma reserva para os recursos necessários para a atividade, caso seja aceita a reserva, a atividade poderá fazer uso dos dados retornados (identificação do recurso e estados da reserva) para nortear seu relacionamento com os recursos que fará uso.

- **Protocolo *Resource*** (Figura 5-5 e) – Na coreografia deste protocolo, o iniciador (um gerente de recursos ou uma atividade associada) aloca ou libera o recurso para que este possa ser usado pelo protocolo de execução (todo recurso deve implementar o protocolo de execução).

5.3 Perspectiva Interna

Segundo a visão apresentada na especificação EDOC [OMG02b], o próximo passo na definição de um modelo independente de plataforma é a definição dos aspectos computacionais que correspondem à especificação computacional do modelo. A especificação computacional descreve os componentes internos do sistema que está sendo modelado. Seu objetivo é complementar a descrição do tratamento que deve ser dado aos fluxos de dados correspondentes aos protocolos externos apresentados na subseção anterior. A especificação computacional de um PIM deve considerar a decomposição funcional do componente em questão (componente que implementa o ambiente de execução para definições de processos), considerando seus objetos internos e interfaces correspondentes.

A Figura 5-6 mostra um diagrama de composição para a estrutura interna do componente que implementa um ambiente de execução para processos de negócios BPRI. O diagrama de composição é próprio da notação disponível no perfil EDOC-CCA e destaca particularmente (na área pontilhada) o ambiente de execução de um processo, seus principais sub-componentes e protocolos internos. Além disso, a figura mostra como um protocolo (protocolo de execução) pode conter várias portas de fluxos ou grupos de portas de fluxos, para permitir que um mesmo relacionamento possa ser mostrado em diferentes níveis de detalhes dentro de uma composição CCA. No caso da Figura 5-6, o protocolo de execução é expandido para mostrar quatro conjuntos diferentes de portas de fluxos (*Eventos*, *Controle, Retorno* e *Monitoração*) que agrupam diferentes interações do protocolo com as entidades internas do componente.

Das primitivas listadas anteriormente para o protocolo de execução, as três primeiras primitivas (*set_context(), start() e get_activities_in_state()*) correspondem a portas de fluxos do grupo *Controle;* (*event_type()*) corresponde a uma porta de fluxo do grupo *Eventos;* (*receive_event()*) corresponde a uma porta de fluxo do grupo *Retorno*, enquanto que as demais fazem parte do grupo de *Monitoração*.

As subseções seguintes apresentam os componentes e as interfaces internas que devem ser considerados para a definição do modelo.

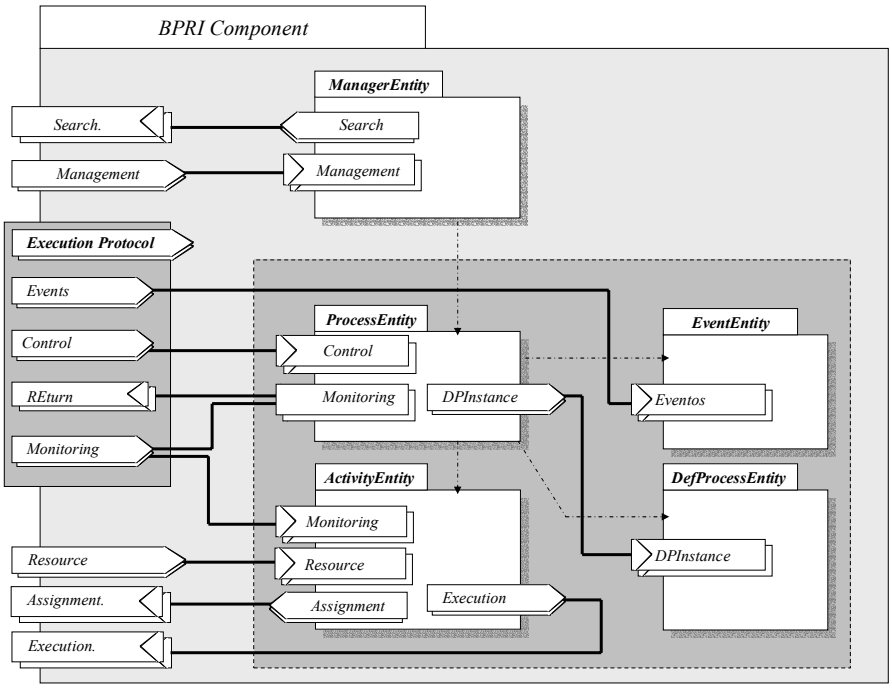

Figura 5-6 - Estrutura interna do componente (em notação CCA).

5.3.1. Componentes Internos

Os componentes internos mostrados na Figura 5-6 representam entidades que gerenciam dados relacionados com máquinas de estados usadas na operação normal do ambiente de execução para processos de negócios. Dependendo do nível de refinamento do modelo, um número maior ou menor destas entidades pode ser mostrado para representar a estrutura do ambiente de execução. Neste modelo, quatro entidades são suficientes para apresentar os principais aspectos da proposta do trabalho. São as entidades: *ManagerEntity, ProcessEntity, ActivityEntity, EventEntity e ProcessDefEntity*.

5.3.1.1 Entidade *ManagerEntity*

É a entidade responsável pela máquina de estados do ambiente como um todo. Sua principal função é interagir com um gerente de repositórios que possui as meta-informações usadas no projeto das definições de processos e seus elementos associados para poder fornecer a um cliente do ambiente de execução um panorama de todos os processos de negócios que podem ser executados pelo ambiente e os requisitos exigidos por cada um deles. Uma vez que o cliente tenha se decidido por um destes processos de negócios, a entidade *ManagerEntity* ativa uma segunda entidade denominada de *ProcessEntity* para acompanhar a execução

do processo e fornecendo a identificação do cliente que o solicitou e a definição de processo escolhida por ele.

5.3.1.2 Entidade *ProcessEntity*

É a entidade responsável pelo ambiente de execução como um todo e pela máquina de estado do processo que deve ser executado. Sua principal função é interpretar a definição de processo armazenada no repositório, para extrair desta as informações necessárias para definir os contextos das atividades, o fluxo de controle e de dados que deverá ser considerado na execução do processo em questão. Considerando-se as várias formas de se descrever uma definição de processo, a entidade *ProcessEntity* faz uso de uma entidade interna (*ProcessDefEntity)* que realiza as adaptações necessárias para que o mesmo ambiente possa funcionar com diferentes tipos de definições de processos. A entidade *ProcessEntity* só inicia sua operação de execução, quando recebe um evento *start*() que indica o momento do início aguarda do cliente que a solicitou (o contexto de execução deve ser definido antes disso). Uma vez iniciada uma execução, a entidade *ProcessEntity* ativa várias instâncias de uma terceira entidade (*ActivityEntity*) que vai acompanhar a execução dos vários passos que compõem o processo.

5.3.1.3 Entidade *ProcessDefEntity*

É a entidade responsável pela manipulação de uma definição de processo. Seu principal objetivo é identificar o tipo de definição usada no repositório de definições e extrair as informações necessárias para que um processo possa gerenciar seu fluxo de controle, seu fluxo de dados, regras de negócio, subscrições e outras informações necessárias para as atividades que são coordenadas por um processo. Como são várias as formas de se definir um processo (mesmo considerando o contexto da especificação EDOC), esta entidade dá maior flexibilidade para que um mesmo modelo de ambiente possa ser usado com diferentes tipos de definições de processos. Por exemplo, definições de processos baseadas em eventos (perfil UML de eventos) ou definições baseadas em conexões fixas definidas no perfil UML para processos de negócios.

5.3.1.4 Entidade *ActivityEntity*

É a entidade responsável pelo acompanhamento de cada um dos passos necessários para executar a definição de processo como um todo. Corresponde à funcionalidade esperada para uma atividade. Sua principal função é utilizar as

informações de seu contexto de execução para localizar os recursos necessários para a execução da atividade (através do gerente de recursos) e utilizá-los através de suas interfaces de execução para que estes retornem os resultados de seus processamentos para compor o resultado final de uma atividade. O contexto de execução de uma atividade é definido pela entidade do tipo *ProcessEntity*.

5.3.1.5 Entidade *EventEntity*

É a entidade responsável pelo fornecimento de informações relacionadas com os diversos tipos de eventos do ambiente de execução. Estes eventos indicam mudanças de estado nos objetos de execução como *ProcessEntity* ou *ActivityEntity*; mudanças no estado de alocação de um recurso; e criação de instâncias de processos. O conjunto dos vários eventos relacionados com a execução de uma instância de processo é denominado de histórico do processo.

5.3.2. Interfaces

A composição mostrada na Figura 5-6 revela como o ambiente interno do componente BPRI está estruturado para cumprir o contrato externo definido pelos protocolos discutidos anteriormente. Além da definição das entidades relacionadas com as principais estruturas de dados usadas (máquinas de estados e controles de instâncias de processos e atividades), a especificação computacional, (que faz parte da definição do modelo PIM) exige também que se definam as interfaces implementadas pelas entidades internas e as eventuais coreografias relacionadas com os protocolos internos.

Na Figura 5-6, com exceção do relacionamento entre *ProcessEntity* e *ProcessDefEntity,* os relacionamentos internos não envolvem necessariamente novos protocolos, mas sim uma série de interações que fazem parte dos protocolos já definidos anteriormente. Por exemplo, como *ManagerEntity* é que inicializa cada instância de *ProcessEntity*, ele controla uma coleção de referências para ProcessEntity denominada de ProcessSequence. Da mesma forma, *ProcessEntity* é quem inicializa cada atividade de *ActivityEntity*, ele controla uma coleção de referências para *ActivityEntity* denominada de *ActivitySequence*. *ProcessEntity* também inicializa os eventos relacionados com a evolução da execução de um processo. Portanto ele também controla uma coleção de referências para *EventEntity* denominada de *EventSequence*. Continuando um pouco mais além, *ActivityEntity* também controla duas coleções: uma associada com seus eventos e outra relacionada com as várias alocações feitas por *ActivityEntity*. Esta última coleção é denominada de *AssignmentSequence*.

As figuras a seguir mostram os diagramas de classe que representam as interfaces envolvidas e as entidades que devem implementá-las. A Figura 5-7

mostra os diagramas de classes para as entidades *ManagerEntity* e *ProcessEntity* e a Figura 5-8 mostra os diagramas para as entidades *EventEntity*, *ProcessDefEntity* e *ActivityEntity*.

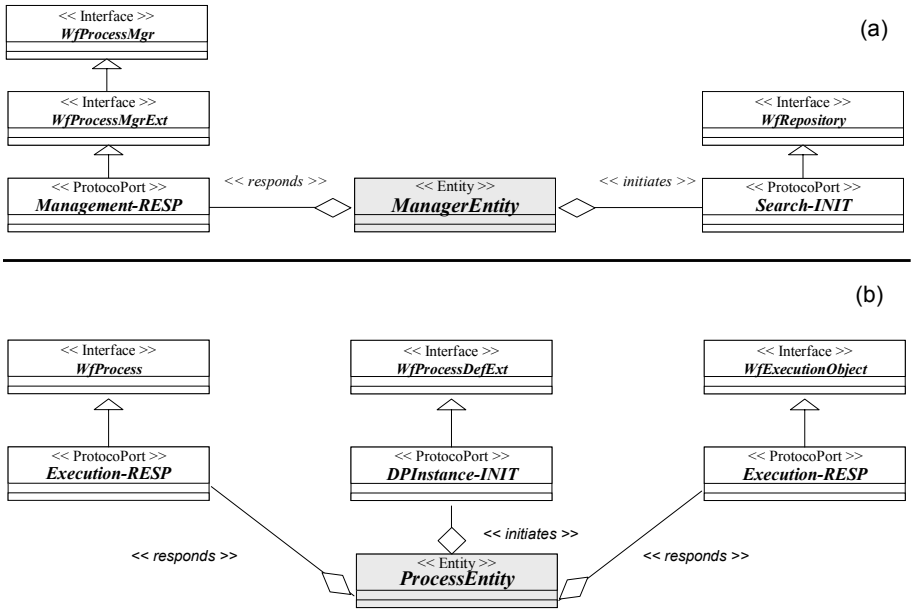

Figura 5-7 -Diagrama de classes - ***ManagerEntity*** **e** ***ProcessEntity***

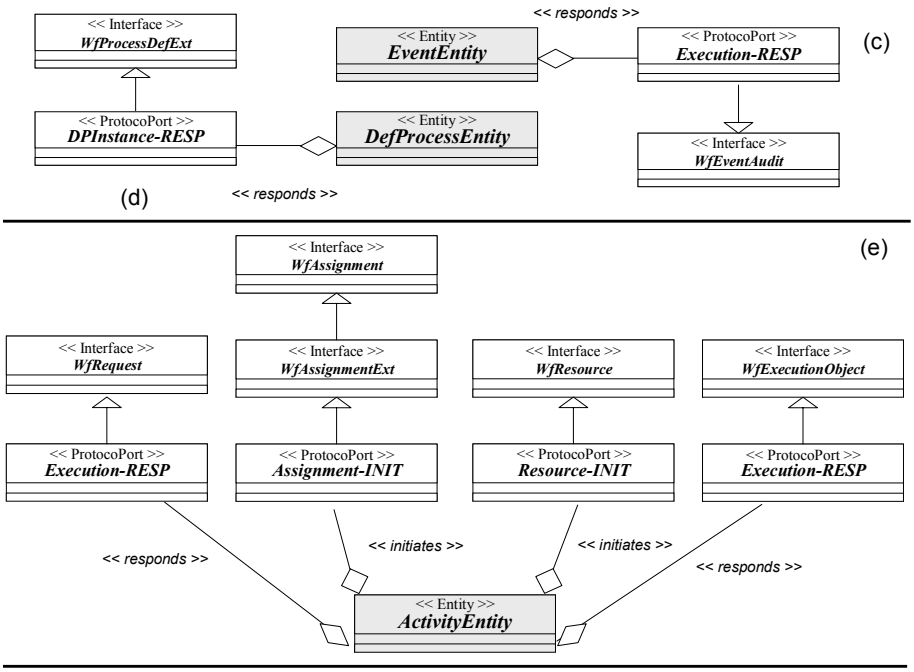

Figura 5-8 - Diagramas de classes - ***EventEntity***, ***ProcessDefEntity*** **e** ***ActivityEntity***

É importante observar-se que na especificação EDOC, as interfaces são consideradas como especializações dos artefatos que representam os protocolos e, portanto, as portas de protocolo que usam um determinado artefato do tipo *protocolo*, podem também ser associadas a artefatos que representam interfaces. Além disso, para maior facilidade de comparação, utiliza-se aqui a mesma terminologia usada para as interfaces da especificação de OMG-WF [OMG00a], pois desta forma, as interfaces que definem novas extensões são mais facilmente identificáveis. Por exemplo, a interface *WfProcessMgrExt* deve ser vista como uma interface que estende *WfProcessMgr* para agregar novos métodos representados pelas primitivas correspondente as portas de fluxo que fazem parte do protocolo de *Gerência* (P2).

5.3.2.1 As Interfaces *WfProcessMgr*, *WfProcessMgrExt* e *WfRepository*

A interface *WfProcessMgr* foi considerada na especificação OMG-WF com funcionalidade semelhante a descrita neste modelo. No entanto, nela não estava definido nenhum relacionamento com um repositório de definições de modelos. Este relacionamento é um requisito importante para o nível de integração definido nos cenários descritos no capítulo 1. O modelo proposto aqui estende a interface *WfProcessMgr* com a funcionalidade adicional necessária para que o repositório de definições possa ser usado durante a execução de um componente BPRI e para permitir que este receba notificações de atualização oriundas do repositório.

A interface *WfProcessMgrExt* especializa *WfProcessMgr* para incluir o método *update*() que é utilizado no protocolo de busca (*Search*) apresentado anteriormente e a interface *WfRepository* define a outra parte da funcionalidade do protocolo de busca, ou seja, os métodos implementados pelo gerente de repositório (**Tabela 5-2**). Seu método *get_pd*() retorna um meta-objeto MOF correspondente a uma definição de processo de negócio solicitada pelo componente BPRI; o método *subscribe*() registra um pedido de notificação de atualizações com relação ao meta-objeto utilizado; e o método *unsubscribe*() cancela um pedido de notificação já existente.

Tabela 5-2 - Interfaces WfRepository, WfProcessMgr e Extensão

WfProcessMgr *	*WfProcessMgrExt*	*WfRepository*
create_process(*requester*)	update()	get_pd()
context_signature()		subscribe()
result_signature()		unsubscribe()
process_mgr_state ()		
set_process_mgr_state()		

** - Para maior legibilidade, apenas os principais métodos de WfProcessMgr são mostrados aqui. O apêndice B mostra as interfaces completas e os métodos e atributos considerados.*

5.3.2.2 As Interfaces *WfProcess, WfProcessExt e WfProcessDefExt*

A interface *WfProcess* também foi considerada na especificação OMG-WF com uma funcionalidade semelhante à descrita neste modelo. No entanto, a necessidade de permitir que diferentes clientes (não necessariamente os que iniciam um determinado processo) possam receber eventos de um processo em execução (acoplamento fraco) levou à adição de dois métodos que especializam *WfProcess* com *WfProcessExt* e adicionam os métodos mostrados na tabela abaixo (Tabela 5-3). Além disso, o fato de um processo em execução poder receber atualizações de sua definição de processo implica na necessidade de um mecanismo de manipulação mais flexível que esteja preparado para as atualizações que surgirão ao longo do tempo. Este mecanismo é representado pela nova interface *WfProcessDefExt* (não definida em OMG-WF).

Tabela 5-3 - Interface WfProcess e Extensão

*WfProcess **	*WfProcessExt*	*WfProcessDefExt***
start()	subscribe_events()	ext_pd_type()
get_ activities_in_state()	unsubscribe_events()	extract_artifact()
result()		extract_data_flow()
set_result()		extract_control_flow()
		extract_input_data()
		extract_output_data()
		extract_compound_task()

** - Para maior legibilidade, apenas os principais métodos de WfProcess são mostrados aqui. O apêndice B mostra as interfaces completas e os métodos e atributos considerados.*

*** - A lista mostrada aqui representa os métodos implementados para o protótipo apresentado no capítulo 6, podendo ser estendida para mais artefatos de acordo com o tipo de definição de processos usada.*

Como a interface *WfProcessDefExt* não fazia parte do contrato externo de um componente BPRI, ela não foi apresentada juntamente com as demais interfaces relacionadas com os protocolos discutidos anteriormente. Sendo assim, os seus principais métodos são mostrados a seguir para dar uma idéia do mecanismo representado pela interface. Antes disso, é importante se ter em mente que a representação de uma definição de processo através de meta-objetos de um repositório MOF permite maior flexibilidade de manipulação, mas tem o custo adicional de fazer uso de estruturas muito complexas que relacionam uma grande quantidade de meta-objetos para representar cada pequeno detalhe relacionado com uma definição de processo específica. Para ter uma idéia desta complexidade, considere uma definição de processo como a mostrada na Figura 3-10 e imagine que para cada elemento presente no desenho que corresponde à notação de um

processo de negócios, deve-se criar uma ou mais instâncias dos artefatos mostrados na Figura 3-8 (eventualmente, também dos artefatos da Figura 3-7) através de meta-objetos que correspondem a estas instâncias. Por exemplo, na Figura 3-10, uma instância de artefato *CompoundTask* é usada para representar uma definição de processo através de uma composição de três atividades. Esta configuração simples implica na criação de 3 instâncias do artefato *Activity,* uma para cada atividade; 24 instâncias do artefato *ProcessPortConnector* para representar as portas da composição *CompoundTask*; 12 instâncias do artefato *Dataflow* para representar as conexões entre estas portas; 7 instâncias do artefato *ProcessFlowPort* para representar pontos de entrada ou saída de dados; 4 instâncias de *ProcessRole* para fazer referências a recursos e artefatos externos que sejam necessários para a execução das atividades; 7 instâncias de *CompositeData* para os dados que são fornecidos no contexto da composição, um para cada *ProcessFlowPort*; 3 instâncias do artefato *InputGroup* para os três grupos de *ProcessFlowPort* que definem as entradas da composição; e 2 instâncias do artefato *OutputGroup* que definem as entradas da composição. Além disso, caso haja a necessidade de considerar regras de negócios relacionadas com os eventos recebidos por processos e atividades, um outro conjunto de instâncias de artefatos deve ser considerado para cada uma das atividades existentes na composição (Figura 3-7). Como a composição é uma estrutura complexa de difícil nevegação e que varia bastante de acordo com o tipo de definição usada (com ou sem suporte a eventos e regras de negócios), a utilização de uma interface como *WfProcessDefExt*, facilita em muito a utilização dos objetos que compõem a representação baseada em grafos de meta-objetos, permitido acesso direto às informações necessárias para compor os fluxos de controle e fluxos de dados usados no componente BPRI. Os métodos considerados na interface *WfProcessDefExt* devem ser vistos como interpretadores do formato utilizado no repositório MOF (que compõe a infra-estrutura de modelagem) que deixam transparente para o implementador do componente uma série de detalhes pouco importantes. Por exemplo, o método (*ext_pd_type()*) analisa uma determinada definição de processo para ver se esta está baseada em fluxo de controle previamente determinado (perfil EDOC-BP), ou se o fluxo de controle é definido com base em eventos e regras de negócio (perfil EDOC-Eventos), ou ainda se é uma definição mista que considera artefatos dos dois perfis. O valor retornado por *ext_pd_type()* é um string que corresponde ao tipo de definição de processo considerado ("*EDOC-BP*", "*EDOC-Eventos*" ou "*Mix*").

O método *extract_artifact(<artefato>)* tem como objetivo fazer acesso direto ao meta-objeto que corresponde ao artefato desejado sem a necessidade de navegação por toda a estrutura correspondente à definição. O valor retornado é o meta-objeto que representa o artefato desejado.

O método *extract_data_flow()* tem como objetivo extrair um conjunto de meta-objetos que participa na definição do fluxo de dados do processo, ou seja, instâncias do artefato *DataFlow*. O valor retornado é uma estrutura com uma lista

de meta-objetos representando as várias instâncias do artefato *DataFlow* e seus relacionamentos.

O método *extract_control_flow()* tem como objetivo extrair um conjunto de meta-objetos que participa na definição do fluxo de controle do processo, ou seja, instâncias dos artefatos *Activities* e instâncias do artefato *ProcessPortConnector*. O valor retornado é uma estrutura com dois campos que representam duas listas de meta-objetos (uma para cada conjunto de instâncias dos artefatos).

O método *extract_input_data()* tem como objetivo extrair um conjunto de meta-objetos que compõem os dados de entrada e suas portas de fluxo correspondentes, ou seja, instâncias dos artefatos *ProcessFlowPort e* instâncias do artefato *InputGroup*. O valor retornado é uma estrutura com três campos que representam três listas de meta-objetos (uma para cada conjunto de instâncias dos artefatos).

O método *extract_output_data()* tem como objetivo extrair um conjunto de meta-objetos que compõem os dados de entrada e suas portas de fluxo correspondentes, ou seja, instâncias dos artefatos *ProcessFlowPort e* instâncias do artefato *OutputGroup*. O valor retornado é uma estrutura com três campos que representam três listas de meta-objetos (uma para cada conjunto de instâncias dos artefatos).

O método *extract_compound_task()* tem como objetivo extrair o meta-objeto que representa o topo da hierarquia de meta-objetos. Ele é importante para permitir que outros dados da definição possam ser acessados a partir do ponto de maior hierarquia na composição.

A relação de métodos disponíveis na interface *WfProcessDefExt* reflete as necessidades do primeiro protótipo de componente BPRI implementado, mas pode evoluir para incluir métodos que permitam tratamentos mais elaborados da estrutura disponível no repositório.

5.3.2.3 As Interfaces *WfActivity* e *WfActivityExt*

A interface *WfActivity* também foi considerada inicialmente na especificação OMG-WF com funcionalidade semelhante à descrita neste modelo, entretanto, a necessidade de considerar diversos tipos de acoplamentos entre atividades e recursos exigiu algumas modificações para que uma atividade fosse capaz de receber também notificações de eventos gerados pelos seus recursos alocados. Isto levou à extensão de *WfActivity* para incluir um novo método (*receive_event())* e a formar a interface *WfActivityExt* (Tabela 5-4).

Tabela 5-4 - Interface WfActivity e Extensão

WfActivity *	*WfActivityExt*
complete()	*receive_event()*
result()	

set_result()	
** - Para maior legibilidade, apenas os principais métodos de WfActivity são mostrados aqui. O apêndice B mostra as interfaces completas e os métodos e atributos considerados.*	

A questão relacionada com o tipo de acoplamento entre atividades e recursos está fortemente relacionada com a utilização ou não do paradigma de eventos na definição de um processo de negócio. Existem dois tipos de acoplamentos a se considerar: acoplamento rígido (*tight coupled*) e acoplamento fraco (*loose coupled*). No acoplamento rígido, os relacionamentos entre os vários componentes que foram mostrados no diagrama de comunidade da Figura 5-1 fazem uso de referências fixas para os participantes dos vários protocolos considerados, enquanto que no acoplamento fraco estas referências são definidas dinamicamente através de subscrições/publicações relacionadas com os eventos gerados/consumidos pelos componentes da comunidade. A especificação OMG-WF considera apenas acoplamento rígido, enquanto que a especificação EDOC considera e defende a utilização de esquemas com acoplamento fraco. A experiência prática, entretanto, revela a necessidade das duas formas de acoplamento, pois ambas têm suas vantagens e desvantagens, que podem ser maiores ou menores dependendo do contexto a ser considerado. O documento que oficializa a RFP para o PIM BPRI [OMG02d] reconhece esta necessidade e neste sentido, o modelo proposto procura se adaptar às duas formas consideradas.

5.3.2.4 As Interfaces *WfAssignment* e *WfAssignmentExt*

A interface *WfAssignment* foi mencionada inicialmente na especificação OMG-WF, mas não chegou a ser totalmente definida na ocasião. Este modelo do PIM BPRI faz uso de uma extensão de *WfAssignment* (*WfAssignment*) que especializa a interface anterior para considerar o acesso a um Gerente de recursos (*ResourceMgr*) que é responsável pelas localizações e reservas de recursos para que estes possam receber itens de trabalho de uma atividade. Itens de trabalho são dados necessários para execução de uma atividade.

Tabela 5-5 - Interface WfAssignment e Extensão

*WfAssignment **	*WfAssignmentExt*
activity()	*get_assignment()*
assignee()	*get_assignment_state()*
set_assignee()	
** - Para maior legibilidade, apenas os principais métodos de WfAssignment são mostrados aqui. O apêndice B mostra as interfaces completas e os métodos e atributos considerados.*	

A utilização de um gerente de recursos facilita em muito a implementação de relacionamentos com diversos tipos de acoplamentos, pois o gerente passa a ser o responsável pelas referências dos recursos utilizados pela a atividade, permitindo a atualização dos mesmos a qualquer momento da execução de um processo. É importante lembrar que uma definição de processo permite que as atividades façam referências apenas ao papel do componente correspondente ao recurso, ou seja, seu *ProcessRole* e não a identificadores específicos para estes, o que facilita a implementação de acoplamentos mais flexíveis.

5.3.2.5 A Interface *WfRequest*

A interface *WfRequest* não sofreu nenhuma alteração e está definida da mesma forma em que foi considerada na especificação OMG-WF.

5.3.2.6 A Interface *WfExecutionObject*

A interface *WfExecutionObject* não sofreu nenhuma alteração e está definida da mesma forma em que foi considerada na especificação OMG-WF.

5.3.2.7 A Interface *WfResource*

A interface *WfResource* não sofreu nenhuma alteração e está definida da mesma forma em que foi considerada na especificação OMG-WF.

5.3.3. Modelo de Informações

O terceiro passo necessário para criar um modelo independente de plataforma é a especificação informacional do sistema [OMG02b]. A especificação informacional define a semântica da informação e o processamento relacionado com estas informações. No caso dos sistemas baseados nos perfis EDOC, a especificação informacional faz uso dos perfis de entidades e de relacionamentos, para fazer a definição da estrutura da informação. Neste ponto os objetos de informação são modelados como entidades e relacionamentos e algumas restrições são definidas em termos de estados enumerados, propriedades de relacionamentos e algumas invariantes UML.

As Figura 5-9 e Figura 5-11 mostram os principais elementos da especificação informacional para o modelo PIM do ambiente de execução de processos de negócios proposto neste capítulo. A especificação divide o conjunto de informações em seis grupos que são representados nas figuras pelas letras de “a” a “f”. O primeiro grupo (a) representa as coleções de identificadores usados pelas

entidades que criam instâncias de outras entidades dentro do componente BPRI. Por exemplo, a entidade *ManagerEntity* cria uma nova instância da entidade *ProcessEntity*, sempre que um cliente faz uso da primitiva *create_process*(), dessa forma ao longo do tempo o Gerente de Processos passa a controlar um lista de identificadores correspondentes às diferentes instâncias de *ProcessEntity*; o segundo grupo (b) representa os vários atributos (variáveis) controlados pelas entidades de BPRI. Por exemplo, a entidade *ManagerEntity* é responsável pela tarefa de informar aos clientes de um componente BPRI a respeito do que é necessário para executar uma definição de processo controlada por ela, bem como por informar o que deve ser esperado como resultado após a execução do processo em questão. Estas informações correspondem aos atributos (variáveis) *context_signature* e *result_signature* e são representados através de instâncias do artefato ENTITY_DATA (do perfil EDOC-Entity) que descrevem listas de pares de informações (nome do atributo e nome do tipo) chamadas de *ProcessDataInfo*. Cada entidade do BPRI tem seus próprios atributos e é responsável por eles.

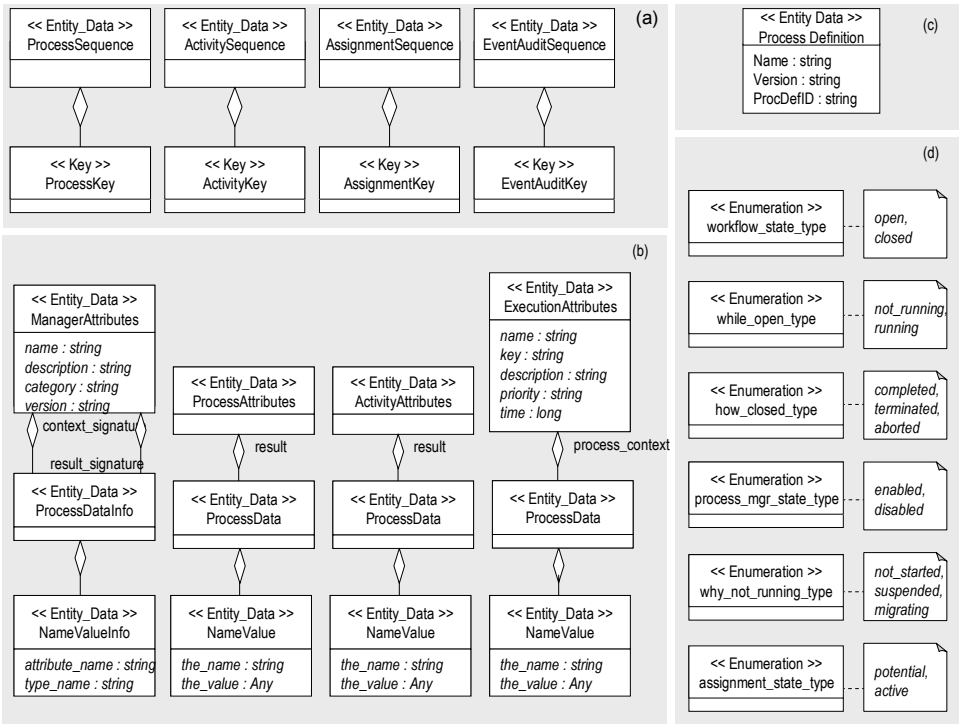

Figura 5-9 - Modelo de Informações com principais elementos (Parte 1)

O terceiro grupo (c) representa informações relacionadas com a definição de processos que foi obtida do repositório de modelos. Estas informações são usadas para quaisquer procedimentos que envolvam o repositório; o quarto grupo (d) representa os tipos enumerados associados com as variáveis de estados do component BPRI. *process_mgr_state* corresponde ao tipo enumerado do estado do componente como um todo, *workflow_state_type* corresponde tipo enumerado do

estado de um objeto em execução dentro do BPRI (com sub-estados descritos em *while_open_type*, *why_not_running_type* e *how_closed_type)* e assignment_state_type corresponde ao tipo enumerado do estado de uma reserva de recurso.

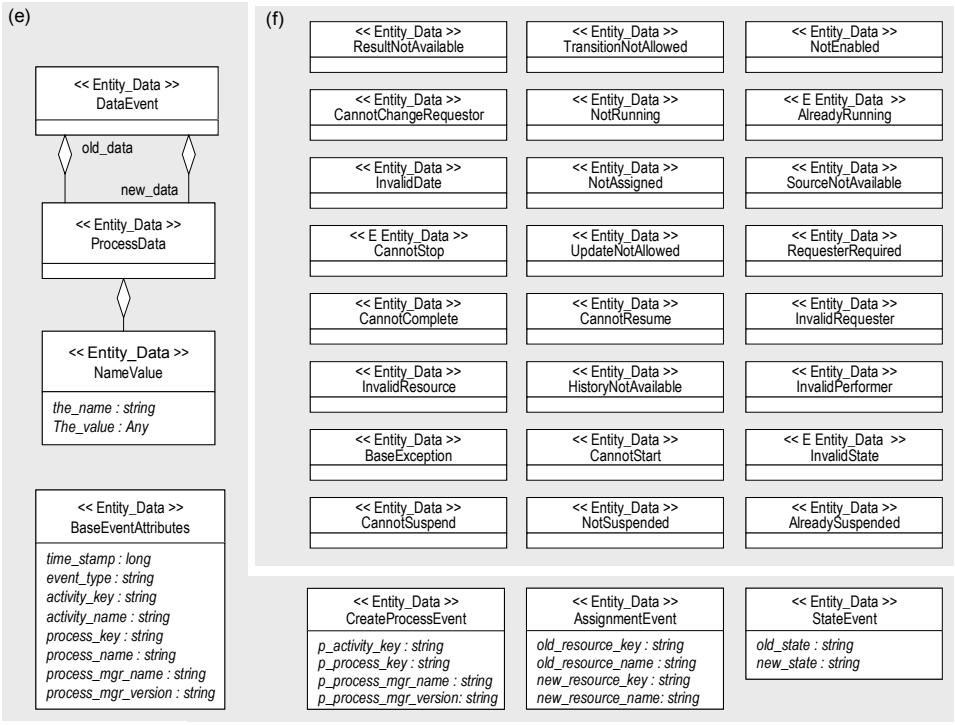

Figura 5-10 - Modelo de Informações com principais elementos (Parte 2)

A Figura 5-10 mostra a segunda parte do modelo de informação, apresentando os demais grupos de informações e tipos. Nela estão o quinto e sexto grupo de informações. O quinto grupo (e) representa as informações relacionadas com os vários eventos emitidos por um componente BPRI para os clientes do ambiente de execução implementado por ele. Entre as várias informações estão eventos que comunicam mudanças de contexto de execução (*DataEvent*). Estes possuem como atributos duas seqüências de pares (*nome/valor*) que representam instâncias de um tipo *ProcessData*, uma delas é equivalente aos valores do contexto antes da mudança e a outra equivale à nova seqüência. Existem eventos para mudanças de estados (*StateEvent*) que sinalizam a criação de uma instância de processo (*CreateProcessEvent*) e eventos genéricos (*BaseEventAttributes*). O último grupo de informações compreende as exceções utilizadas nos vários métodos que estão disponíveis nas interfaces modeladas. Uma destas exceções merece especial atenção, tendo em vista que representa uma coleção de erros em única estrutura permitindo que mais de um erro seja indicado através desta exceção. O seu nome é *BaseException* e ela é a mais comum de todas as exceções apresentadas.

5.3.4. Inter-relacionamentos entre especificações

Além do modelo de informação propriamente dito, a especificação informacional pode ser enriquecida com diagramas que mostram inter-relacionamentos entre as entidades da especificação computacional, suas interfaces e as várias informações mostradas nas figuras anteriores (Figura 5-9 e Figura 5-10).

Este tipo de diagrama é muito importante por dar uma visão dos pontos comuns entre as três especificações utilizadas no modelo PIM considerado (empresarial, informacional e computacional), permitido saber que informações são importantes para cada uma das entidades que compõem a perspectiva interna do componente BPRI. Por exemplo, na Figura 5-11, o modelo informacional é considerado do ponto de vista da entidade *ManagerEntity*. A entidade tem sua própria máquina de estados (p*rocess_mgr_state*) onde dois estados são possíveis: habilitada (*enabled*) ou desabilitada (*disabled*). Na condição em que está desabilitada a entidade faz com que o ambiente de execução fique indisponível, caso contrário, ele deve operar normalmente. A entidade possui um conjunto de variáveis públicas que são disponibilizadas para seus clientes (*requesters*), entre as quais estão as informações de contexto (*context_signature*) e resultado esperado (*result_signature*); possui uma estrutura de dados que representa uma coleção de instâncias de *ProcessEntity* (*ProcessSequence*) que foram criadas por *GerenteProcesso* e um *iterator*[13] para facilitar o acesso a esta coleção. As coleções estão normalmente associadas com os relacionamentos internos nos quais a entidade central participa. Como *GerenteProcesso* possui um relacionamento de criação com ProcessEntity é natural que esta possua um meio de acessar e controlar todas as instâncias que criou.

Outra informação importante utilizada por *ManagerEntity* é a estrutura de controle da definição de processo que está em uso na entidade em questão e que faz referência a um meta-objeto controlado por um gerente de repositório externo. Esta parte do modelo é nova e não foi definida anteriormente no OMG-WF.

A Figura 5-11 mostra que a interface implementada por *ManagerEntity* apresenta três grupos de métodos com seus respectivos parâmetros e exceções. O grupo indicado pela circunferência número "1" corresponde aos métodos que permitem acesso às variáveis que compõem a estrutura de atributos *ManagerAttributes*. Como todos os atributos são apenas de leitura, os métodos do grupo implementam apenas acesso aos valores dos atributos, sem permitir a alteração dos mesmos. O grupo indicado pela circunferência número "2" corresponde aos métodos que dão acesso à coleção de *ProcessEntity* que representa o relacionamento entre *GerenteProcesso* (fábrica de objeto) e *ProcessEntity* (objeto fabricado). Por fim, O grupo indicado pela circunferência número "3" corresponde às operações descritas como primitivas dos protocolos externos nos quais o componente BPRI participa.

[13] *Iterators – Iterators são componentes que auxiliam na navegação através de coleções de dados.*

Todas as operações de uma instância de Interface correspondem a portas de fluxos que fazem parte de um protocolo. Toda porta de fluxo está associada a instâncias de elementos *CompositeData* (Perfil EDOC-CCA) que representam seus parâmetros e eventualmente suas exceções.

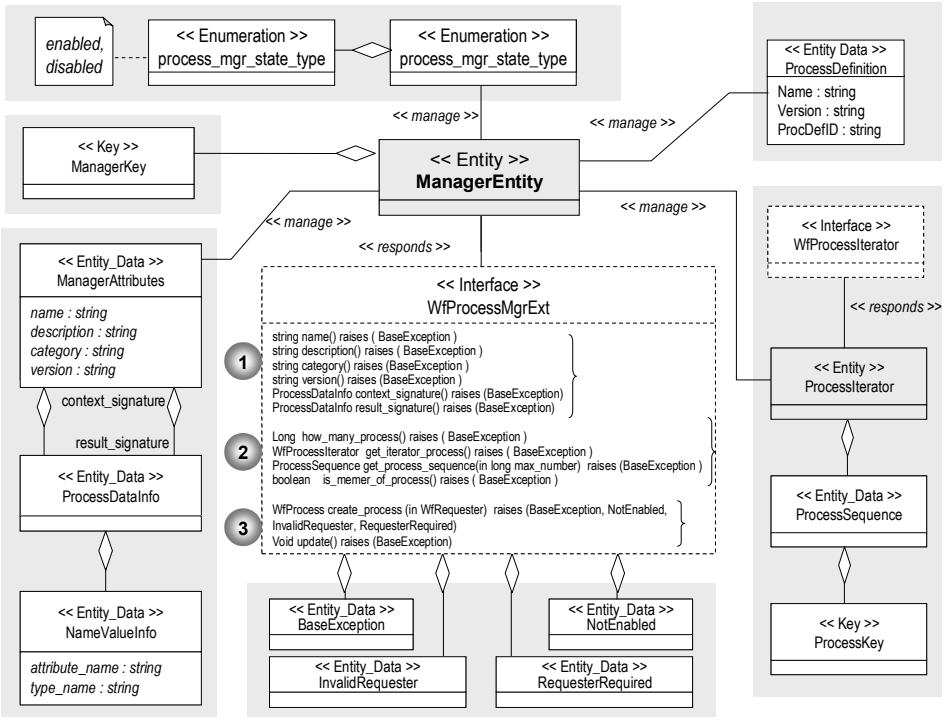

Figura 5-11 - Modelo Inter-relacional para a entidade *ManagerEntity*

5.4 Considerações sobre o Modelo

O modelo independente de plataforma proposto neste capítulo procura adaptar os principais conceitos utilizados na especificação OMG-WF para o novo contexto observado para os processos de negócios, acrescentando funcionalidade não disponível na especificação anterior. A Figura 5-12 mostra um resumo dos principais pontos considerados. Os pontos indicados pelas circunferências numeradas correspondem às principais contribuições feitas no modelo PIM para adicionar novas funcionalidades não disponíveis na especificação OMG-WF.

- A circunferência 1 corresponde à adição da interface de acesso a um repositório de modelos de definições de processos de negócios (*WfRepository*). Esta interface permite que um componente que foi gerado a partir do PIM BPRI possa acessar uma representação de seu comportamento feita a partir de perfis UML EDOC. Esta característica dá maior flexibilidade

para os componentes gerados, uma vez que estes não precisam ser re-compilados após modificações em suas definições de processos;

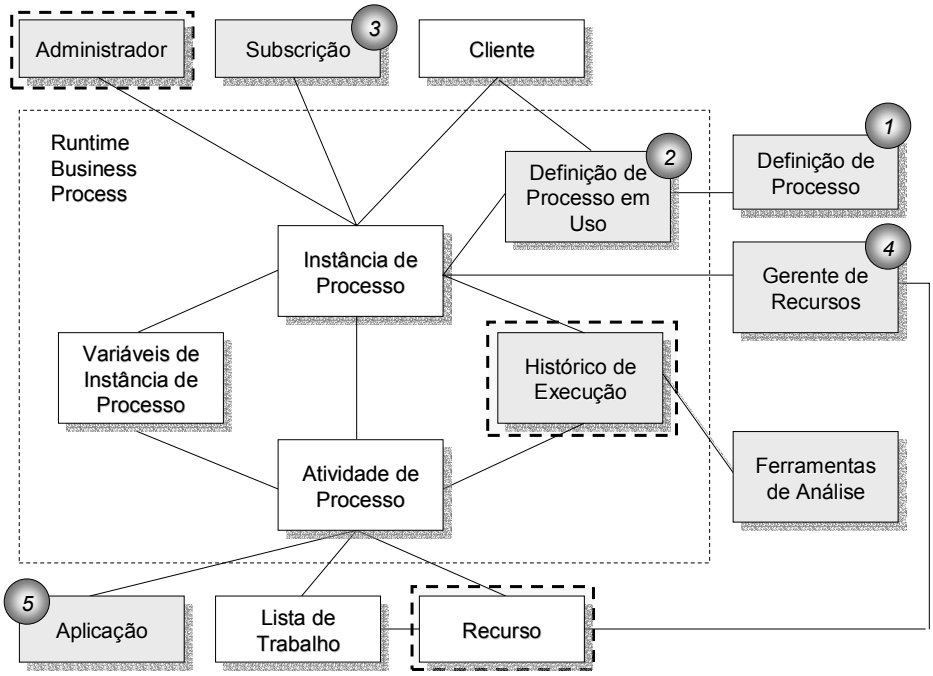

Figura 5-12 - Contribuições para o Modelo solicitado na RFP [OMG02d]

- A circunferência 2 corresponde à representação de uma definição de processo em execução (*WfProcessDefExt*) que permite a manipulação de um meta-objeto do repositório dentro de um objeto de execução correspondente a uma instância de processo. Esta característica deixa transparente para o componente como seus fluxos de controle e de dados são extraídos de uma definição de processo genérica;

- A circunferência 3 corresponde à adição de métodos de subscrição e cancelamento de subscrição para permitir que um determinado ambiente possa se cadastrar para receber eventos de um determinado componente. Esta característica permite que um administrador crie uma rede de subscrição entre vários ambientes de execução. Esta rede de subscrição é importante para permitir que definições de processos baseadas em eventos sejam usadas por um componente BPRI;

- A circunferência 4 corresponde à utilização do conceito de gerente de recursos, como uma forma de desacoplar o componente BPRI dos recursos utilizados por estes. Esta característica permite que o gerente de recursos esteja constantemente atualizando as referências para os melhores recursos

capacitados para serem usados por uma atividade. Neste processo são usados as expressões de critério de seleção e critério de criação que estão definidas dentro de um artefato *ProcessRole*; e

- A circunferência 5 corresponde à indicação de que as aplicações são acessadas através de recursos adaptadores (*adapters*) que fazem uso do gerente de recursos para permitir o acesso do componente BPRI às várias aplicações de um servidor de aplicações. Esta característica dispensa a utilização de um protocolo específico para as interações entre atividades e aplicações, além de contribuir para que esquemas de acoplamento fraco sejam mais facilmente implementados.

A Figura 5-12 também mostra algumas funcionalidades tratadas no modelo PIM BPRI (retângulos pontilhados) que embora não possuam interfaces específicas para o que é desejado na RFP estão embutidas nas interfaces anteriores do modelo de OMG-WF. Por exemplo, não há uma interface específica para o histórico, mas todo objeto de execução do BPRI que implementa a interface *WfExecutionObject* possui métodos para o acesso às informações de histórico de eventos (protocolo de execução - portas de monitoração – controle de coleções de eventos). Desta forma, uma interface de histórico *WfHistory* poderia ser uma simples generalização da interface *WfExecutionObject*. Algo parecido acontece com a funcionalidade desejada para um administrador. Um administrador deve ser capaz de monitorar e controlar objetos de execução dentro do componente BPRI. Sendo assim, ele é um caso especial de cliente que faz uso do protocolo de execução em suas portas de controle, monitoração e eventos. No caso dos recursos externos e suas listas de trabalho, o desacoplamento obtido com a utilização de um gerente de recursos externos extrai esta preocupação do componente BPRI e limita sua interação com os recursos através dos protocolos externos mostrados na Figura 5-1.

5.5 Resumo

Este capítulo apresentou uma proposta de modelo PIM para o ambiente de execução de processos de negócios. O modelo é descrito através dos artefatos definidos nos perfis UML da especificação EDOC e visualizado graficamente através das notações relacionadas com cada um destes artefatos, procurando atender aos principais pontos da RFP oficializada em [OMG02d]. Além disso, procura acrescentar suporte às duas formas de definir processos de negócios fornecidas na especificação EDOC: uma baseada em eventos (perfil de eventos) e uma baseada em conexões de atividades previamente definidas (perfil para processos de negócios). Ambas foram descritas no capítulo 3 através da apresentação dos perfis UML correspondentes a cada uma delas. Apesar das abordagens considerarem meta-modelos próprios com diferentes artefatos de

modelagem, a especificação EDOC sugere que nenhuma das duas é plenamente completa e que podem, eventualmente, serem combinadas para compor definições mais ricas. Mesmo assim, é fato que nem esta combinação entre os dois perfis é suficiente para definir em modelos um componente de processo de negócio [OMG02d]. Havendo também a necessidade da definição do ambiente de execução para a definição de processo que foi feita.

O modelo proposto foi definido com base a especificação OMG-WF e permite que o mesmo componente que seja gerado a partir deste possa fazer uso das duas formas de definição de processos de negócios da especificação EDOC ou que faça uso combinado destas definições. Para isso introduz um novo componente dentro da arquitetura de um workflow facility proposta de OMG-WF: o interpretador de definições EDOC (*ProcessDefEntity*) e mostra, de forma independente de plataforma, como este componente interno deve interagir com os componentes normais da arquitetura para que as instâncias dos artefatos dos dois perfis (perfil de eventos e perfil de processos de negócios) possam ser acessados e usados pelo componente modelado, sem a necessidade de alterações no código do componente, quando houverem mudanças em sua definição de processo.

No capítulo seguinte, o modelo proposto para o ambiente de execução é submetido aos cenários relacionados com o ciclo de vida de um modelo PIM MDA, através de um protótipo que implementa uma infra-estrutura de modelagem MDA.

Capítulo 6
Protótipos e Validação

6.1 Introdução

Este capítulo apresenta a parte final da metodologia deste trabalho, ou seja, o desenvolvimento das provas de conceitos para as principais idéias da tese e as validações necessárias para os protótipos que incorporam estas provas nos vários cenários considerados. A seção 6.2 apresenta uma visão geral das funcionalidades dos protótipos implementados; a seção 6.3 apresenta as principais funcionalidades do protótipo usado para criar uma infra-estrutura de suporte para a criação de PIM's e PSM's; a seção 6.4 discute os principais aspectos implementacionais do protótipo de infra-estrutura; a seção 6.5 mostra como o protótipo de componente de processo de negócio BPRI é gerado a partir da infra-estrutura definida, considerando os quatro cenários do ciclo de vida de um modelo MDA; e por fim, a seção 6.6 apresenta as considerações finais sobre os resultados obtidos com as implementações e validações realizadas.

6.2 Visão Geral

Conforme o planejado na metodologia discutida no capítulo 1, as idéias apresentadas nos capítulos anteriores são reunidas aqui na forma de protótipos que concretizam os requisitos necessários para os principais cenários apresentados. Esta concretização de idéias permite que importantes *provas conceituais* sejam obtidas para que as idéias deste trabalho possam ser comparadas com outras propostas semelhantes, bem como para dar prosseguimento ao estudo das mudanças necessárias na teoria dos sistemas de gerência de processos de negócios para sua perfeita adaptação à arquitetura MDA e sua metodologia de desenvolvimento.

Os protótipos estão intimamente relacionadas com os cenários definidos anteriormente e consideram a Figura 6-1 como ponto de partida para a definição das principais funcionalidades dos dois protótipos considerados: um para

concretizar o ambiente em que os perfis e modelos PIM e PSM são definidos e o outro que implementa um ambiente de execução de processos de negócios que foi gerado a partir do PIM criado no capítulo anterior.

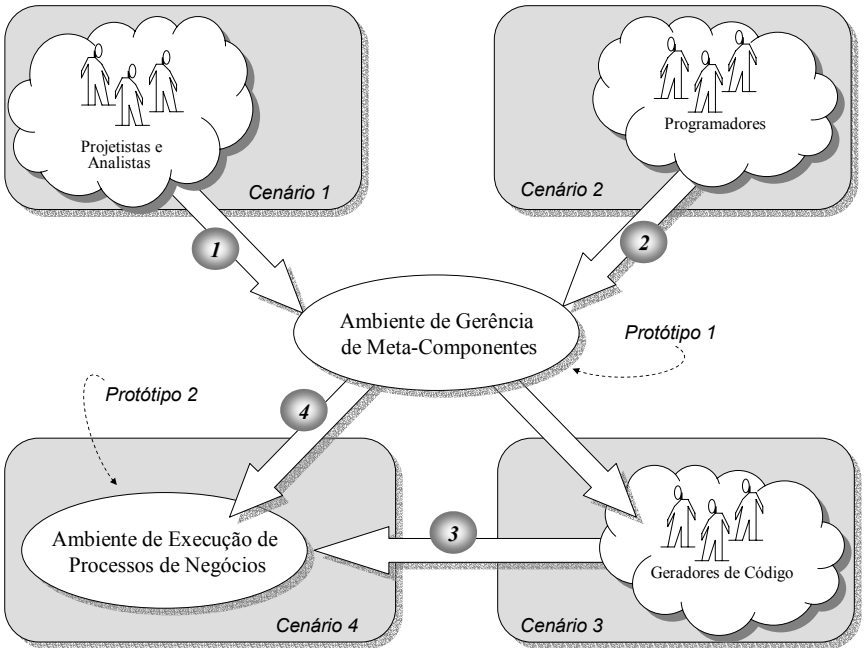

Figura 6-1 - Protótipos e Cenários Considerados

A Figura 6-1 apresenta o contexto em que os dois protótipos se relacionam para interagir com seus usuários nos quatro cenários fundamentais. O protótipo 1 é o responsável pela gerência das meta-informações utilizadas nas várias fases do projeto do PIM, definido no capítulo anterior, enquanto que o protótipo 2 é uma implementação de um sistema de gerência de processos de negócios capaz de atualizar sua definição de processos de negócios a partir do gerente de repositório que é implementado pelo AGMC. Para melhor referenciar os dois protótipos, vamos chamar o protótipo 1 de AGMC (Ambiente de Gerência de Meta-componentes) e o protótipo 2 de protótipo de componente BPRI (*Business Process Runtime Interface Component*).

No cenário 1, os perfis UML usados no projeto são definidos através de meta-modelos MOF; no cenário 2 os modelos PIMs que representam as definições de processo e o ambiente de execução são criados através da instanciação dos artefatos dos meta-modelos que representam os perfis UML; no cenário 3, modelos dependentes de plataforma são criados a partir de mapeamentos entre um PIM e uma plataforma final que tenha sido definida na forma de perfil UML (PDM); e finalmente no cenário 4, o BPRI é criado a partir das interfaces geradas do modelo PSM correspondente ao ambiente de execução de processos de negócios.

6.3 Principais Funcionalidades

Nesta seção, os principais elementos do AGMC são descritos de forma mais detalhada através de diagramas de caso de uso e de diagramas de pacotes, partindo de uma visão genérica do sistema até atingir um ponto em que as principais funcionalidades sejam reagrupadas para fazer parte de um único *framework*.

Um diagrama de caso de uso ([JAC94] e [RAT97]) define a funcionalidade de um sistema da perspectiva das entidades que interagem com o mesmo. Os principais elementos de um diagrama deste tipo são os atores e seus casos de uso. Eventualmente, o diagrama pode ser estendido para considerar também as comunidades de atores, os papéis, os relacionamentos e outros objetos cabíveis no contexto. Por outro lado, um diagrma de pacotes mostra como as classes que implementam um sistema são organizadas em pacotes que agrupam as classes relacionadas com a funcionalidade de um componente de software específico.

Considerando que um sistema que implementa uma infra-estrutura de suporte a MDA é também um ambiente de gerência de meta-informações, a Figura 6-2 mostra a estrutura de comunidades para um ambiente de gerência genérico deste tipo.

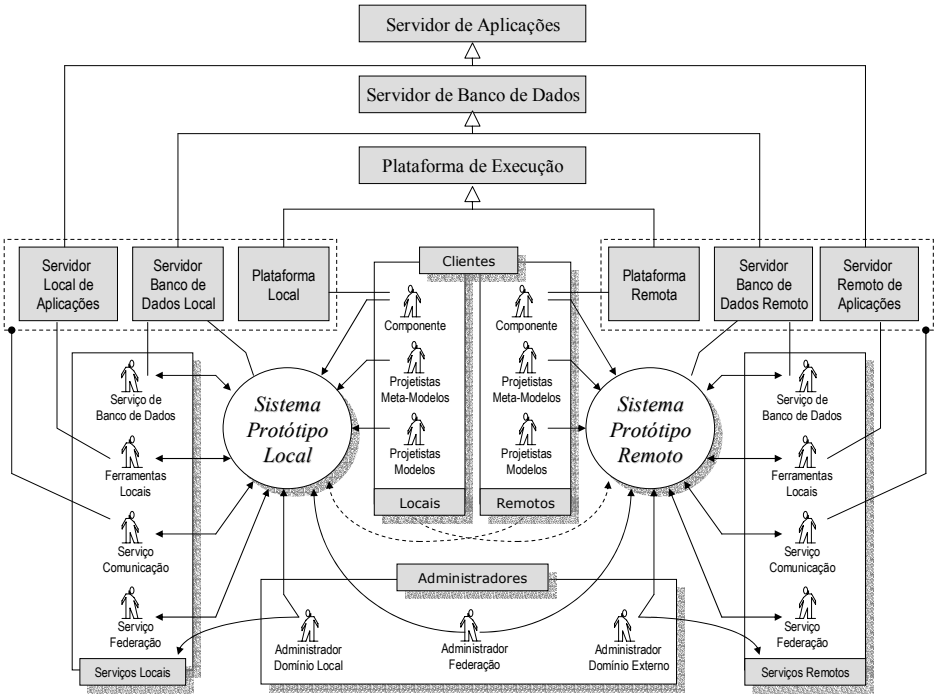

Figura 6-2 - Comunidades, objetos, papéis e relacionamentos

Na Figura 6-2, os retângulos em cinza representam servidores, os retângulos em branco representam comunidades de atores e as circunferências são duas

instâncias de um mesmo sistema de gerência de meta-informações implementado em nosso protótipo. Cada uma das comunidades tem seus próprios objetivos e diferentes elementos que representam papéis importantes no momento da definição de seus casos de uso. Podemos agrupar estas comunidades em três grupos: o grupo dos clientes, o grupo dos serviços e o grupo de administradores.

As comunidades de clientes são formadas por projetistas e componentes. Elas têm como objetivos principais: fazer uso do sistema para gerenciar meta-modelos e modelos, obter informações de configuração previamente definidas e receber notificações de re-configuração que possam ser relevantes para a execução adequada dos componentes que estão em diferentes plataformas de *middleware*; As comunidades de administradores são formadas por todos os atores que, de alguma forma, são responsáveis pelos diversos serviços e sistemas do contexto. Elas têm como objetivos principais: manter os principais sistemas em operação e devidamente configurados; O terceiro grupo de comunidades envolve as comunidades de serviços utilizados, direta ou indiretamente, pelo sistema de gerência de meta-informações.

A Figura 6-3 mostra casos de uso de alto nível com os principais relacionamentos externos do ambiente de gerência do protótipo AGMC. A princípio, três subsistemas seriam necessários para implementar as principais funcionalidades desejadas. São eles: o subsistema de controle, o sub-sistema de meta-modelos e o sub-sistema de modelos.

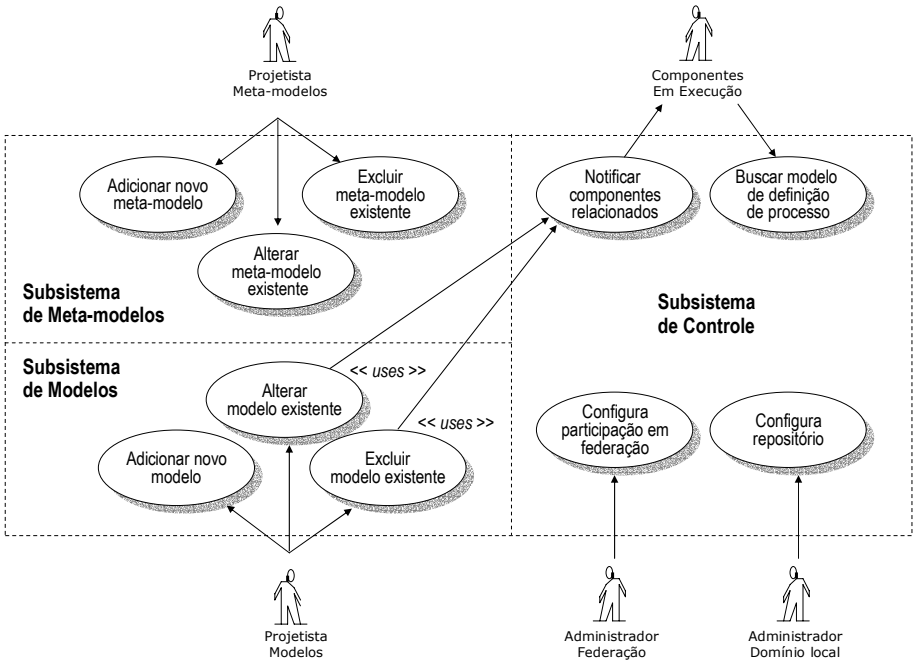

Figura 6-3 – Subsistemas Importantes para a Gerência de Meta-informações

O subsistema de controle é o responsável pelas interações entre os demais subsistemas internos, bem como pelas interações com os clientes e administradores. Ele gerencia o conjunto de coleções de meta-informações como um todo, deixando transparente para seus clientes qual subsistema é realmente o responsável por um determinado grupo de coleções. Uma coleção de meta-informações pode ser interpretada como um conjunto de modelos, meta-modelos ou meta-objetos armazenados no repositório do protótipo.

O subsistema de meta-modelos é o responsável pelas coleções de meta-modelos, sejam eles apenas extensões do meta-modelo UML, ou meta-modelos com linguagem de modelagem própria. Este subsistema tem a importante função de permitir a criação e gerência de perfis UML que serão utilizados na criação de PIM's e PSM's.

O subsistema de modelos é o responsável pelas coleções de modelos que são definidos pelos usuários a partir dos perfis UML previamente definidos. Ele é também responsável pela gerência dos meta-objetos que encapsulam os modelos relacionados com um determinado projeto.

6.3.1. Subsistema de Controle

Para uma melhor visualização da funcionalidade do Subsistema de Controle, o diagrama de pacotes da Figura 6-4 mostra uma visão interna do subsistema para apresentar seus principais elementos.

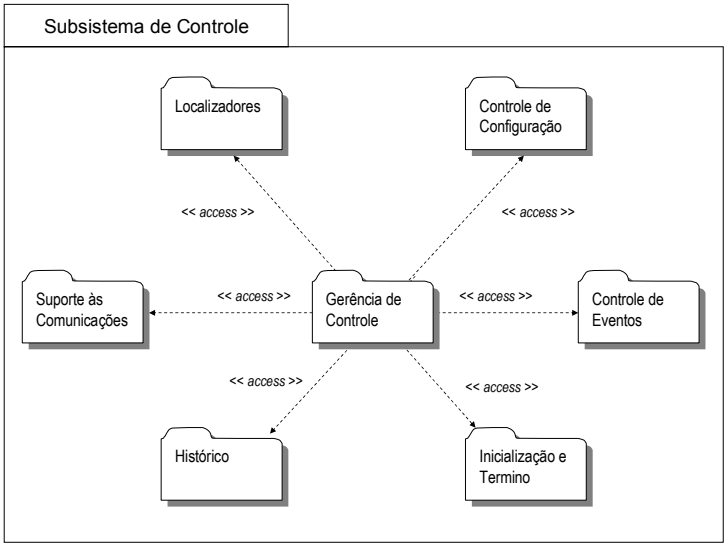

Figura 6-4 – Subsistema de Controle

Os sete elementos internos são componentes que agrupam funções vitais para gerência de coleções do repositório do protótipo. São eles:

- **Gerência de coleções –** São os componentes principais do subsistema de controle. Eles são o ponto de entrada para os clientes do protótipo, pois fazem o controle dos catálogos de coleções[14] do repositório, incluindo inclusive, os catálogos importados de sistemas protótipo remotos. Todos os outros grupos de componentes no subsistema de controle dão suporte a este componente em uma de suas necessidades.
- **Inicialização e Terminação –** São componentes responsáveis pela busca de informações de configuração, inicialização das conexões entre subsistemas e terminação amigável do protótipo como um todo.
- **Controle de Histórico –** São componentes responsáveis pelas funções de registro de eventos internos e informações de depuração das diferentes funções do subsistema.
- **Controle de Comunicações –** São componentes que permitem a utilização da infra-estrutura de comunicação externa.
- **Controle de Eventos –** Estes componentes são responsáveis pela publicação e subscrição de eventos relacionados com as coleções do repositório. Eles permitem, por exemplo, que um cliente se subscreva para receber notificações de alterações de meta-informações pertencentes a uma das coleções disponíveis no repositório do protótipo.
- **Controle de Configuração –** Este pacote contém os componentes usados na configuração do gerente de repositório que controla as coleções de meta-modelos e modelos.
- **Contole de Localizadores –** Os componentes localizadores implementam protocolos que permitem a uma federação de serviços de repositórios[15]. São vários os tipos de componentes localizadores, dependendo do componente escolhido na configuração inicial do protótipo, uma forma diferente de federação ser estabelecida. Por exemplo, para usar um serviço de *trading* baseado em CORBA, deve-se configurar o componente que implementa um cliente do CORBA *Trading Service* [OMG96]; para usar um serviço de nomes [INP00], deve-se configurar um componente que implementa um cliente do serviço de nomes e assim por diante.

Algumas características deste subsistema são particularmente dignas de nota. Por exemplo, embora os componentes da Gerência de Coleções sejam como índices para catálogos de coleções internos e externos de diversos níveis, uma vez

[14] *Catálogos de Coleções – são espaços de nomes (namespaces) para as referências que representam os meta-objetos que compõem uma coleção.*

[15] *Federação de serviços de repositórios – é uma federação de sistemas de gerência de meta-informações que compartilham os mesmos esquemas de bancos de dados e interagem entre si para permitir acesso uniforme e transparente a uma série de recursos geograficamente distribuídos nos repositórios da federação. É um conceito similar ao de federações de bancos de dados, pois os gerentes de repositórios são aplicações de bancos de dados [HAA02].*

fornecida uma referência de coleção para um cliente, este acessa a coleção de forma independente da Gerência de Coleções, ou seja, a Gerência de coleções não é intermediária no acesso, apenas na consulta de referências. Mesmo assim, qualquer alteração nestas coleções gera eventos que são tratados pela Gerência para manter a consistência dos catálogos.

É importante ressaltar que estas considerações são também válidas para clientes externos. Clientes externos são clientes que desejam usar uma meta-informação que faz parte de uma das coleções dos catálogos exportados pelo sistema local. Neste caso, a federação é usada apenas para viabilizar a troca de catálogos. Uma vez que um cliente externo já tenha lido em seu catálogo uma referência local, ele pode vir diretamente ao repositório local e acessar o que deseja.

6.3.2. Subsistema de Meta-modelos

O Subsistema de Meta-modelos também é descrito em maior detalhe no diagrama de pacotes da Figura 6-5. Os sete elementos internos são componentes que agrupam funções vitais para gerência de meta-modelos do repositório do protótipo. São eles:

- **Gerência de Meta-modelos** – São os componentes responsáveis pelos meta-objetos que representam meta-modelos, ou seja, eles usam o modelo MOF para representam os artefatos dos meta-modelos. O acesso a Gerência de meta-modelo é feito através de interfaces fixas definidas na especificação [OMG02a]. Estas interfaces são implementadas por objetos chamados de *Moflets M2*.

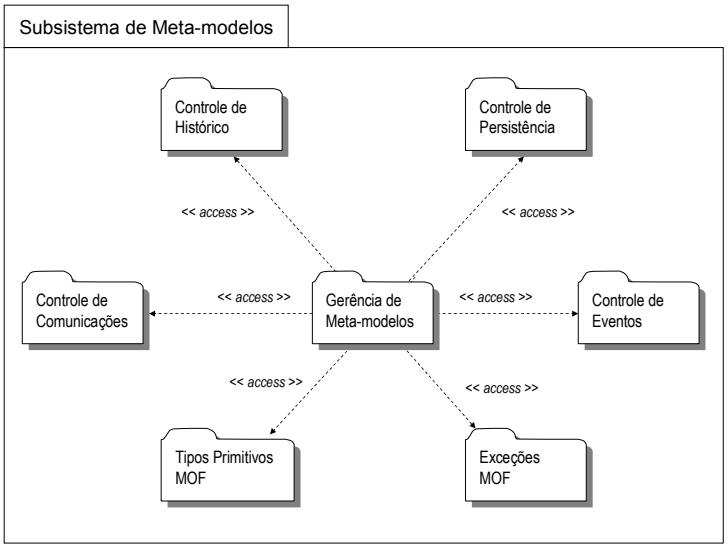

Figura 6-5 - Subsistema de Meta-modelos

- **Controle de Persistência –** Estes componentes permitem que os meta-objetos criados sejam armazenados em sistemas de bancos de dados, de forma, a não se perderem entre execuções distintas do protótipo. Dependendo da configuração feita, diferentes bancos de dados podem ser usados via SQL.
- **Controle de Eventos –** Têm a mesma função explicada no caso de uso anterior, ou seja, estes componentes são responsáveis pela publicação e subscrição de eventos relacionados com as coleções do repositório. Eles permitem, por exemplo, que um cliente se subscreva para receber notificações de alterações de meta-informações pertencentes a uma das coleções disponíveis no repositório do protótipo.
- **Controle das Comunicações –** Têm a mesma função do caso de uso anterior, ou seja, são componentes que permitem a utilização da infra-estrutura de comunicação externa.
- **Controle de Histórico –** Têm a mesma função do caso de uso anterior, ou seja, são componentes responsáveis pelas funções de registro de eventos internos e informações de depuração das diferentes funções do subsistema.
- **Tipos Primitivos MOF -** São componentes que implementam tipos primitivos MOF, como por exemplo: *Float, int, Long, Short, String, Fixed, Double, Char, Byte, Boolean, Any, TypeCode e Object.*
- **Exceções MOF –** São componentes que implementam exceções MOF usadas em moflets de meta-modelos e modelos. São exemplos de exceções deste tipo: *WrongType, featureWrongScope*, etc. Ver seção "*Exception Framework*" da especificação MOF [OMG02a] para ter uma descrição detalhada de cada exceção.

6.3.3. Subsistema de Modelos

O Subsistema de Modelos também é descrito em maior detalhe no diagrama de pacotes da Figura 6-5. Seus relacionamentos externos também continuam os mesmos da Figura 6-3. Neste caso de uso já podemos perceber algumas diferenças advindas do tratamento direto que é dado aos clientes locais e remotos, pois eles acessam diretamente o subsistema de modelos, uma vez que tenham obtido as referências dos pacotes em que estejam interessados. Os sete elementos internos são componentes que agrupam funções vitais para gerência de modelos do repositório do protótipo. São eles:

- **Gerência de Modelos –** São os componentes responsáveis pelos meta-objetos que representam as meta-informações dos modelos, ou seja, eles

representam os artefatos dos modelos usando meta-objetos que implementam as interfaces do meta-modelo associado (Moflets M1).

- **Controle de Persistência –** Têm a mesma função explicada no caso de uso anterior, ou seja, estes componentes permitem que os meta-objetos criados sejam armazenados em sistemas de bancos de dados, de forma, a não se perderem entre execuções distintas do protótipo. Dependendo da configuração feita, diferentes bancos de dados podem ser usados via SQL.

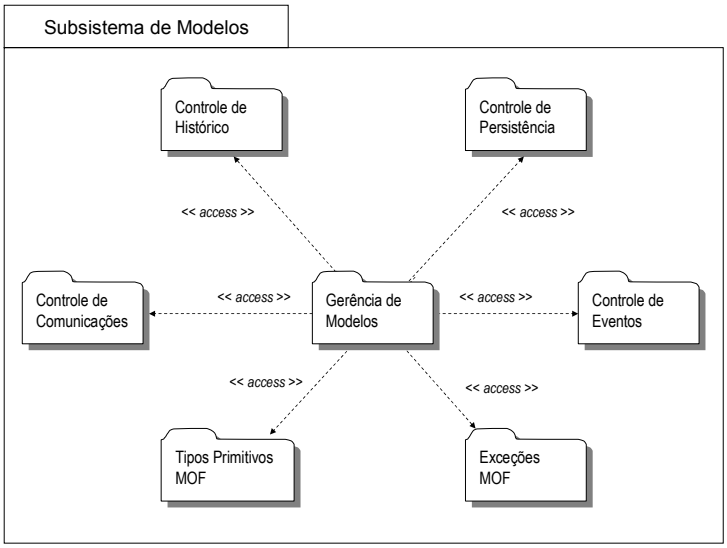

Figura 6-6 – Subsistema de Modelos

- **Controle de Eventos –** Têm a mesma função explicada no caso de uso anterior, ou seja, estes componentes são responsáveis pela publicação e subscrição de eventos relacionados com as coleções do repositório. Eles permitem, por exemplo, que um cliente se subscreva para receber notificações de alterações de meta-informações pertencentes a uma das coleções disponíveis no repositório do protótipo.
- **Controle de Comunicações –** Têm a mesma função do caso de uso anterior, ou seja, são componentes que permitem a utilização da infra-estrutura de comunicação externa.
- **Controle de Histórico –** Têm a mesma função do caso de uso anterior, ou seja, são componentes responsáveis pelas funções de registro de eventos internos e informações de depuração das diferentes funções do subsistema.
- **Tipos Primitivos MOF** - Têm a mesma função do caso de uso anterior, ou seja, são componentes que implementam tipos primitivos MOF, como por exemplo: *Float, int, Long, Short, String, Fixed, Double, Char, Byte, Boolean, Any, TypeCode e Object.*

- **Exceções MOF –** Têm a mesma função do caso de uso anterior, ou seja, são componentes que implementam exceções MOF usadas em *moflets* de meta-modelos e modelos. São exemplos de exceções deste tipo: *WrongType*, *FeatureWrongScope*, etc. Ver seção "*Exception Framework*" da especificação MOF [OMG02a] para ter uma descrição detalhada de cada exceção.

6.3.4. Funcionalidade em um único framework

A Figura 6-7 descreve, em termos gerais, como o framework seria inserido no contexto descrito pelos diagramas de pacotes apresentados na seção anterior (Figura 6-4, Figura 6-5 e Figura 6-6). O framework basicamente agrupa os principais componentes comuns de cada um dos diagramas juntamente com os componentes de inicialização, configuração e terminação. Desta forma, as gerências de meta-objetos de diferentes níveis fariam uso de um ambiente comum, o framework do protótipo AGMC, para realizar as funcionalidades descritas anteriormente para cada uma delas. Um aspecto importante desta forma de agrupar os principais componentes funcionais é que os componentes relacionados com eventuais mapeamentos tecnológicos (correspondentes à gerência de modelos e meta-modelos para uma determinada plataforma) são colocados fora do framework, pois podem ser gerados automaticamente por diferentes ferramentas enquanto não se desenvolve um módulo de geração próprio para estes mapeamentos no protótipo usado no trabalho. Além disso, como uma das diretrizes do trabalho (seção 6.3) é a interoperação com outras ferramentas baseadas em MOF, manter compatibilidade neste aspecto passa a ser uma necessidade.

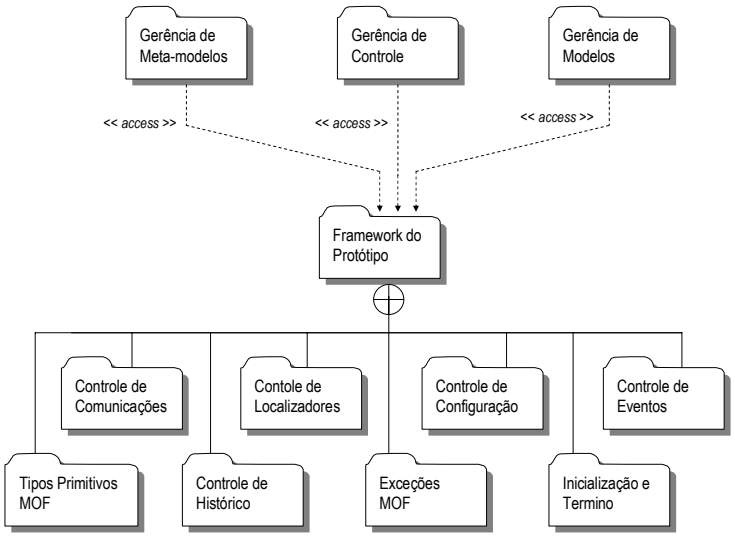

Figura 6-7 - Visão do protótipo e seu framework

6.4 Aspectos de Implementação

Um dos principais direcionamentos no desenvolvimento do protótipo AGMC, foi o de criar um ambiente que pudesse servir de base para a aplicação dos conceitos de MDA no projeto do PIM para um ambiente de execução de processos de negócios. Mais ainda, o protótipo deveria ser compatível com a ferramenta usada para gerar os moflets para os diversos meta-modelos considerados: a ferramenta dMOF do DSTC [DST00]. Os moflets implementam as interfaces do mapeamento tecnológico entre artefatos de meta-modelos e suas interfaces e implementações. Desta forma, o código fonte gerado pela ferramenta dMOF para os meta-modelos correspondentes aos perfis UML poderiam ser compilados com o framework do AGMC, já que este não dispõe de ferramentas para gerar estas interfaces a partir de meta-modelos. Como conseqüência, a implementação do AGMC foi feita na mesma linguagem da ferramenta geradora de moflets, ou seja, na linguagem Java.

Nas subseções seguintes, alguns esclarecimentos detalham melhor o AGMC e seu contexto, bem como sua relação com o conceito de moflet e os principais componentes de seu framework. São eles: o grupo de Controle de Persistência, o grupo de Localizadores e o grupo de Controle de Eventos.

6.4.1. O AGMC e o Conceito de Moflet

Um importante esclarecimento deve ser feito para melhor explicar a papel do conceito de *moflets* que foi introduzido na documentação da ferramenta dMOF [DST00] e o contexto em que estes são usados no protótipo AGMC. Revisite-se mais uma vez a Figura 6-7 e considere-se que os componentes de Gerência de Meta-modelos e Gerência de Modelos possam ser vistos como implementações de artefatos que representam meta-modelos e modelos sobre *runtimes* que permitam a execução de suas funcionalidades numa determinada plataforma de acesso.

Além disso, considere-se que várias ferramentas são capazes hoje de gerar implementações em código fonte para estes artefatos. No protótipo AGMC, os *runtimes* das principais ferramentas são substituídos por diferentes *runtimes* do AGMC para que os códigos fontes de moflets gerados por diferentes ferramentas possam ser usados em conjunto. A Figura 6-8 ilustra o relacionamento entre meta-modelos e seus artefatos com os moflets que implementam suas funcionalidades especificadas no MOF. O caso acima ilustra o caso dos moflets associados com artefatos de meta-modelos que são usados para definir modelos. São os chamados moflets M1. Existem também os moflets relacionados com artefatos do modelo MOF que são usados para definir Meta-modelos. São chamados de Moflets M2. Para maior simplicidade, apenas um tipo de moflet (M1) é usado nas ilustrações desta seção.

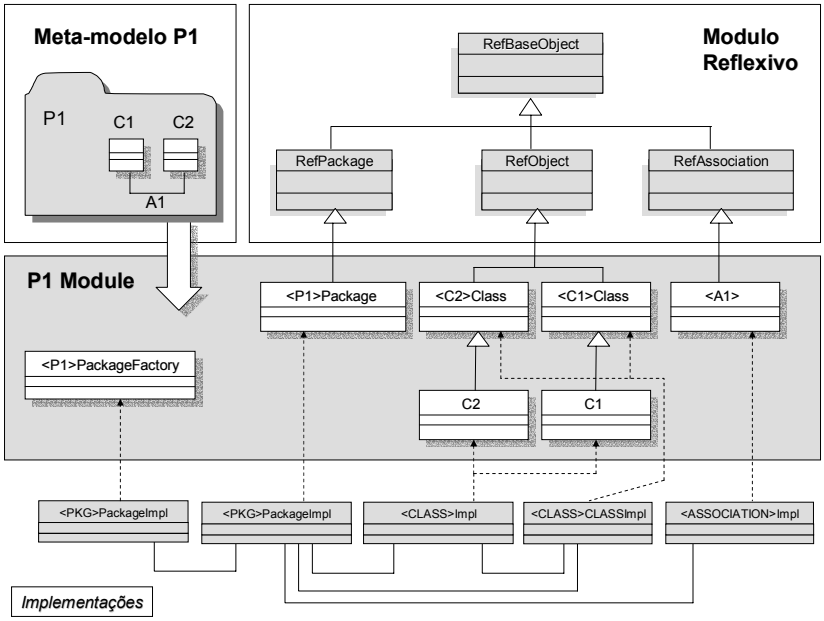

Figura 6-8 - Relacionamento entre meta-modelos e Moflets

6.4.2. Controle de Persistência

O Controle de Persistência, como o próprio nome diz, é o responsável pelo armazenamento persistente de meta-objetos que representam meta-informações. Isto significa que embora o sistema implemente uma arquitetura de meta-níveis em memória, o estado dos meta-objetos de cada um dos níveis deve ser conservado entre uma finalização do sistema em um dado instante e a sua próxima execução. Sendo assim, é natural a utilização de bancos de dados por parte do Controle de persistência. Esta implementação, faz uso de um serviço de armazenamento via JDBC (*Java Database Connectivity*). A JDBC é uma especificação de uma API (*Application Program Interface*) para permitir que programas escritos em Java possam acessar bancos de dados usando SQL.

A função de um serviço de armazenamento via JDBC é permitir o armazenamento e recuperação dos meta-objetos de forma automática e segura. A Figura 6-9 mostra as principais classes relacionadas com o controle de persistência e os relacionamentos com componentes fora do framework do AGMC. Na figura, a principal classe é a classe Persistência, pois é ela que gerencia o armazenamento dos meta-objetos em memória ou em bancos de dados através do serviço JDBC implementado na *Meta-Objetos Backend.*

Duas considerações são importantes neste grupo de componentes. A primeira diz respeito ao tratamento especial que deve ser dado aos meta-objetos que representam as associações, ou seja, os relacionamentos entre artefatos de um meta-modelo.

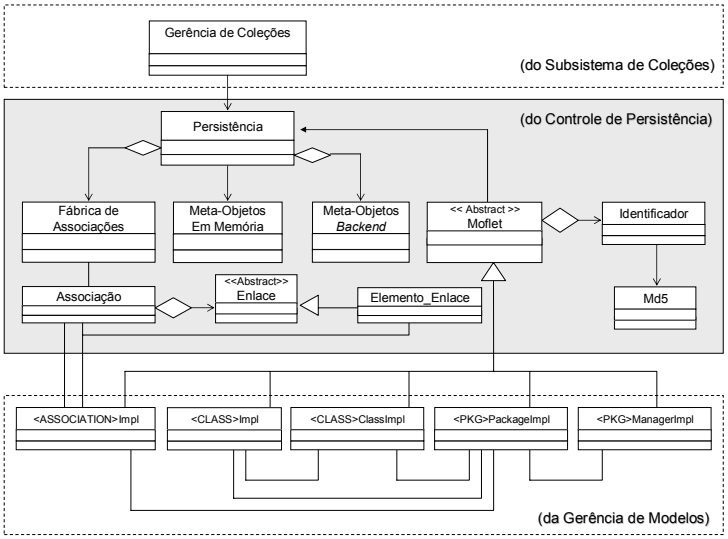

Figura 6-9 - Diagrama de classes para Controle de Persistência de modelos

Considere a Figura 6-10 para acompanhar esta discussão e vamos relembrar alguns conceitos do capítulo 2 relacionados com a instanciação entre níveis adjacentes de uma arquitetura de gerência de meta-informações.

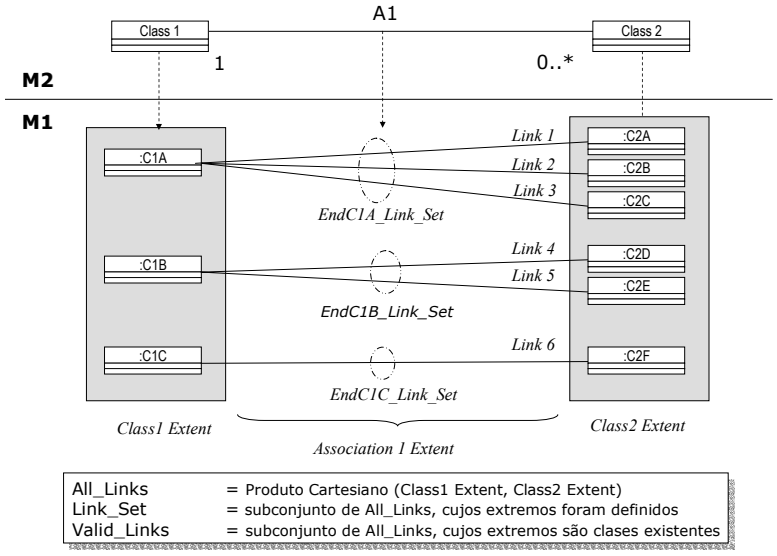

Figura 6-10 - Associações e Enlaces (Links)

Na Figura 6-10 são apresentados dois meta-objetos classe (*Class* 1 e *Class* 2) que se relacionam através de uma associação (*A1*) com multiplicidade definida nos extremos como: "1" e "0..*". Isto significa, por exemplo, que a associação é uma agregação e que as instâncias de *Class* 1 podem estar relacionadas com 0 ou mais instâncias de *Class* 2. Considerando os casos apresentados na figura e a

multiplicidade em questão, podemos ver que uma instância de associação pode implicar em vários enlaces (links) entre Class 1 e Class 2. Esta complexidade adicional no trato das instâncias de associações é o que faz com que elas sejam tratadas de forma particular pelo controle de persistência. De qualquer forma, tanto os meta-objetos de associações como os demais tipos existentes, são persistidos pelo mesmo serviço JDBC implementado pela classe *Meta-objetos backend*. A diferença está na maior complexidade dos objetos de associação e na forma como os meta-objetos de associação são gerenciados à medida que novos enlaces são adicionados, modificados e/ou excluídos para alterar uma instância de associação.

A segunda consideração diz respeito à necessidade de identificar, de forma única e permanente, todo e qualquer meta-objeto MOF. Como a especificação MOF apenas recomenda um esquema genérico com elementos variáveis, esta questão fica a critério do implementador. A forma geral recomendada é: "<prefixo> : <parte específica>". Como por exemplo nas duas formas abaixo, que são usadas no DCE (*Distributed Computing Environment*) – da IBM:

- "*DCE:d59809ab-124a-121a-55c2-08000343ab1c*" ou
- "*DCE: d59809ab-124a-121a-55c2-08000343ab1c:1234*".

Observe-se que na parte especifica do esquema de identificação única podem entrar as mais diversas informações que venham a contribuir para a unicidade da identificação. O esquema adotado na implementação do protótipo é um pouco diferente e tem as seguintes características:

> *"AGMC : <4 bytes para Endereço IP> : <4 bytes para número aleatório> : <5 bytes para tempo obtido por System.currentTimeMillis()> : <número de controle interno>"*

O tempo obtido pela função *System.currentTimeMillis()* é a diferença, medida em mili-segundos, entre o tempo atual e a meia noite do dia 1 de janeiro de 1970. Todos os campos do esquema de identificação são codificados em hexadecimal para maior facilidade de manipulação, mas existe também a possibilidade de codificar cada uma das partes do identificador através do algoritmo MD5 (*Message Digest 5*), dependendo da configuração definida para o AGMC.

Embora haja liberdade para as diversas implementações que geram classes semelhantes aos moflets das ferramentas dMOF ou UREP, nosso framework de implementação permite interoperação para estas diversas variedades de moflets por funcionar como um *runtime* que altera, de forma transparente, uma série de aspectos dos vários códigos fontes gerados. Pois ele é compilado juntamente com os códigos fontes gerados para adaptá-los à arquitetura do protótipo AGMC. Desta

forma, a convergência para um único formato de Identificador de meta-objetos poderia ser feita sem problemas.

6.4.3. Localizadores

Este grupo de componentes tem como objetivo principal acessar um esquema de federação de repositórios para intercambiar catálogos de meta-objetos e permitir a clientes de uma instância de AGMC acessar meta-objetos de outra instância. Este grupo pode ser formado de um ou mais objetos localizadores, dependendo de quais tipos de federações são suportados. Por exemplo, se forem usados localizadores baseados na função de Trading RM-ODP [ITU97], o conjunto de interfaces correspondente a um localizador é mostrado na Figura 6-11 e deve considerar os métodos de quatro interfaces abstratas que representam as principais funcionalidades de um membro de federação. São as interfaces de: estabelecimento de federação, interações de federação, gerência de contexto[16] e gerência de contratos[17]. Os contratos documentam os acordos entre pares de localizadores de uma mesma federação para a importação de serviços de repositório disponível nos elementos envolvidos.

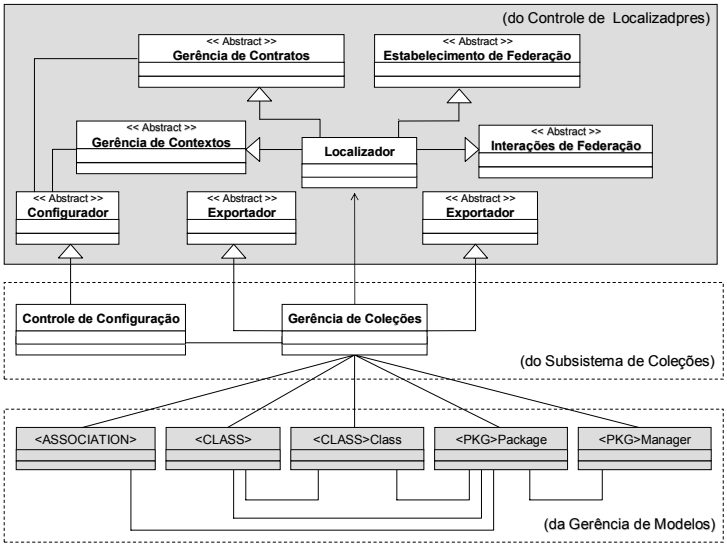

Figura 6-11 - Diagrama de classes para localizador e relacionamentos.

Os contextos representam espaços de nomes onde os catálogos que contém as coleções de referências para meta-objetos são disponibilizados para clientes

[16] *Contextos - são catálogos que representam espaços de nomes diferentes para as coleções de um repositório.*

[17] *Contratos – são acordos entres dois elementos de uma federação para a importação de serviços ou objetos disponíveis em seus catálogos.*

externos. O usuário de um localizador é, do ponto de vista do AGMC, o componente Gerente de Coleções, sendo assim ele exerce as funções de exportador e importador de referências para os meta-objetos.

A Figura 6-12 mostra um diagrama de relacionamento ilustrado com os principais atores e interfaces consideradas, mostrando também alguns métodos disponíveis nestas interfaces, considerando-se localizadores que implementam esquema próprio de federação.

Na figura, o administrador de federação faz uso de uma ferramenta de configuração que implementa a interface de gerência de contratos para estabelecer a federação usando as interfaces de estabelecimento de federação implementadas pelo localizador. Certamente que outros administradores de federação participam neste processo e os administradores locais das instâncias de AGMC participam com a definição de contextos e definições gerais necessárias para a instância local.

Uma vez definida uma federação, os clientes de um AGMC acessam seus catálogos e os catálogos importados da federação de forma transparente para que quando uma referência externa for necessária e já tiver sido importada pelo localizador local, seja suficiente para o cliente acessar diretamente os moflets remotos ao fazer uso da referência fornecida.

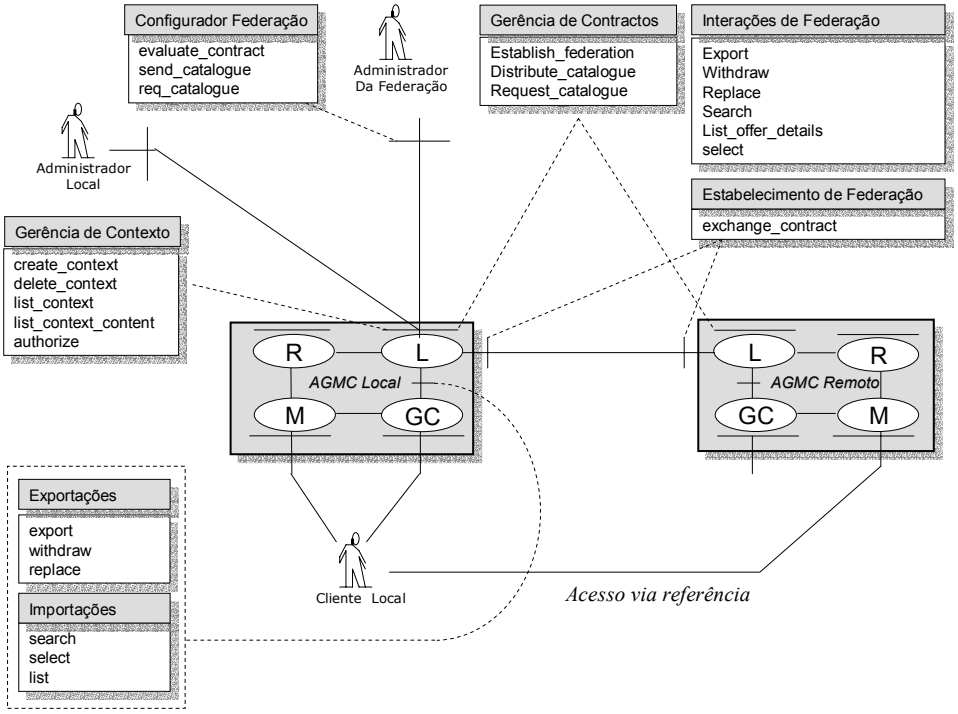

Figura 6-12 - Diagrama para Localizador implementando federação.

Uma alternativa de federação é a utilização de componentes que fazem uso de esquema de federação externo. Neste caso, apenas as interfaces de exportação e importação são utilizadas pelo localizador, ficando todo o resto das

funcionalidades a cargo dos agentes da federação. O esquema de federação da Figura 6-13 permite componentes localizadores mais simples, mas tem a desvantagem de não ser flexível no que diz respeito a forma de definir a federação. De qualquer forma a implementação do AGMC como um framework de componentes permite que novos componentes localizadores sejam projetados e disponibilizados no framework posteriormente sem a necessidade de se re-projetar o mesmo.

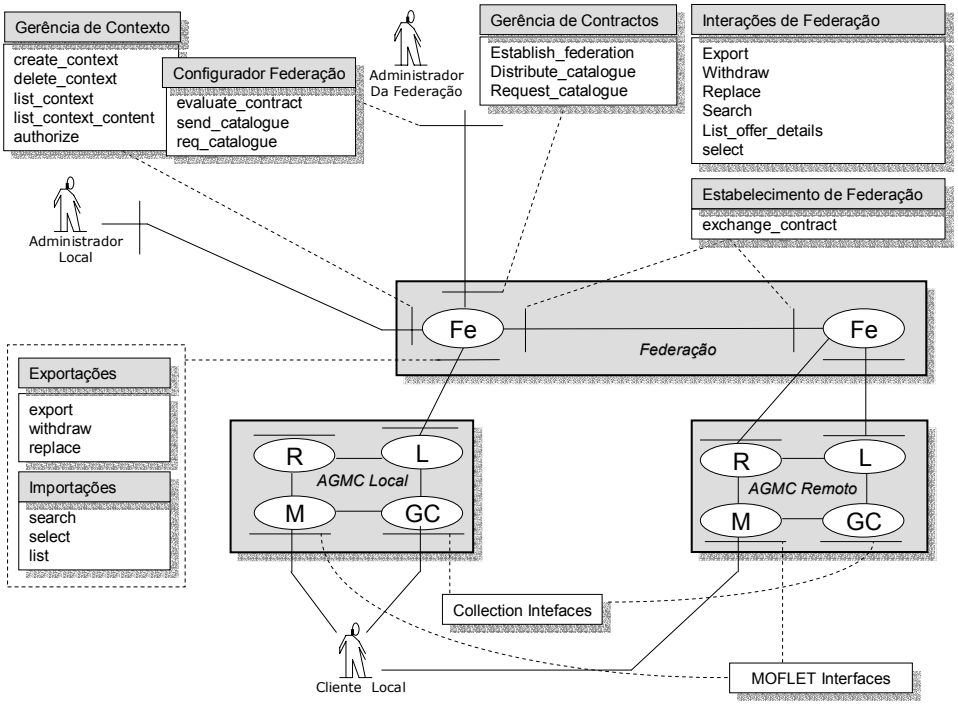

Figura 6-13 - Diagrama para Localizador com federação Externa.

6.4.4. Controle de Eventos

O controle de eventos implementa um serviço de eventos que adota um modelo de comunicação do tipo *Push*. No modelo de comunicação *Push*, os objetos fornecedores de eventos controlam o fluxo de dados de eventos através do envio de eventos para os consumidores que se conectaram ao controle de canal. O controle de canal é uma implementação de um canal de eventos específico. A Figura 6-14 mostra um esboço dos principais componentes envolvidos neste processo de geração e consumo de eventos. Observe que os moflets são os principais fornecedores de eventos no protótipo AGMC, pois eles estão associados com os modelos que são usados em tempo de execução pelos componentes BPRI.

O Controle de eventos tem um papel importante no processo de sinalização de componentes externos que fazem uso de reconfiguração dinâmica para atualizarem os modelos que são usados durante sua execução normal. Para que

isto seja possível, os componentes devem ser gerados com funcionalidades adicionais que permitam o acesso em tempo de execução ao AGMC e a utilização dos canais disponíveis nos componentes do Controle de eventos. É o que veremos em mais detalhes na próxima seção.

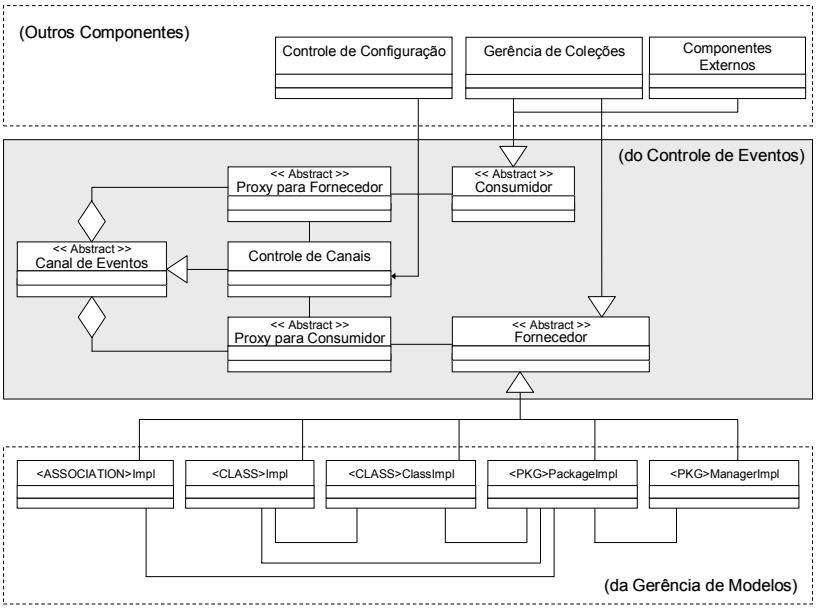

Figura 6-14 - Diagrama de classes para o Controle de Eventos.

6.5 Principais Cenários

Uma vez criado o protótipo que incorpora o ambiente de gerência de meta-informações (AGMC), o próximo passo é usá-lo para submeter o modelo PIM desenvolvido no capítulo anterior aos vários cenários considerados na metodologia descrita anteriormente e verificar sua validade para cada um deles.

Neste sentido, a Figura 6-15 faz um apanhado geral dos aspectos implementacionais relacionados com o desenvolvimento do protótipo, mostrando as relações entre as suas principais funcionalidades e as partes externas do AGMC que as implementam. Estas partes compreendem: aos serviços do AGMC, seu programa servidor, seu framework e os vários *runtimes* que permitem que diferentes moflets possam ser integrados no AGMC. Este *runtime* só não será necessário quando o AGMC for capaz de gerar seus próprios moflets para serem utilizados na arquitetura. O que nesta fase do trabalho ainda não é possível.

Os serviços devem incluir interfaces gráficas para : a definição de perfis UML representados através de modelos (“1”); a definição de pacotes de modelos PIMs e PSMs a partir de instâncias de artefatos correspondentes aos meta-modelos previamente definidos (“2”); a definição de mapeamentos que permitam a geração de código final a partir de um determinado PSM (“3”); e a interação com

componentes em tempo de execução("4"). Como o protótipo em sua fase inicial faz uso maciço dos moflets gerados a partir da ferramenta dMOF, os primeiros serviços são basicamente programas clientes baseados em moflets que não dispõem ainda de interfaces gráficas amigáveis, mas que são capazes de cumprir a funcionalidade esperada nos cenários considerados.

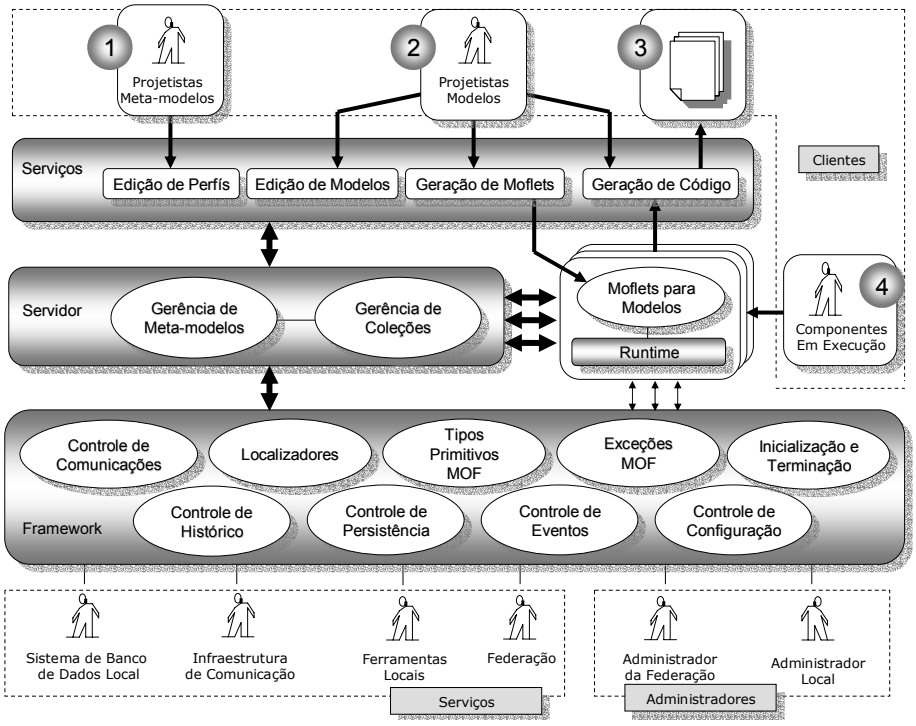

Figura 6-15 – O AGMC e os cenários considerados

6.5.1. Cenário 1 – Criação de Perfis UML

Conforme já foi discutido no capítulo 4, a abordagem feita aqui considera a utilização de meta-modelos MOF para representar os vários perfis UML que são necessários para a criação de um PIM ou de um PSM. Para que isto seja possível, os perfis UML devem ser transformados, de acordo com as regras de mapeamento definidas no perfil UML para MOF proposto na especificação EDOC. Além disso, como todo perfil UML é, na realidade, uma extensão do meta-modelo UML, um subconjunto deste deve também ser representado na forma de um meta-modelo MOF que possa ser importado pelos demais meta-modelos que representam perfis.

Uma vez que tenham sido definidos todos os meta-modelos que vão fazer parte do AGMC, estes são representados formalmente através de uma linguagem de descrição de meta-modelos denominada MODL (*meta-object description language*) para depois serem processados por uma ferramenta externa [DST00] que vai

interpretá-los para gerar as interfaces e implementações (moflets) correspondentes ao mapeamento tecnológico descrito na especificação MOF que foi discutido no capítulo 2. A Listagem 6-1 mostra um exemplo de representação em MODL (parcial) para o meta-modelo MOF correspondente ao perfil UML para CCA. O diagrama de classes correspondente é mostrado no apêndice A.

Listagem 6-1 – Listagem parcial do meta-modelo para CCA

```
package EDOC {
  import UML14;

  package CCA  {
     enum GranularityKind      {QUOTE "program", QUOTE "owned", QUOTE "shared"};
     enum DirectionType        {QUOTE "initiates",QUOTE "respondes"};
     enum PseudoStateKind      {QUOTE "choice",QUOTE "fork",QUOTE "initial",QUOTE "join",QUOTE
                                 "success",QUOTE "failure"};
     enum Status               {QUOTE "successful",QUOTE "timeoutFailure",QUOTE "technicalFailure",QUOTE
                                 "businessFailure",QUOTE "anyFailure",QUOTE "anyStatus"};
      typedef string           expression_t;

    abstract class Choreography : UML14::BehavioralElements::StateMachines::StateMachine {
     reference connections to abstractTransitionType of has_connections;
     reference nodes      to nodeType            of has_nodes;
    };
    abstract class UsageContext {
     reference portsUsed to portUsageType of has_portsUsed;
    };
    abstract class PortOwner {
     reference ports to portType of has_ports;
    };
    abstract class Composition : Choreography, UML14::BehavioralElements::Collaborations::Collaboration {
     reference compositionUses to componentUsageType of has_composition_uses;
     reference bindings to contextualBindingType of has_bindings;
    };

...

    class ProcessComponent : UsageContext, Composition, PortOwner, UML14::Foundation::Core::Classifier  {
     attribute GranularityKind granularity;
     attribute boolean isPersistent;
     attribute string primitiveKind;
     attribute string primitiveSpec;
     reference properties to propertyDefinitionType of has_properties;
    };
    class CommunityProcess : Composition {
    };
    abstract class Port : QUOTE "UML14::Foundation::Core::Class" {
     attribute boolean          isSynchronous;
     attribute boolean          isTransactional;
     attribute DirectionType    direction;
     attribute Status           postCondition;
    };
    class MultiPort : PortOwner, Port  {
```

```
  };
  class FlowPort : Port {
    reference type to dataElementType of has_type;
  };
  class InitiatingRole {
    attribute string name;
  };
  class RespondingRole {
    attribute string name;
  };
  class Protocol : PortOwner, Choreography, QUOTE "UML14::Foundation::Core::Class"  {
    reference initiator to initiatingRoleType of has_initiator;
    reference responder to respondingRoleType of has_responder;
  };
  class QUOTE "Interface" : Protocol {
  };
  class ProtocolPort : Port  {
    reference usesProt to protocolType of has_protocol;
  };
  class OperationPort : Port, PortOwner {
  };

…

  association has_protocol {
    end single  ProtocolPort protocolPortType;
    end single  Protocol     protocolType;
  };
  association has_initiator {
    end single       Protocol       protocolType;
    end bag [0..1] of InitiatingRole initiatingRoleType;
  };
  association has_responder {
    end single       Protocol       protocolType;
    end bag [0..1] of RespondingRole respondingRoleType;
  };
  association has_type {
    end set [0..*] of FlowPort    flowPortType;
    end single       DataElement dataElementType;
  };
  association has_properties {
    end single       ProcessComponent    processComponentType;
    end set [0..*] of PropertyDefinition  propertyDefinitionType;
   };
  association has_composition_uses {
    end single       Composition    compositionType;
    end set [0..*] of ComponentUsage componentUsageType;
   };
  association has_bindings {
    end single       Composition       compositionType;
    end set [0..*] of ContextualBinding contextualBindingType;
   };
  association has_ports {
    end single       PortOwner portOwnerType;
    end set [0..*] of Port      portType;
   };
```

```
...
  };  // end CCA profile package
...
};  // end EDOC profiles package
```

Após a criação dos moflets correspondentes ao meta-modelo do exemplo, as classes e associações da Listagem 6-1 passam a ter um conjunto de interfaces que permitem a criação de suas instâncias pelas diversas aplicações que fazem uso do AGMC para criar modelos a partir dos meta-modelos correspondentes aos perfis UML. É importante que fique claro que a agregação no AGMC é feita através da compilação dos códigos fontes geradas pelas ferramentas externas, com a biblioteca que implementa o framework do protótipo. A idéia é substituir o framework original da ferramenta por um outro framework que permita maior controle do *backend* que será usado pelos códigos fontes incorporados, bem como para permitir que outras ferramentas diferentes da dMOF possam também agregar seus moflets e usufruir do mesmo *backend* usado.

Uma das principais críticas que se faz à representação de perfis UML através de meta-modelos MOF é a de que estes criam semântica nova ao invés de estender a semântica original do meta-modelo UML. Como foi comentado no início desta subseção, existe também uma descrição em MODL para um subconjunto do meta-modelo UML que é importada pelos demais meta-modelos. Isto permite que extensão desejada seja obtida através de herança entre as várias classes inter-relacionadas, como é o caso, por exemplo, das classes *ProcessComponent*, *Composition*, *Choreography* que herdam semântica de classes do pacote UML14.

6.5.2. Cenário 2 – Criação de Modelos a partir de Perfis UML

Neste cenário, as representações gráficas criadas para o modelo PIM do ambiente de execução de processos de negócios (BPRI), que foram apresentadas no capítulo anterior, são transformadas em instâncias dos artefatos dos meta-modelos MOF correspondentes aos perfis UML usados na representação.

Por exemplo, na Figura 6-16, uma parte do modelo PIM do capítulo anterior (diagrama de comunidade) é usada para mostrar quais os elementos e meta-modelos que estariam relacionados com a representação gráfica de um dos componentes do diagrama (para maior simplicidade, apenas uma das seis portas de protocolo do componente BPRI está sendo considerada).

O processo de instanciação mostrado acima é feito através dos relacionamentos entre dois dos quatro níveis da arquitetura MOF: os níveis M1 e M2. Sendo assim, e considerando-se também o que já foi exposto na subseção 6.4.1, cada um dos elementos criados através desta instanciação de artefatos surgiu em

conseqüência do uso das interfaces resultantes dos mapeamentos tecnológicos próprios da especificação MOF. Estas interfaces, via de regra, são usadas por aplicações de modelagem que oferecem um ambiente de edição gráfico amigável para a criação de diagramas UML de diversos tipos (inclusive os que são próprios da notação usada na especificação EDOC) e que mapeiam as várias ações de seus ambientes gráficos em comandos próprios das interfaces resultantes do mapeamento tecnológico.

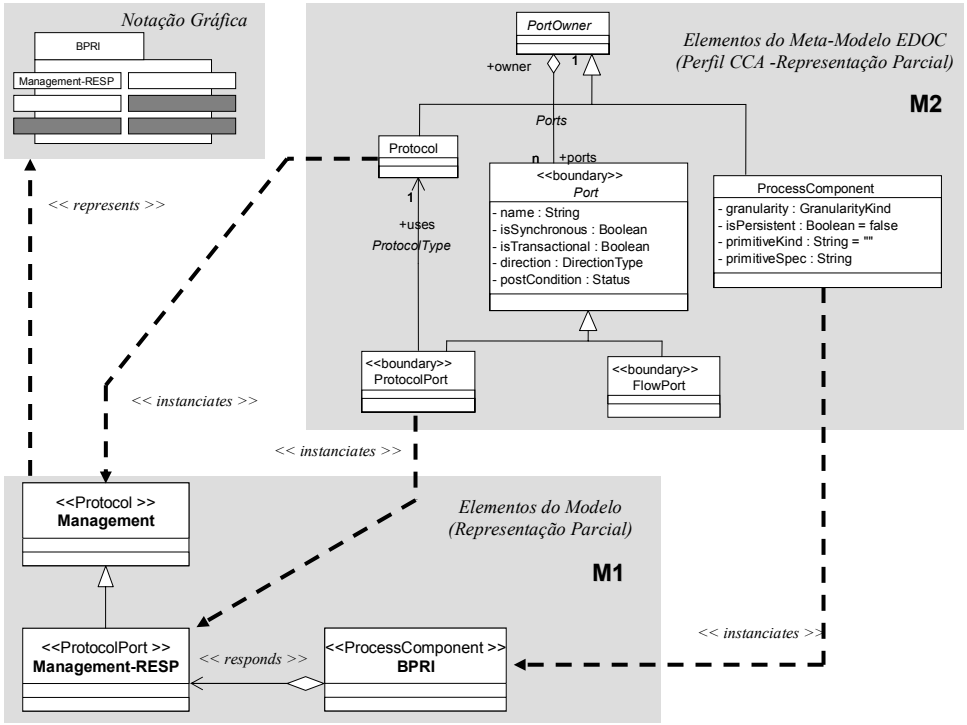

Figura 6-16 - Criação de elementos do PIM a partir de meta-modelo

Listagem 6-2 - Código para criação dos elementos mostrados na Figura 6-16

```
Import java.io.*;
public class ClientExample1 {

 public static void main(String args[])  {
   try {
     org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
     LineNumberReader input = new LineNumberReader(new FileReader("AGMC.ior"));
     org.omg.CORBA.Object object = orb.string_to_object(input.readLine());

     AGMC.AGMCPackageFactory factory = AGMC.AGMCPackageFactoryHelper.narrow(object);
     AGMC._AGMCPackage pkg = factory.create_agmc_package();
     EDOC._EDOCPackage edocPkg = pkg.edoc_ref();
     EDOC.CCA._CCAPackage ccaPkg  = edocPkg.cca_ref();
```

```
    EDOC.CCA.ProtocolClass protocolClass = ccaPkg.protocol_ref();
    EDOC.CCA.ProtocolPortClass protocolPortClass = ccaPkg.protocol_port_ref();
    EDOC.CCA.ProcessComponentClass processComponentClass = ccaPkg.process_component_ref();
    EDOC.CCA.RespondingRoleClass respondingRoleClass = ccaPkg.responding_role_ref();

    EDOC.CCA.Protocol ManagementProtocol = protocolClass.create_protocol("Gerência");
    EDOC.CCA.RespondingRole  respRole = respondingRoleClass.create_respondingRole("BPRIt");
    EDOC.CCA.DirectionType dtype0 =  EDOC.CCA.DirectionType.initiates;
    EDOC.CCA.DirectionType dtype1 =  EDOC.CCA.DirectionType.respondes;
    EDOC.CCA.Status stype = EDOC.CCA.Status.successful;

    EDOC.CCA.ProtocolPort  protocolPortSet[] = new EDOC.CCA.ProtocolPort[6];
    EDOC.CCA.ProtocolPort   ManagementProtocolPortResp =
          protocolPortClass.create_protocol_port(("Gerência – RESP", true, false, dtype1, stype);
    protocolPortSet[0] = ManagementProtocolPortResp;

    EDOC.CCA.GranularityType gtype0 =  EDOC.CCA.DirectionType.program;
    EDOC.CCA.GranularityType gtype1 =  EDOC.CCA.DirectionType.owned;
    EDOC.CCA.GranularityType gtype2 =  EDOC.CCA.DirectionType.shared;
    EDOC.CCA.ProcessComponent processComponentBPRI =
          processComponentBPRI.create_process_component("BPRI", gtype0,true,"","");

    ManagementProtocolPortResp.set_usesProt(ManagementProtocol);
    processComponentBPRI.set_ports(protocolPortSet);
    ManagementProtocol.set_responder(respRole);

   } catch (org.omg.CORBA.UserException e){
    e.printStackTrace();
    System.err.println("Caught an unexpected MOF exception: " + e);

   } catch (org.omg.CORBA.SystemException e){
    e.printStackTrace();
    System.err.println("Caught an unexpected CORBA system exception: " + e);
   } catch (Exception e) {
    System.out.println("Exception : " + e) ;
    e.printStackTrace(System.out);

   } // endtry

  } // endmethod

} // endclass
```

A Listagem 6-1 mostra um exemplo das linhas de códigos que uma aplicação teria que executar, para poder refletir no AGMC, as ações gráficas necessárias correspondentes a criação dos elementos do PIM mostrados, a partir de um ambiente gráfico implementado nesta aplicação. Para um melhor entendimento, a listagem pode ser subdividida em três partes: a inicialização, o processo de criação

de instâncias dos artefatos do meta-modelo MOF associado e a criação de instâncias de relacionamentos entre eles.

Listagem 6-3 - Procedimentos de inicialização

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
LineNumberReader input = new LineNumberReader(new FileReader("AGMC.ior"));
org.omg.CORBA.Object object = orb.string_to_object(input.readLine());
AGMC.AGMCPackageFactory factory = AGMC.AGMCPackageFactoryHelper.narrow(object);
AGMC._AGMCPackage pkg = factory.create_agmc_package();
EDOC._EDOCPackage edocPkg = pkg.edoc_ref();
EDOC.CCA._CCAPackage ccaPkg  = edocPkg.cca_ref();
EDOC.CCA.ProtocolClass protocolClass = ccaPkg.protocol_ref();
EDOC.CCA.ProtocolPortClass protocolPortClass = ccaPkg.protocol_port_ref();
EDOC.CCA.ProcessComponentClass processComponentClass = ccaPkg.process_component_ref();
EDOC.CCA.RespondingRoleClass respondingRoleClass = ccaPkg.responding_role_ref();
```

A Listagem 6-3 mostra o processo de inicialização necessário para acessar um conjunto de definições MOF disponibilizados no AGMC. Estes procedimentos só precisam ser executados no início da conexão entre a aplicação cliente e o protótipo. O primeiro passo é obter a referência que representa o AGMC (AGMC.ior), pois ela dá acesso às referências das várias fábricas de objetos que são necessárias para criar as instâncias desejadas. Na Listagem 6-3, apenas o pacote correspondente ao perfil CCA está sendo mostrado (ele faz parte do pacote EDOC), mas vários outros estão disponíveis, como por exemplo, um pacote para o perfil de entidades, para o perfil de eventos, para o perfil de processos de negócios, para um perfil que corresponde a um PDM CORBA e um pacote para definições UML, como será explicado mais adiante.

Listagem 6-4 - Criação de instâncias de artefatos

```
EDOC.CCA.Protocol managementProtocol = protocolClass.create_protocol("Management");
EDOC.CCA.RespondingRole  respRole = respondingRoleClass.create_respondingRole("BPRIt");
EDOC.CCA.DirectionType dtype0 =  EDOC.CCA.DirectionType.initiates;
EDOC.CCA.DirectionType dtype1 =  EDOC.CCA.DirectionType.respondes;
EDOC.CCA.Status stype = EDOC.CCA.Status.successful;
EDOC.CCA.ProtocolPort  protocolPortSet[] = new EDOC.CCA.ProtocolPort[6];
EDOC.CCA.ProtocolPort   ManagementProtocolPortResp =
      protocolPortClass.create_protocol_port(("Gerência – RESP", true, false, dtype1, stype);
protocolPortSet[0] = ManagementProtocolPortResp;
EDOC.CCA.GranularityType gtype0 =  EDOC.CCA.DirectionType.program;
EDOC.CCA.GranularityType gtype1 =  EDOC.CCA.DirectionType.owned;
EDOC.CCA.GranularityType gtype2 =  EDOC.CCA.DirectionType.shared;
EDOC.CCA.ProcessComponent processComponentBPRI =
      processComponentBPRI.create_process_component("BPRI", gtype0,true,"","");
```

Na Listagem 6-4, as fábricas de objetos são usadas para criar instâncias para cada um dos elementos do diagrama usado no PIM. No nosso caso são os

elementos *Management* (instância de *Protocol*), *Management–RESP* (instância de *ProtocolPort*) e BPRI (instância de *ProcessComponent*). É importante que se note que para criar uma destas instâncias, os valores de todos os seus atributos visíveis externamente devem ser fornecidos no momento da criação. Por exemplo, para criar uma instância do artefato ProcessComponent deve-se fornecer cinco argumentos no método que create_process_component(). São o nome do componente ("BPRI"); a sua granularidade, ou seja, o escopo no qual o componente opera ("Program"); uma indicação se este componente é persistente ou não (true); uma indicação se a semântica da implementação do componente é complementada fora do componente ("" – não complementações externas); e uma indicação de onde é feita esta complementação ("" – não há necessidade de descrever nenhum local).

A Listagem 6-5 mostra os relacionamentos correspondentes a referências indicadas no meta-modelo que contém os artefatos *Protocol*, *ProtocolPort* e *ProcessComponent.* Para ter-se uma idéia destes relacionamentos, considere-se, por exemplo, o caso do artefato *ProtocolPort.* Como todo *ProtocolPort* possui um dono (*owner*), uma instância de *ProtocolPort* deve ser referenciada por uma instância de um artefato *PortOwner (*elemento abstrato). *ProcessComponent* é uma especialização de *PortOwner* e portanto possui uma relação denominada *ports* que é usada para descriminar suas portas. De forma semelhante, como toda *ProtocolPort* faz uso de um protocolo, a instância de *ProtocolPort* referencia a instância de *Protocol* numa relação denominada *uses*. Além disso, uma instância de *Protocol* pode fazer uso de uma instância de *RespondingRole* para representar o *ProcessComponent* que receberá a primeira mensagem do protocolo *Gerência* (no caso o ProcessComponent BPRI).

Listagem 6-5 - Criação de instâncias dos relacionamentos

```
ManagementProtocolPortResp.set_usesProt(ManagementProtocol);
processComponentBPRI.set_ports(protocolPortSet);
ManagementProtocol.set_responder(respRole);
```

6.5.3. Cenário 3 – Criação de PSM para Geração de Código

Quando um modelo PIM atinge o nível de refinamento desejado pelo seu projetista, o passo seguinte é a escolha da plataforma final na qual o modelo será gerado. Esta escolha consiste, basicamente, na definição de qual PDM será utilizado juntamente com o PIM no processo de junção mostrado na Figura 4-3. A seleção do PDM pode ser motivada por diversos fatores, como por exemplo, a compatibilidade com sistemas legados, decisões de projeto ou diretrizes previamente definidas em um grupo de desenvolvimento. No caso deste trabalho, o PDM para a plataforma CORBA [OMG01o] foi escolhido por três razões. A primeira delas está relacionada com o fato do PDM CORBA ser o único a estar totalmente padronizado dentro da OMG até a data de conclusão deste trabalho. É

importante ressaltar que existem outros tipos de PDMs disponíveis (EJB e CCM), mas nenhum deles atingiu o nível de maturidade que se tem, no momento, para o PDM CORBA. A segunda razão se deve a aspectos de compatibilidade com a ferramenta utilizada para gerar as interfaces MOF dos meta-modelos. A ferramenta dMOF foi desenvolvida para o ambiente CORBA e portanto, para que os componentes gerados fizessem uso de um repositório baseado em moflet, estes deveriam também estar na mesma plataforma. Por fim, a terceira razão está relacionada com um dos requisitos que foram colocados na RFP [OMG02d] que mostrava a necessidade de um modelo PIM para a execução de processos de negócios baseados na especificação EDOC. O requisito pedia explicitamente que o modelo PIM BPRI deveria considerar o mapeamento do PIM para o ambiente de execução de workflows definido OMG-WF. Este, por sua vez, é definido para a plataforma CORBA.

O Perfil UML para CORBA especifica como usar a linguagem UML para representar os elementos de uma interface IDL, sem se preocupar em como o comportamento (implementação) deve ser implementado. Sendo assim, o mapeamento de um PIM genérico para que se torne um PSM CORBA deve, essencialmente, procurar entre os elementos do PIM, aqueles elementos que possam ser representados por elementos de uma interface IDL CORBA. Isto implica nas correspondências mostradas na Tabela 6-1:

Tabela 6-1 - Tabela de mapeamento do PIM para PSM (PDM CORBA)

Elemento do PIM	Elemento do PSM
Interfaces usadas por ProtocolPorts	CORBAInterface
FlowPort	CORBAInterface Operations
FlowPort CompositeData	CORBAInterface Attributes
EntityData (Seqüências)	CORBASequence
EntityData (Estruturas)	CORBAStruct
EntityData (Tipos Enumerados)	CORBAEnum
EntityData (Exceções)	CORBAException

A Figura 6-17 mostra um exemplo de modificação no modelo de informação (mostrado anteriormente na Figura 5-9), após o mapeamento que transforma os elementos do modelo PIM em elementos de um modelo PSM correspondente. Este mapeamento pode ser automatizado através de programas que inspecionem o modelo PIM em busca de instâncias dos elementos da primeira coluna da Tabela 6-1 e criem instâncias dos elementos correspondentes aos da coluna dois em outro modelo definido no gerente de repositórios apropriado (AGMC). Uma vez executada esta operação, o modelo PSM pode ser refinado para incluir novos elementos não disponíveis no PIM, como por exemplo, comandos de pré-processamento IDL, comandos de inclusão de pacotes externos ou interfaces do tipo *placeholder*, Os *placeholder* reservam espaço para futuras definições que não

estão disponíveis no momento em que o PSM foi refinado. Por exemplo, no caso da IDL gerada para o PIM proposto no capítulo anterior, não havia como descriminar que todas as interfaces deveriam herdar definições da interface *BaseBusinessObject* que é um *placeholder*, ou seja, ainda não possui métodos nem atributos (a idéia é que venham a ser definidos no futuro). Desta forma um refinamento posterior do PSM teve que ser feito para criar estes elementos. Além disso, nem todos os elementos do PIM, fazem parte do mapeamento especificado para um PSM. Por exemplo, quando o objetivo é obter um PSM correspondente às interfaces de um *Workflow Management Facility* tal qual especificado no OMG-WF, algumas interfaces (as estendidas) não podem fazer parte do arquivo IDL final que deve ser usado e devem ser filtradas no processo de geração do PSM.

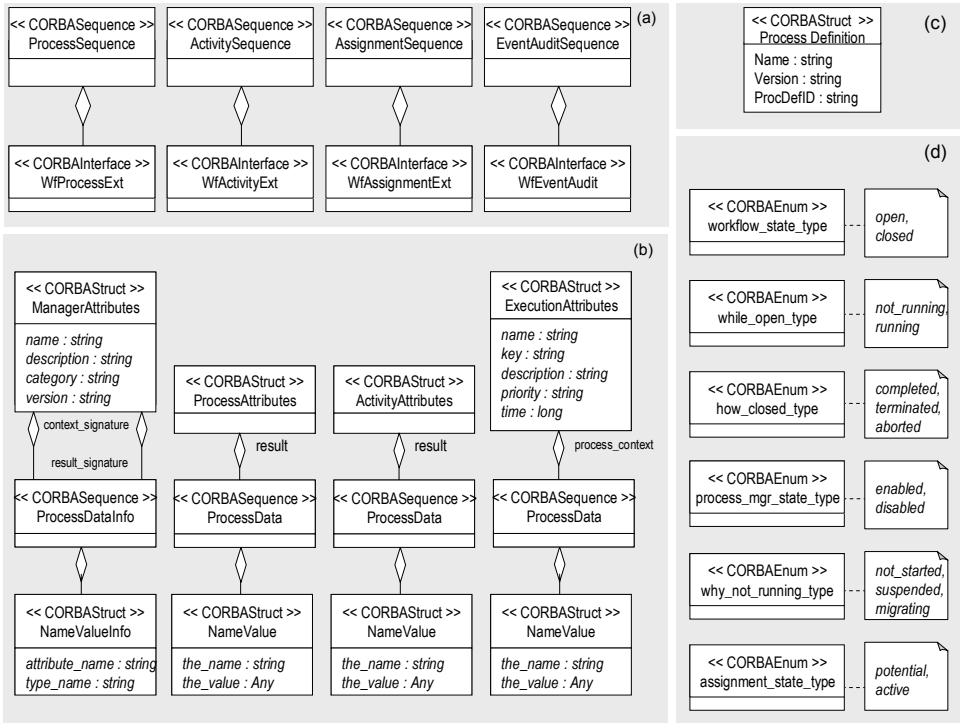

Figura 6-17 - Modelo de Informação depois de mapeamento para PSM.

A geração do arquivo IDL a partir do PSM é um processo direto, uma vez que existe uma correspondência um a um entre os elementos do PSM (que usa a PDM CORBA) e os elementos de uma arquivo IDL.

Tendo sido gerado o arquivo IDL, cada uma das interfaces receberá uma implementação (*servant*) correspondente a seu artefato *Entity* (do PIM) e o conjunto da solução como um todo será inicializado por um código de servidor baseado em POA (Portable Object Adapter), que salvará a referência de seu *servant* principal em uma URL. O *servant* principal é aquele que implementa a interface *WfProcessMgr* e tem equivalência direta com a entidade *ManagerEntity* definida no PIM. Como o projeto do protótipo 2 foi feito em detalhes no capítulo anterior não

há necessidade de repetir para o componente BPRI, os mesmos procedimentos usados para descrever o protótipo 1 (AGMC).

6.5.4. Cenário 4 – Acessando o AGMC a partir do BPRI

O último cenário a ser considerado neste trabalho é o que mostra a interação entre o AGMC e o protótipo de BPRI. Para que esta interação seja possível, é necessário que o repositório (implementado por AGMC e seus moflets) receba uma descrição de definição de processo feita a partir de um dos perfis UML que são usados para descrever processos de negócios (ou uma combinação dos dois). Estes perfis são: o perfil para processos de negócios (EDOC-BP) e o perfil para eventos (EDOC-Events).

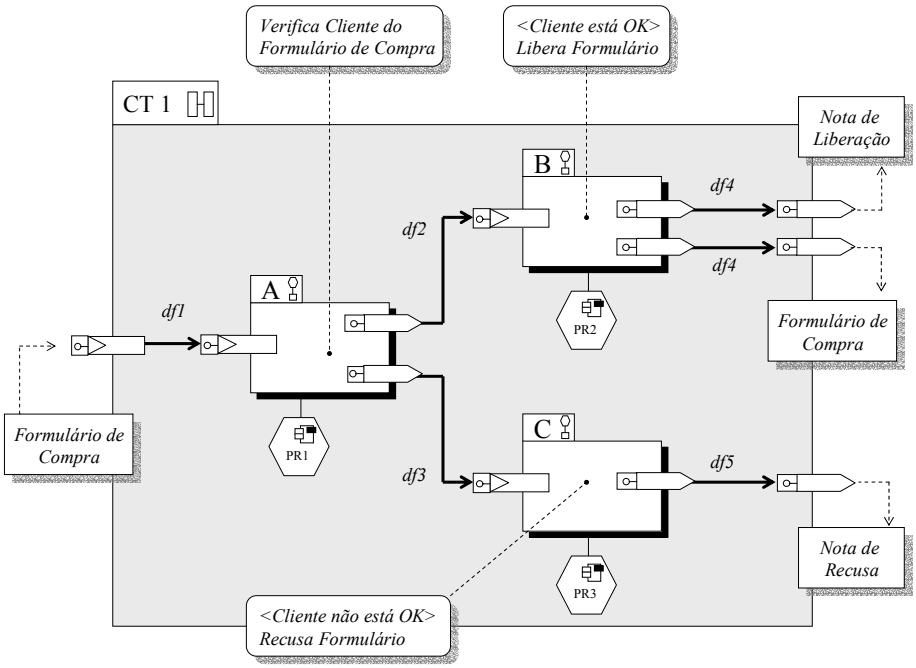

Figura 6-18 - Exemplo de definição de processo.

A Figura 6-18 mostra um exemplo de definição de processo que é necessária para a validação de um usuário que deseja comprar um conjunto de itens discriminados em um formulário de compras. O nome da definição de processo é CT1 e ela é basicamente uma representação de uma composição de atividades que formam este processo de validação do usuário comprador. O formulário de compras deve ser visto como um documento que representa os dados de entrada para a definição de processos e deve incluir as informações que foram fornecidas a partir do preenchimento de um formulário em HTML (*Hypertext Markup Language*),

fornecido via Web. A primeira atividade considerada no processo (Atividade A) é a que verifica os dados de cadastro e o limite de crédito do usuário comprador. Caso o usuário esteja cadastrado e o limite de créditos esteja de acordo com o total do formulário de compra, este formulário é enviado para a atividade B, que é responsável pela emissão da nota de liberação que vai permitir que outras verificações sejam feitas (verificação de disponibilidade no estoque para os itens do formulário, por exemplo). Caso o usuário comprador, tenha algum problema de cadastro, ou o seu limite de seu crédito tenha sido ultrapassado, o formulário de compra é enviado para uma outra atividade que vai emitir uma nota de recusa para o formulário em questão. É importante considerar-se aqui, que o formulário possui um campo de processamento interno em que o resultado da verificação de crédito pode ser incluído. Desta forma, o devido roteamento entre atividades alternativas pode fazer uso de condições (pré-condições ou pós-condições) que fazem referência ao campo de resultado para determinar quais as próximas atividades a serem executadas.

A Figura 6-19 mostra as principais correspondências entre o meta-modelo MOF correspondente ao perfil UML para processos de negócios (Perfil EDOC-BP) e a notação usada para representar uma definição de processo graficamente. Para maior simplicidade, apenas algumas correspondências puderam ser colocadas na figura. O meta-modelo MOF para processos de negócios é mostrado por completo na Figura 3-8.

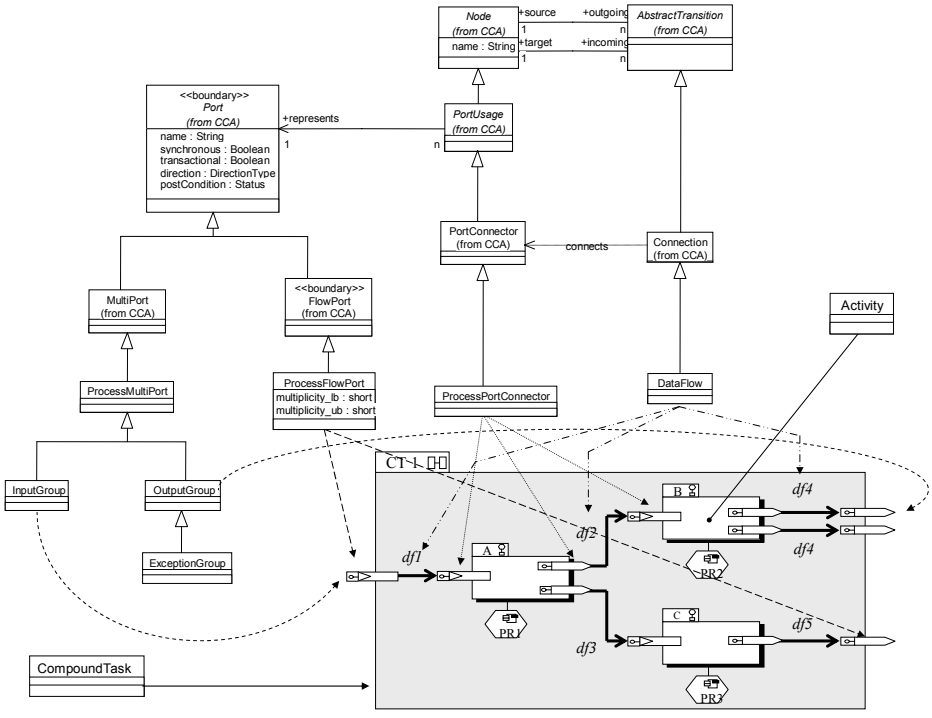

Figura 6-19 - Notação EDOC-BP e meta-modelo MOF correspondente.

O passo seguinte para que a interação entre o protótipo AGMC e o componente BPRI possa ser possível é a formalização da definição de processo especificada anteriormente, na forma de um outro modelo PIM: o modelo PIM para a definição de processo *CT1*. Esta formalização é feita através do programa mostrado na Listagem 6-6.

Listagem 6-6 - Código necessário para o PIM da definição de processo CT1

```
import java.io.*;
public class ClientExample2 {

  public static void main(String args[])  {
    try {
      org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
      LineNumberReader input = new LineNumberReader(new FileReader("AGMC.ior"));
      org.omg.CORBA.Object object = orb.string_to_object(input.readLine());

      AGMC.AGMCPackageFactory factory = AGMC.AGMCPackageFactoryHelper.narrow(object);
      AGMC._AGMCPackage pkg = factory.create_agmc_package();
      EDOC._EDOCPackage edocPkg = pkg.edoc_ref();
      EDOC.CCA._CCAPackage ccaPkg  = edocPkg.cca_ref();
      EDOC.BusinessProcesses._BusinessProcessesPackage bpPkg = pkg.business_processes_ref();

      EDOC.CCA.CompositeDataClass compositeDataClass = ccaPkg.composite_data_ref();
      EDOC.BusinessProcesses.CompoundTaskClass compoundTaskClass = bpPkg.compound_task_ref();
      EDOC.BusinessProcesses.ActivityClass activityClass = bpPkg.activity_ref();
      EDOC.BusinessProcesses.DataFlowClass dataFlowClass = bpPkg.data_flow_ref();
      EDOC.BusinessProcesses.ProcessPortConnectorClass processPortConnectorClass =
            bpPkg.process_port_connector_ref();
      EDOC.BusinessProcesses.ProcessFlowPortClass processFlowPortClass = bpPkg.process_flow_port_ref();
      EDOC.BusinessProcesses.ProcessRoleClass processRoleClass = bpPkg.process_role_ref();
      EDOC.BusinessProcesses.PerformerClass performerClass = bpPkg.performer_ref();
      EDOC.BusinessProcesses.ArtifactClass artifactClass = bpPkg.artifact_ref();
      EDOC.BusinessProcesses.ResponsiblePartyClass responsiblePartyClass = bpPkg.responsible_party_ref();
      EDOC.BusinessProcesses.InputGroupClass inputGroupClass = bpPkg.input_group_ref();
      EDOC.BusinessProcesses.OutputGroupClass outputGroupClass = bpPkg.output_group_ref();

       // Create an instance of CompoundTask ----------------------------------------------------------
      EDOC.BusinessProcesses.CompoundTask ct1 = compoundTaskClass.create_compound_task("shared",
                                        true,  "",   "" );

      // Create an instance of the Activities---------------------------------------------------------
      EDOC.BusinessProcesses.Activity act1 = activityClass.create_activity("Atividade A");
      EDOC.BusinessProcesses.Activity act2 = activityClass.create_activity("Atividade B");
      EDOC.BusinessProcesses.Activity act2 = activityClass.create_activity("Atividade C");
      EDOC.BusinessProcesses.Activity activitySet[] = new EDOC.BusinessProcesses.Activity[3];
      activitySet[0] = act1;
      activitySet[1] = act2;
      activitySet[2] = act3;
      ct1.set_composition_uses(activitySet);

   // Create an instance of the ProcessPortConnector----------------------------------
      System.out.println("Creating ProcessPortConnector");
      EDOC.BusinessProcesses.ProcessPortConnector ppc1 =
```

```
        processPortConnectorClass.create_process_port_connector("ppc1");
 …
 EDOC.BusinessProcesses.ProcessPortConnector ppc12 =
        processPortConnectorClass.create_process_port_connector("ppc12");

// Llink the ProcessPortConnectors to the CompoundTask ct1 --------------------------------
 EDOC.BusinessProcesses.ProcessPortConnector processPortConnectorSet[] =
        new EDOC.BusinessProcesses.ProcessPortConnector[12];
 processPortConnectorSet[0] = ppc1;
 …
 processPortConnectorSet[11] = ppc12;
 ct1.set_nodes(processPortConnectorSet);

// Create an instance of the DataFlow ----------------------------------
 EDOC.BusinessProcesses.DataFlow df1 = dataFlowClass.create_data_flow();
 …
 EDOC.BusinessProcesses.DataFlow df6 = dataFlowClass.create_data_flow();
 EDOC.BusinessProcesses.DataFlow dataFlowSet[] = new EDOC.BusinessProcesses.DataFlow[6];
 dataFlowSet[0] = df1;
 …
 dataFlowSet[5] = df6;
 ct1.set_connections(dataFlowSet);

 // Sets source and target in DataFlow ----------------------------------
  df1.set_source(ppc1);
  df1.set_target(ppc2);
  df2.set_source(ppc3);
  df2.set_target(ppc4);
  df3.set_source(ppc5);
  df3.set_target(ppc6);
  df4.set_source(ppc7);
  df4.set_target(ppc8);
  df5.set_source(ppc9);
  df5.set_target(ppc10);
  df6.set_source(ppc11);
  df6.set_target(ppc12);

// Create an instance of the ProcessFlowPort ----------------------------------
  System.out.println("Creating ProcessFlowPort");
  EDOC.CCA.DirectionType dtype0 =  EDOC.CCA.DirectionType.initiates;
  EDOC.CCA.DirectionType dtype1 =  EDOC.CCA.DirectionType.respondes;
  EDOC.CCA.Status stype = EDOC.CCA.Status.successful;
  EDOC.BusinessProcesses.ProcessFlowPort pfp1 =
        processFlowPortClass.create_process_flow_port("pfp1", true, false, dtype1, stype);
  …
  EDOC.BusinessProcesses.ProcessFlowPort pfp4 =
        processFlowPortClass.create_process_flow_port("pfp4", true, false, dtype0, stype);

  EDOC.BusinessProcesses.ProcessFlowPort   processFlowPortSet[] =
          new EDOC.BusinessProcesses.ProcessFlowPort[6];
  processFlowPortSet[0] = pfp1;
  …
  processFlowPortSet[4] = pfp4;
  ct1.set_ports(processFlowPortSet);

  // Defines links between a connector and a flow ----------------------------------
```

```
    ppc1.set_represents(pfp1);
    ppc8.set_represents(pfp2);
    ppc10.set_represents(pfp3);
    ppc12.set_represents(pfp4);

    // Defines links between an activity and its connectors ----------------------------------
    EDOC.BusinessProcesses.ProcessPortConnector activityPortsSet[] =
          new  EDOC.BusinessProcesses.ProcessPortConnector[3];
    activityPortsSet[0] = ppc2;
    activityPortsSet[1] = ppc3;
    activityPortsSet[2] = ppc5;
    act1.set_ports_used(activityPortsSet);

    activityPortsSet[0] = ppc4;
    activityPortsSet[1] = ppc7;
    activityPortsSet[2] = ppc9;
    act2.set_ports_used(activityPortsSet);

    EDOC.BusinessProcesses.ProcessPortConnector activityPortsSet[] =
          new  EDOC.BusinessProcesses.ProcessPortConnector[2];
    activityPortsSet[0] = ppc6;
    activityPortsSet[1] = ppc11;
    act3.set_ports_used(activityPortsSet);

   // Create instances for the DataGroups ------------------------------------------------------
   EDOC.BusinessProcesses.InputGroup     inGroup1 = inputGroupClass.create_input_group("InG1");
   EDOC.BusinessProcesses.OutputGroup  outGroup1 = outGroupClass.create_output_group("OutG1");
   EDOC.BusinessProcesses.OutputGroup  outGroup2 = outGroupClass.create_output_group("OutG2");
   EDOC.BusinessProcesses.OutputGroup  outGroup3 = outGroupClass.create_output_group("OutG3");

   EDOC.BusinessProcesses.ProcessFlowPort   inGroupFlowPortSet[] =
           new EDOC.BusinessProcesses.ProcessFlowPort[1];
   inGroupFlowPortSet [0] = pfp1;
   inGroup1.set_ports(inGroupFlowPortSet);

   EDOC.BusinessProcesses.ProcessFlowPort   outGroupFlowPortSet[] =
           new EDOC.BusinessProcesses.ProcessFlowPort[1];
   outGroupFlowPortSet [0] = pfp1;
   outGroup1.set_ports(outGroupFlowPortSet);
   outGroupFlowPortSet [0] = pfp2;
   outGroup2.set_ports(outGroupFlowPortSet);
   outGroupFlowPortSet [0] = pfp3;
   outGroup3.set_ports(outGroupFlowPortSet);

    EDOC.CCA.CompositeData formCompras = compositeDataClass.create_composite_data(
          “nome”, orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_string),
          “endereço”, orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_string),
          “CPF”,  orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_string),
          “status”, orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_ushort));
          “itens”, orb.get_primitive_tc(org.omg.CORBA.TCKind.tk_sequence));
    pfp1.set_type(formCompras);
    …

  } catch (org.omg.CORBA.UserException e){
    e.printStackTrace();
    System.err.println("Caught an unexpected MOF exception: " + e);
```

```
   } catch (org.omg.CORBA.SystemException e){
     e.printStackTrace();
     System.err.println("Caught an unexpected CORBA system exception: " + e);

   } catch (Exception e) {
     System.out.println("Exception : " + e) ;
     e.printStackTrace(System.out);

   } // endtry
  } // endmethod
} // endclass
```

A Listagem 6-6, mostra um exemplo de como uma determinada definição de processos deve ser formalizada através dos moflets disponíveis no protótipo AGMC. Em comparação com a listagem usada na formalização do PIM do ambiente de execução (Listagem 6-2), pode-se observar que agora há a necessidade também da utilização do meta-modelo MOF que representa um perfil EDOC-BP, pois para acessar qualquer um de seus artefatos é necessário que se obtenha antes de tudo uma referência para sua fábrica de pacotes. Além disso, para cada um dos artefatos pertencentes ao meta-modelo EDOC-BP, há a necessidade também das fábricas de objetos que compõem o pacote. Todos os artefatos e relacionamentos usados na listagem podem ser observados graficamente na Figura 3-8 do capítulo 3. Com a disponibilização da definição de processo no repositório do AGMC, os componentes BPRI já podem fazer uso do mesmo para atualizarem suas definições de processo, sem a necessidade de novas compilações, sempre que uma delas sofrer alterações. A Figura 6-20 mostra um diagrama de seqüência UML com as principais interações que podem ser feitas neste cenário.

No diagrama de seqüência mostrado, tudo começa com a criação da definição de processo usada pelo componente BPRI. Este processo pode ser feito por qualquer projetista que tenha acesso ao repositório do protótipo AGMC; A seguir o componente BPRI já pode ser inicializado para trabalhar em conjunto com o AGMC. Usando o protocolo de comunicação definido no capítulo anterior, o BPRI se subscreve para obter notificações de atualização para a sua definição de processo (*subscribe*()) e apanha a versão disponível (*get_dp*()); a partir deste momento, qualquer usuário do ambiente de execução de processos de negócios (via Web ou não) pode obter informações sobre o tipo de processo executado pelo ambiente BPRI e iniciar a execução de uma instância do mesmo, ao fornecer as informações exigidas pelo contexto de execução (formulário de compra); se durante a execução de uma instância, a definição de processo disponível no repositório for alterada, o AGMC notificará ao componente subscrito, a respeito da necessidade de uma nova atualização de sua definição de processos (*update*()); Ao receber a notificação o componente apanha a nova versão disponível e tem três alternativas de procedimento para se adequar a situação: abortar as instâncias em execução, aguardar seus términos para criar novas instâncias na nova versão, ou

executar um procedimento de migração das instâncias em execução para a nova versão. Como o tratamento do processo de migração não é um dos requisitos do PIM BPRI, este foi deixado como opcional.

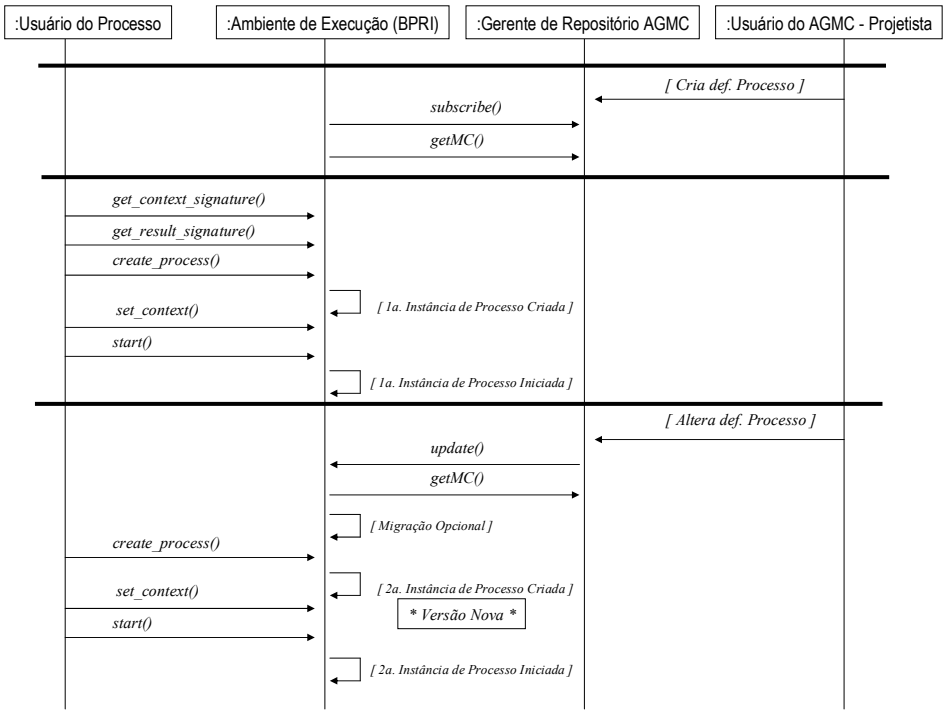

Figura 6-20 - Diagrama de seqüência UML para o cenário 4.

6.5.5. Considerações Finais

Esta subseção apresenta algumas considerações adicionais com relação aos requisitos apresentados no capítulo 4, no sentido de fornecer mais detalhes sobre como o modelo e a implementação feita aqui procurou atender a cada um deles.

a) Criação de um PIM expresso em UML;

A criação do PIM foi feita através dos artefatos disponíveis no framework de modelagem EDOC [OMG02b] procurando considerar não só os aspectos relacionados com a notação do mesmo, mas também a sua representação formal em uma infra-estrutura MDA baseado nas versões 1.4 das especificações UML e MOF.

b) Definição de um mapeamento do PIM para as interfaces propostas na especificação OMG-WF ;

O mapeamento do PIM BPRI para as interfaces disponíveis no OMG-WF é feito da mesma forma descrita na subseção 6.5.3, mas considerando apenas as interfaces próprias da especificação, ou seja, desconsiderando todas as interfaces que estendem as já disponíveis no modelo de OMG-WF (WfProcessMgrExt, WfProcessExt, WfActivityExt e WfAssignmentExt) ou que não estão disponíveis no mesmo (WfRepository e WfProcessDefExt). Além disso, os tipos, ProcessDefinition e assignment_state_type devem ser excluídos do mapeamento por não estarem presentes na especificação anterior.

c) O PIM deve ser estruturado de forma a permitir que diferentes mapeamentos possam estar associados a diferentes graus de acoplamentos, ou seja, o modelo deve ser capaz de se transformar em um componente que opera como um sistema de gerência fortemente acoplado ou como um componente onde há acoplamento fraco entre clientes, componentes e recursos;

O PIM foi feito para permitir mapeamentos específicos para dois tipos de acoplamentos: acoplamento fraco e rígido. Para obter um componente com acoplamento fraco, a utilização de todas as interfaces de extensão deve ser considerada (principalmente a que inclui os métodos de susbcrição a eventos – WfProcessExt); para fazer acoplamento rígido basta que se faça uso das regras de mapeamento propostas para OMG-WF, pois esta especificação foi idealizada para acoplamento rígido.

d) Definição de um modelo de máquina de estados para os processos do sistema modelado no PIM;

O modelo proposto (Figura 5-3) é semelhante ao definido no OMG-WF, mas agrega um novo subestado (migrating) para o caso da implementação considerar algum tipo de migração no momento da atualização das instâncias de processos.

e) Definição de uma interface de solicitações capaz de acessar informações sobre processos disponíveis, seus nomes, descrições e variáveis de contexto que permitam a execução e monitoração dos processos; Operações para suspender ou terminar um processo ou modificar suas variáveis de processo; notificar a conclusão ou terminação de um processo com a apresentação das variáveis relevantes para seu resultado e o acesso aos estados de sub-processos relacionados com um determinado processo invocado;

Qualquer solicitante que implementa a interface WfRequester (Solicitante) terá acesso aos serviços solicitados acima, através das interfaces WfProcessMgrExt, WfProcessExt e WfExecutionObject.

f) Definição de uma interface de acesso ao histórico de eventos de um processo e suas associações;

Qualquer solicitante que implementa a interface WfRequester (Solicitante) terá acesso aos serviços de histórico acima, através da interface correspondente a um objeto em execução: WfExecutionObject.

g) Definição de uma interface para importar definições de processos a partir de repositórios, onde foram modeladas, para dentro do ambiente de execução definido pelo PIM .

Esta interface não estava disponível modelo de interfaces OMG-WF, mas foi incluída através de WfRepository.

h) Definição de um escopo consistente para transações envolvendo processos de negócios. Por exemplo, o escopo de uma transação de um processo em execução não deve entrar em conflito com o escopo de um de seus sub-processos.

Cada processo ou sub-processo está relacionado com seu próprio ambiente de execução e sua própria definição de processos, como cada componente que implementa seu ambiente de execução tem seu escopo definido de forma independente, isto também se torna verdade para as definições de processos executadas pelos mesmos.

i) Definição de tratamento de exceções dentro do modelo do processo ou outras funções e serviços com as quais o componente modelado venha a interagir.

Todas as exceções necessárias para a implementação de tratamento de erro que foram consideradas no OMG-WF foram também modeladas através de artefatos ENTITY_DATA que podem ser mapeados posteriormente em artefatos próprios de uma plataforma específica (ex. CORBAException). Além disso, o modelo pode ser refinado para adicionar novas instâncias correspondentes a novos tipos de exceções.

6.6 Resumo

Este capítulo apresentou os principais aspectos implementacionais relacionados com o trabalho desta tese. Dois protótipos foram desenvolvidos neste sentido, com o objetivo de verificar as idéias discutidas nos capítulos anteriores. Um dos protótipos é um ambiente de gerência de meta-informações (AGMC) e tem como principal função fornecer a infra-estrutura de suporte na qual o PIM do ambiente de execução de processos de negócios pode ser desenvolvido e refinado.

O segundo protótipo é o próprio ambiente de execução que foi criado a partir das interfaces IDL extraídas do PSM correspondente ao PIM.

O processo de validação dos protótipos foi feito com base na utilização dos protótipos em quatro cenários propostos no capítulo 4. Os três primeiros cenários estão intimamente relacionados com a metodologia necessária para a formalização do PIM e para a realização dos mapeamentos necessários para transformá-lo em PSM de uma plataforma de *middleware* específica. O quarto cenário considera as interações entre um componente gerado e o repositório no qual foi gerado. Este cenário é um exemplo clássico de gerência de meta-informações em tempo real e mostra um aspecto pouco explorado nas primeiras provas de conceito que procuram validar os princípios da arquitetura MDA.

Capítulo 7
Conclusões e Trabalhos Futuros

7.1 Considerações Gerais

Embora o principal objetivo da arquitetura MDA seja a interoperabilidade de sistemas de informação, hoje é certo que este não precisa ser o único. À medida que os princípios da arquitetura caminham para a maturidade, gradualmente, novas aplicações, cenários e requisitos se tornam mais evidentes.

A arquitetura MDA tem um papel importante nos processo de geração de código e na integração de sistemas de software, mas ainda não é suficiente, pois são muitas as atividades associadas com o desenvolvimento de sistemas que ainda não são abordadas formalmente pela arquitetura e sua metodologia. São atividades como, por exemplo, a verificação, a validação e os testes de sistemas de software. Além disso, cada um dos domínios de serviços considerados pela OMG para a arquitetura MDA há de contribuir com seus próprios requisitos e cenários, de forma a motivar o surgimento de novos perfis UML ou de complementos destes, baseados em modelos PIM. Esta tese apresentou um destes casos, dentro do domínio em que os sistemas de gerência de processos de negócios estão inseridos.

O trabalho propôs um modelo independente de plataforma (PIM) para um ambiente de execução de processos de negócios, como uma tentativa de se complementar o perfil EDOC-BP, no sentido de evitar ambigüidades ou indefinições que possam levar a problemas de interoperabilidade entre diferentes componentes de processos de negócios, que façam uso de definições de processos EDOC para compor soluções colaborativas complexas, nas quais tenham que interagir entre si. A principal causa destas ambigüidades está relacionada com o fato do perfil EDOC-BP permitir apenas a definição da estrutura de um processo de negócio, sem definir como este processo deve ser executado. Isto pode levar a diferentes interpretações com relação à forma como as definições de processos EDOC-BP devam ser tratadas.

A proposta do modelo PIM para o ambiente de execução, permite que pontos importantes, comumente associados a problemas de interoperabilidade, tais como as interações entre os usuários e o ambiente de execução; entre o ambiente de execução e seus recursos; e entre o ambiente de execução e seu repositório de

modelos pudessem ser explicitadas, formalmente, através da definição dos protocolos que regem estas interações, bem como das principais mensagens e informações trocados entre estes, de uma forma geral e independente de plataforma.

Outro aspecto importante do trabalho que o difere da maioria das propostas de modelos independentes de plataforma, é o fato de que cada aspecto conceitual considerado no projeto do PIM BPRI foi formalmente expresso através da notação gráfica da especificação EDOC, para ser depois concretizado através de um protótipo que implementa uma infra-estrutura MDA baseada nas versões atuais das especificações MOF e UML (versões 1.4). Neste sentido, o modelo foi submetido aos vários cenários relacionados com o ciclo de vida de um modelo da arquitetura MDA, para ser então, analisado e validado em cada um deles.

A seção 7.2 apresenta as principais contribuições deste trabalho, bem como as eventuais limitações e/ou observações pertinentes para cada uma delas. A seguir, na seção 7.3, uma série de propostas de trabalho são apresentadas como possíveis trabalhos futuros para a extensão dos resultados aqui obtidos e a eventual aplicação destes em novos cenários.

7.2 Contribuições da Tese

Em temos gerais, este trabalho contribui para o processo de amadurecimento da arquitetura MDA com mais uma prova de conceito, que verifica e valida um modelo independente de plataforma nos cenários básicos da arquitetura e um cenário adicional, onde os componentes gerados a partir deste modelo fazem uso do próprio ambiente que os gerou para se configurar dinamicamente. Neste sentido, as seguintes contribuições foram feitas:

- Criação de um modelo PIM para componentes que implementam ambientes de execução para processos de negócios EDOC;

- Desenvolvimento de um protótipo de infra-estrutura de suporte para a formalização do modelo PIM criado e a sua verificação ao longo dos cenários propostos no início do trabalho.

- Implementação de um segundo protótipo que incorpora as interfaces geradas para um componente compatível com o modelo proposto e a sua verificação através das interações previstas entre este componente, seus usuários e o repositório onde foi disponibilizada sua definição de processo EDOC-BP.

7.2.1. O Modelo PIM para Processos de Negócios

A proposta do modelo para componentes que implementam ambientes de execução para processos de negócios EDOC, foi fortemente influenciada pela experiência adquirida nos trabalhos publicados em [SIL00], [RIC02], [SIL02a] e [SIL02b].

Em todos estes trabalhos, o principal aspecto em comum foi a utilização de repositórios MOF para armazenar representações de processos de negócio que pudessem ser utilizadas pelos ambientes de execução para se atualizar dinamicamente. No entanto, apenas [SIL02b] considerou definições de processos baseadas na especificação EDOC, e o fazia a partir de um ambiente de execução CORBA, compatível com a especificação OMG-WF. O modelo PIM criado permitiu basicamente, que as idéias aplicadas em [SIL02b] pudessem ser expandidas para outras plataformas, à medida em que novos PDMs venham a ser disponibilizados através de novos perfis UML. Provavelmente, os próximos PDMs a serem padronizados serão os PDMs para EJB (*Enterprise Java Beans*) [SUN03] e para CCM (*CORBA Component Model)* [OMG02m].

Os principais problemas encontrados na criação do modelo PIM estavam relacionados com a complexidade das definições e relacionamentos descritos na especificação EDOC e a pouca disponibilidade de exemplos complexos, como o ambiente de execução considerado. Além disso, a dificuldade de encontrar ferramentas plenamente compatíveis com a especificação MDA levaram a necessidade de criar um protótipo adicional que pudesse dar suporte aos cenários básicos relacionados com o ciclo de vida dos modelos PIMs.

7.2.2. O Protótipo da Infra-estrutura MDA

A utilização dos princípios conceituais da arquitetura MDA está intimamente relacionada com a disponibilidade de ferramentas que permitam sua aplicação prática na engenharia de software. Apesar já existirem algumas implementações disponíveis comercialmente, é senso comum entre os estudiosos da área que ainda não há uma ferramenta MDA que dê suporte a todos os aspectos importantes da arquitetura [BEL02b], tais como: suporte gráfico para o projeto de modelos, suporte às diversas plataformas de middlware, suporte à representação de ações em UML, suporte a modelos CIM e etc. Sendo assim, ao invés de usar uma ferramenta MDA fechada, este trabalho procurou investir na criação de uma infra-estrutura que podesse ser o primeiro passo na direção de uma ferramenta MDA própria. Este primeiro passo foi à criação do protótipo de infra-estrutura de modelagem que oferece um conjunto básico de funcionalidades definidas pelos cenários do trabalho.

O protótipo da infra-estrutura é uma contribuição que procura aproveitar o que já está disponível em termos de especificações e técnicas de modelagem /meta-modelagem para concretizar uma infra-estrutura de suporte para MDA com

uma funcionalidade mínima para os objetivos do trabalho. Entre estas funcionalidades, estão: a capacidade de usar meta-modelos MOF para representar perfis UML (evitando perdas semânticas através da representação de parte do meta-modelo UML como um meta-modelo MOF que é importado por todos os meta-modelos de perfis); a capacidade de criar modelos PIM e PSM a partir dos meta-modelos definidos anteriormente; a capacidade de permitir a criação de mapeamentos entre os PIMs e PSMs correspondentes; e a capacidade de permitir o acesso, em tempo de execução, para os componentes gerados que precisem gerenciar meta-informações ao longo de suas operações normais. No entanto, é importante que fique claro, que ao fazer uso de ferramentas e técnicas compatíveis com as versões atuais das especificações UML e MOF(1.4), algumas inconsistências conceituais entre o meta-meta-modelo MOF e o meta-modelo UML são inevitáveis. Algumas delas têm razões históricas e contextuais, que só serão plenamente tratadas quando estes forem reestruturados para incorporar um melhor alinhamento entre os mesmos. Como exemplos das inconsistências mencionadas pode-se citar: a necessidade de transformar as definições dos perfis através dos mapeamentos do perfil UML para MOF, antes de sua utilização na arquitetura MOF, e a impossibilidade de representar adequadamente todo o meta-modelo UML, pois não há correspondência estrita entre este e o meta-meta-modelo MOF, o que faz com que a maioria das ferramentas MOF tenham que fazer uso de apenas uma parte do meta-modelo. Felizmente, já existem algumas iniciativas na direção das novas versões para as especificações UML e MOF (versões 2.0), que não deverão ter problemas de alinhamento entre si. São os trabalhos relacionados com as RFPs da OMG para UML 2.0 ([OMG00k], [OMG01p], [OMG00j] e [OMG00l]) e para MOF 2.0 ([OMG01q], [OMG01s], [OMG02n], [OMG02o] e [OMG01r]).

7.2.3. O Protótipo do Ambiente de Execução

A implementação de um protótipo a partir das interfaces geradas no AGMC é uma importante forma de verificar a adequação do modelo para casos reais de interoperabilidade. Entre as verificações que foram consideradas aqui estão: a verificação do componente em interações relacionadas com os protocolos externos descritos no capítulo 5, particularmente os relacionados com as interações entre clientes e o componente, e entre o componente e um repositório de modelos usados por este para incorporar sua definição de processo.

7.3 Trabalhos Futuros

Este trabalho representa um primeiro passo na direção de especificações mais completas para componentes de processos de negócios, dentro do contexto da especificação EDOC e da arquitetura MDA. No entanto, muitos outros passos

devem se seguir a este, para que os trabalhos desenvolvidos do domínios dos processos de negócio possam evoluir para o nível de abstração proposto pela arquitetura MDA. Além disso, o protótipo de infra-estrutura de suporte pode ser motivo de outras contribuições que agreguem valor em uma série de aspectos em que este ainda não está adequado. Sendo assim, alguns possíveis aperfeiçoamentos são propostos a seguir, para servirem de base para trabalhos futuros nesta área.

- **Suporte a Migração –** o modelo PIM proposto para o ambiente de execução procurou nortear-se pelos principais requisitos da RFP [OMG02d], agregando, adicionalmente, a capacidade do componente gerado ser capaz de receber notificações para atualizar-se com relação a sua definição de processo. Este processo de atualização, entretanto, só ficará completo, com o suporte à migração de instâncias que executam a versão anterior da definição de processo. Como este recurso não foi considerado na RFP, este recurso foi deixado de fora do modelo proposto na tese. No entanto, há espaço para a definição de estratégias de migração baseadas em perfis UML, que descrevam regras de migração e que possam ser utilizadas, dinamicamente, pelo modelo proposto aqui. Neste sentido, a OMG tem trabalhado para padronizar perfis UML capazes de representar diversos tipos de sistemas de regras [OMG02l] .

- **Serviços de Interface –** um passo que se segue naturalmente ao desenvolvimento de uma infra-estrutura é a criação de serviços que se beneficiem das vantagens oferecidas por esta infra-estrutura para oferecerem algum tipo de diferencial em suas prestações de serviços. Nesta direção, há muitas oportunidades de novos trabalhos envolvendo o protótipo AGMC, para o desenvolvimento de serviços de edição gráfica de perfis UML e modelos; serviços de geração de moflets em diversas plataformas; e validadores de meta-modelos e modelos que busquem por inconsistências antes de iniciarem o processo de geração de código ou transformações.

- **As Transformações –** outra direção extremamente rica em novos trabalhos é a que busca por linguagens de representação de transformações e mecanismos que permitam suas aplicações no contexto de uma infra-estrutura, como a proposta no AGMC. A idéia é fazer uso destas linguagens e mecanismos para viabilizar mapeamentos automatizados dentro da arquitetura MDA. Neste sentido, vários trabalhos já estão em andamento e algumas propostas como o meta-modelo de transformações do CWM já apontam algumas tendências na direção de uma especificação formal.

- **Geração de Código –** a aplicação de MDA, no cenário de geração de código a partir de modelos, é um dos cenários mais considerados nas provas de conceito estudadas e está intimamente relacionado com a questão da

automatização das transformações, uma vez que é um caso específico de transformação. No entanto, ele tem uma peculiaridade que o faz ser tratado de forma diferente. A geração depende não só de um modelo dependente de plataforma, mas também dos modelos que descrevem as plataformas propriamente ditas (PDMs). Apesar de existirem várias representações proprietárias para as plataformas, apenas a plataforma CORBA tem uma especificação formal de um perfil UML. No entanto, isto deve mudar até o final de 2003, o que abre espaço para a especificação de novos PSMs para o modelo PIM proposto.

- **Novos Cenários** – nesta tese, apresentamos quatro cenários para o ciclo de vida de um modelo MDA. Embora sejam os mais utilizados, principalmente os três primeiros, muitos ainda devem ser considerados antes da arquitetura MDA atingir sua maturidade. São exemplos de novos cenários os processos de validação, verificação e teste de sistemas de software, pois todos estão associados ao desenvolvimento de software no qual se insere a proposta desta tese.

- **Geração Própria de Moflets –** considerando que o AGMC, em sua versão inicial, fez uso extensivo de código fonte de moflets gerado por outras ferramentas, particularmente pela ferramenta dMOF, seria interessante a criação de um serviço no AGMC que fizesse uso direto do repositório de definições internas para gerar código fonte para seus próprios moflets. Isto daria maior independência para arquitetura sem perda de interoperabilidade, uma vez que esta foi feita para interoperar com moflets gerados por outras ferramentas externas.

8
Referências

[BEL02a] BELAUNDE, M. PELTIER, M. "From EDOC Components to CCM components : a precise mapping specification", Fundamental Approaches to Software Engineering, 5th International Conference, FASE 2002, held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings. Lecture Notes in Computer Science 2306 Springer, pg. 143-158, 2002.

[BEL02b] BELAUNDE, M. Assessment of the Model Driven Technologies – Foundation and Key Technologies. Eurescom IST MODA-TEL Consortium, Deliverable number 2.1, www.modatel.org/~Modatel/pub/ deliverables/D2.1-final.pdf, 2002.

[BEZ01] BEZIVIN, J.; PLOQUIN, N. Tooling the MDA framework: a new software maintenance and evolution scheme proposal. Workshop on Engineering Complex Object-Oriented Systems for e-Commerce ECOOP 2001, June 19th, 2001

[BLA98] BLAIR, G. S., G. COULSON, P. ROBIN AND M. PAPATHOMAS. An Architecture for Next Generation Middleware.Proc. IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware' 98) , pp.191-206, Lake District, UK, Springer, September 1998.

[CAZ98] CAZZOLA, W. Evaluation of Object-Oriented Reflective Models, In: proceedings of ECOOP Workshop on Reflective Object-Oriented Programming and Systems (EWROOPS'98), in 12th European Conference on Object-Oriented Programming (ECOOP'98), Brussels, Belgium, July, Extended Abstract also published on ECOOP'98 Workshop Readers, S. Demeyer and J. Bosch editors, LNCS 1543, ISBN 3-540-65460-7pages 386-387.

[COS01] COSTA, F. M. Combining Meta-Information Management and Reflection in an Architecture for Configurable and Reconfigurable Middleware, PhD thesis, Computing Department, Lancaster University, 2001.

[CRA00] CRAWLEY, S. A Strawman Proposal for a MOF 1.4 Repository Framework. Proposal to MOF 1.4 RTF, Distributed Systems Technology Centre (DSTC), Brisbane, Australia, September 2000.

[CRA97] CRAWLEY, S.; DAVIS, S.; INDULSKA, J.; MCBRIDE, S; RAYMOND, K. Meta – Information Management. In: Procedings of the 2nd IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS' 97) , Canterbury, United Kingdom, July 1997a.

[DEN00] DENNIS, A.; KING, D. Internet and the Digital Economy. Proceedings of the 33rd Hawaii International Conference on System Sciences, Hawaii – USA, 2000.

[DST00] DSTC. dMOF tool User's Guide, release 1.1, Distributed Systems TC, Brisbane, Australia, 2000.

[DUB02] DUBRAY, J.-J. A Novel Approach for Modeling Business Process Definitions. EIGNER, Waltham, MA, 2002.

[FER99] FERREIRA, A. B. H. - Novo Dicionário Aurélio – Século XXI, 1ª. Ed., Editora Nova Fronteira, 1999.

[FRA02] FRANKEL, D. Using Model-Driven Architecture to Manage Metadata, White paper, IONA Technologies, june, 2002.

[GEI01] GEIBS, K. Middleware Challenges Ahead. IEEE Computer Magazine, pg. 24-30, June 2001.

[GEO95] D. GEORGAKOPOULUS, M. HORNICK, A. SHETH, "An Overview of Workflows Management: From Process Modeling to Workflow Automation Infrastructure", Distributed and Parallel Databases, Kluewr Academic Publishers, No. 3, 1995.

[GIO01] GIORDANI, B.; MENDES, M.J. A Supply Chain Management Framework using the TINA-C Business Model and a jFlow Workflow Prototype, Workflow Handbook 2001, Published in association with the Workflow Management Coalition (WfMC) , Edited by Layna Fischer, 2001.

[GIO99] GIORDANI, B.T. Protótipo de um Sistema de Gerência de Workflows baseado em Eventos. Tese de mestrado, Orientação Prof. Manuel de Jesus Mendes, Depto. Eng. De Computação e Automação Industrial, Universidade Estadual de Campinas, maio de 1999.

[GUD96] GUDWIN, R.R. Introdução à Semiótica Computacional. XVII Seminário Nacional dos Estudantes de Engenharia - Goiânia-GO, 21-25/Julho, 1996.

[HAA02] Haas, L. M.; Lin, E. T. and Roth, M. A.. Data integration through database federation, IBM System Journal, vol. 41, number 4, pp. 578-596,, 2002.

[HUR95] HURSCH, W.; LOPES L. & C. V. Separation of Concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, MA USA, February 1995.

[IAN98] IANNELLA, R. Mostly Metadata: A Bit Smarter Technology. Victorian Association for Library Automation (VALA) 1998 Biennial Conference (Keynote Address), Melbourne, 28-30 January 1998.

[INP00] INPRISE. Visibroker Programmer's Guide, version 4.0, Inprise Corporation, 100 Enterprise Way, Scotts Valley, CA 95066-3249, 2000.

[ISO01] ISO, Overview of the MPEG-7 Standard (version 5.0, Committee Draft N4031). ISO/IEC JTC1/SC29/WG11, 2001, http://www.cselt.it/mpeg/standards/mpeg-7/mpeg-7.htm.

[ITU00] ITU-T/ISO. ITU-T Draft Rec. X.960 | ISO/IEC FDIS 14769 – Information Technology - Open Distributed Processing - Type Repository Function, 2000.

[ITU95a] ITU-T/ISO. ITU-T X.901 | ISO/IEC 10746-1 ODP Reference Model Part 1: Overview, 1995.

[ITU95b] ITU-T/ISO (1995b). ITU-T X.903 | ISO/IEC 10746-3 Open Distributed Processing Reference Model Part 3: Architecture.

[ITU95c] ITU-T/ISO (1995d). Reference Model for Open Distributed Processing, Parts 1,2,3. ITU-T X.901-X.904 | ISO/IEC IS 10746-(1,2,3).

[ITU96] ITU-T/ISO (1996). ITU-T X.902 | ISO/IEC 10746-2 Open Distributed Processing - Reference Model Part2: Foundations.

[ITU97] ITU-T/ISO. ITU/T Draft Rec X950 - 1 | ISO/IEC IS 13235-1, ODP Trading Function Specification, 1997.

[ITU98] ITU-T/ISO. ITU-T X.930 | ISO/IEC 14753 Open Distributed Processing - Interface References and Binding, 1998.

[IYE01] IYENGAR, S. Model Driven Architecture (MDA): Motivations, Status & Future. 5th IEEE International Enterprise Distributed Object Computing Conference EDOC 2001, Seattle, Washington USA, September 4-7, 2001

[IYE02] IYENGAR, S. OMG Model Driven Architectures—Modeling, Metadata, Middleware and Mappings for Enterprise Integration. W6: Aligning Data Management Skills with Business Goals, April 28 – May 2, 2002 – San Antonio Convention Antonio, Texas, USA, 2002

[IYE98] IYENGAR, S. A Universal Repository Architecture using the OMG UML and MOF. In: Proc. 2nd Enterprise Distributed Object Computing Conference (EDOC' 98) , pp.35-44, La Jolla, CA USA, IEEE Computer Society Press, November 1998.

[JAB96] JABLONSKI, S.; BUSSLER, C. Workflow Management, Modeling Concepts, Architecture and Implementation. International Thomson Computer Press (ITP), pg. 101-192, 1a. edition, 1996.

[JAC94] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, 1994.

[JOH88] JOHNSON, R. AND FOOTE, B. Designing Reusable Classes. Journal of Object-Oriented Programming, Volume 1, Number 2, June/July 1988.

[KAZ99] KAZMAN, R.; MÜLLER, H.; CARRIÈRE, J. The Role of Architecture in Understanding Software. In: Proceedings 7th International Workshop on Program Comprehension, Pittsburgh, Pennsylvania, Usa, May 05 - 07, 1999.

[KIC91] KICZALES, G.; RIVIERES, J. AND D. G. BOBROW, D. G. The Art of the Metaobject Protocol. MIT Presss, 1991.

[KIC92] KICZALES, G. Towards a new model of abstraction in the engineering of software. IMSA'92 Proceedings, 1992.

[KOB01a] Kobryn, C., Introduction to UML: Structural and Use Case Modeling, Object Modeling with OMG UML Tutorial Series, OMG document omg/2001-03-02, 2001.

[KOB01b] Kobryn, C., Object Modeling with UML: Behavioral Modeling, Object Modeling with OMG UML Tutorial Series, OMG document omg/2001-03-03, 2001.

[KOB01c] Kobryn, C., Advanced Modeling with UML, Object Modeling with OMG UML Tutorial Series, OMG document omg/2001-03-04, 2001.

[MAE87] MAES, P. Concepts and Experiments in Computational Reflection. In: Proceedings. ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA' 87) , pp.147-155, Orlando, FL USA, October 1987.

[MAE88] MAES, P. Issues in Computational Reflection. In Meta-Level Architectures and Reflection, P. Maes and D. Nardi (eds.), pp.21-35, North-Holland, Amsterdan, The Netherlands, 1988.

[MAL96] MALENFANT, J.; JACQUES, M.; DEMERS, F. A Tutorial on Behavioural Reflection and its Implementation. Proc. of Reflection' 96 , pp.1-20, San Francisco, CA USA, April 1996.

[MEJ02] MEJIA, J.A.S. Um Meta-modelo Unificado para Processos de Workflow. Tese de doutorado, Orientação Prof. Manuel de Jesus Mendes, Depto. Eng. De Computação e Automação, Universidade Estadual de Campinas, junho de 2002.

[OKA95] Okamura, H. A Study on Multi-Model Reflection Framework in Distributed Environments. PhD thesis, Department of Computer Science, Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223, JAPAN, February 1995.

[OLI98] OLIVA, A.; GARCIA, I. C.; BUZATO, L. E. The Reflective Architecture of Guaraná. Technical Report, Institute for Computer Science, UNICAMP, Campinas, SP Brazil, September 1998.

[OMG00a] OMG. Workflow Management Specification, version 1.2, OMG doc, URL http://www.omg.org/cgi-bin/doc?formal/00-05-02, 2000.

[OMG00b] OMG. Unified Modeling Language Specification, version 1.3. Object Management Group, Needham, MA USA. OMG doc, URL http://www.omg.org/cgi-bin/doc?formal/00-03-01, 2000.

[OMG00c] OMG. Common Warehouse Metamodel (CWM) Specification (OMG Document ad/2000-01-01). Object Management Group, Needham, MA USA, 2000.

[OMG00d] OMG. Meta Object Facility (MOF) Specification, version 1.3 (OMG Document formal/2000-04-03). Object Management Group, Needham, MA, USA.

[OMG00e] OMG. XML Metadata Interchange (XMI) Specification, version 1.1 (OMG Document formal/2000-11-02). Object Management Group, Needham, MA USA.

[OMG00f] OMG. Model Driven Architecture. by Richard Soley and the OMG Staff Strategy Group, Document: omg/2000-11-05, White paper, Draft 3.2, Needham, MA, USA – November 27, 2000

[OMG00g] OMG. UML Profiles – OMG Document : lifesci/00-06-02:, June, 2002, Object Management Group, Needham, MA USA.

[OMG00h] OMG. UML 2.0 Infrastructure RFP – OMG Document : ad/00-09-01:, September, 2000, Object Management Group, Needham, MA USA.

[OMG00i] OMG. UML 2.0 Superstructure RFP – OMG Document : ad/00-08-09:, September, 2000, Object Management Group, Needham, MA USA.

[OMG00j] OMG. Request for Proposal: UML 2.0 Infrastructure RFP – OMG Document : ad/00-09-01, June 18, 2001, Object Management Group, Needham, MA USA.

[OMG00k] OMG. Request for Proposal: UML 2.0 Superstructure RFP – OMG Document : ad/00-08-09, August 25, 2001, Object Management Group, Needham, MA USA.

[OMG00l] OMG. Request for Proposal: UML 2.0 OCL RFP– OMG Document : ad/00-09-03, June 18, 2001, Object Management Group, Needham, MA USA.

[OMG00m] OMG. Organization Structure Facility – OMG Document : dtc/01-09-04, October, 2001, Object Management Group, Needham, MA USA.

[OMG01a] OMG. The Joint UML for EDOC Submission (Part 1) – OMG Document: ad/2001-06-09, Needham, MA, USA.

[OMG01b] OMG. The Joint UML for EDOC Submission - Supporting Annexes (Part 2) – OMG Document: ad/2001-06-10, Needham, MA, USA.

[OMG01c] OMG. Partial Evaluation of the Joint EDOC Submission – OMG Document : ad/2001-08-23, August 20, 2001, Needham, MA, USA

[OMG01d] OMG. Partial Evaluation of the Joint EDOC Submission – OMG Document : ad/2001-08-32, August 27, 2001 Needham, MA, USA

[OMG01e] OMG. Evaluation of the Joint EDOC Submission – OMG Document : ad/2001-09-01, Needham, MA, USA

[OMG01f] OMG. Unified Modeling Language, v1.4 – OMG Document: formal/01-09-67, Needham, MA, USA

[OMG01g] OMG. Evaluation of the Joint EDOC Submission – OMG Document: ad/2001-09-03, Needham, MA, USA

[OMG01h] OMG. UML Profile for Enterprise Distributed Object Computing Draft Adopted Specification– OMG Document: ptc/2001-12-04, Needham, MA, USA

[OMG01i] OMG. Team response to EDOC Evaluation (August 23, 2001). OMG Document: ad/2001-08-31, Needham, MA, USA

[OMG01j] OMG. Team response to EDOC Evaluation (August 27, 2001). OMG Document: ad/2001-08-33, Needham, MA, USA

[OMG01k] OMG. Model Driven Architecture, A Technical Perspective. OMG Document: ab/2001-02-05, Needham, MA, USA

[OMG01l] OMG. Final MOF 1.4 RTF Issues and Resolutions. OMG Document: ptc/2001-08-23, Needham, MA, USA, 2001.

[OMG01m] OMG. MOF 2.0 Roadmap – OMG Document : ad/01-11-06:, November, 2001, Object Management Group, Needham, MA USA.

[OMG01n] OMG. Object Constraint Language Specification – OMG Document : formal/01-09-77:, September, 2001, Object Management Group, Needham, MA USA.

[OMG01o] OMG. UML Profile for CORBA Specification – OMG Document : ptc/01-01-07:, January, 2001, Object Management Group, Needham, MA USA.

[OMG01p] OMG. Request for Proposal: UML 2.0 Diagram Interchange RFP – OMG Document : ad/01-02-39, October 22, 2001, Object Management Group, Needham, MA USA.

[OMG01q] OMG. Request for Proposal: MOF 2.0 Core RFP – OMG Document : ad/01-11-14, November 13, 2001, Object Management Group, Needham, MA USA.

[OMG01r] OMG. Request for Proposal: MOF 2.0 XMI Mapping RFP– OMG Document : ad/01-11-13, November 14, 2001, Object Management Group, Needham, MA USA.

[OMG01s] OMG. Request for Proposal: MOF 2.0 to OMG IDL Mapping RFP – OMG Document : ad/01-11-07, November 14, 2001, Object Management Group, Needham, MA USA.

[OMG01t] OMG. Workflow Resource Assignment Interfaces (RAI) Request for Proposal – OMG Document : bom/00-01-03, August, 2000, Object Management Group, Needham, MA USA.

[OMG01u] OMG. UML Profile for Enterprise Distributed Object Computing – Annexes – OMG Document: ad/2001-08-20, Needham, MA, USA

[OMG02a] OMG. Meta Object Facility Specification, ver. 1.4, OMG doc, URL http://www.omg.org/cgi-bin/doc?formal/02-04-03, 2002.

[OMG02b] OMG. UML Profile for Enterprise Distributed Object Computing Specification – OMG Document: ptc/2002-02-05, Needham, MA, USA

[OMG02c] OMG. UML Extensions for Business Process Definition, Request for Proposal – OMG Document: bei/2002-03-01, Needham, MA, USA

[OMG02d] OMG. Business Process Runtime Interfaces (BPRI) Platform Independent Model (PIM) Request for Proposal – OMG Document: bei/2002-06-08, Needham, MA, USA.

[OMG02e] OMG. XML Metadata Interchange (XMI) Specification, version 1.2 (OMG Document formal/2002-01-01). Object Management Group, Needham, MA USA.

[OMG02f] OMG. UML TM Profile and Interchange Models for Enterprise Application Integration (EAI) Specification, adopted standard, URL http://www.omg.org/cgi-bin/doc?ptc/02-02-02, 2002.

[OMG02g] OMG. Business Process Management Facility (BPMF) Platform Independent Model (PIM) RFP (Draft 3), adopted standard, URL http://www.omg.org/cgi-bin/doc?cem/02-01-02, 2002.

[OMG02h] OMG. UML Extensions for Business Process Definition RFP, adopted standard, URL http://www.omg.org/cgi-bin/doc?bei/02-03-02, 2002.

[OMG02i] Model Driven Architecture (MDA), OMG Document: ormsc/2002-01-01, Needham, MA, USA, 2002.

[OMG02j] OMG. UML Software Process Engineering Metamodel Specification, final adopted standard, URL http://www.omg.org/cgi-bin/doc?ptc/02-01-23, 2002.

[OMG02k] OMG. UML TM Profile for Schedulability, Performance, and Time Specification, final adopted standard, URL http://www.omg.org/cgi-bin/doc?ptc/02-09-06, 2002.

[OMG02l] OMG. Business Rules with MDA – OMG Document : ad/02-10-05:, October, 2002, Object Management Group, Needham, MA USA.

[OMG02m] OMG. UML Profile for CORBA Components – OMG Document : telecom/02-09-02, September, 2002, Object Management Group, Needham, MA USA.

[OMG02n] OMG. Request for Proposal: MOF 2.0 Versioning and Development Lifecycle RFP – OMG Document : ad/02-06-23, June 25, 2002, Object Management Group, Needham, MA USA.

[OMG02o] OMG. Request for Proposal: MOF 2.0 Query / Views / Transformations RFP – OMG Document : ad/02-04-10, October 28, 2002, Object Management Group, Needham, MA USA.

[OMG03a] OMG. Response to the OMG UML 2.0 Infrastructure RFP, Document Number ad/2003-01-10, Version 0.51 January 6th, 2003.

[OMG03b] OMG. Business Process Runtime Interfaces Platform-Independent Model Specification – Preliminary Initial Submission, Document Number bei/2003-03-01, Version 0.3 march 3rd, 2003.

[OMG03c] OMG. Initial Submission to Business Process Runtime Interfaces RFP, Document Number bei/2003-03-02, Version 0.3 march 3rd, 2003 .

[OMG03d] OMG. MDA Guide – Version 1.0.1. OMG Document : omg/03-06-01, june, 12th 2003, Object Management Group, Needham, MA USA.

[OMG96] OMG. OMG Object Trader Service Specification. orbos/96-05-06, Object Management Group, Needham, MA USA, 1996.

[OMG97a] OMG. jFlow Initial Submission to the Workflow RFP, OMG doc, URL http://www.omg.org/cgi-bin/doc?bom/97-08-05, 1997.

[OMG97b] OMG. EDS Initial Submission to the Workflow RFP, OMG doc, URL http://www.omg.org/cgi-bin/doc?bom/97-08-065, 1997.

[OMG98a] OMG. Draft of the Revised joint Workflow Management Facility submission, OMG doc, URL http://www.omg.org/cgi-bin/doc?bom/98-02-01, 1998.

[OMG98b] OMG. EDOC Presentation. OMG Document URL: http://www.omg.org/cgi-bin/doc?bom/98-11-02, 1998.

[OMG98c] OMG. CORBAservices: Common Object Services Specification. OMG Document URL: http://www.omg.org/cgi-bin/doc?formal/98-12-09, 1998.

[OMG99a] OMG. Components FTF Edited Drafts of CORBA Core Chapters (CORBA 3.0) (Document Number ptc/99-10-03). Object Management Group, Needham, MA USA, 1999.

[OMG99b] OMG. CORBA Components Model - FTF drafts of MOF chapter (CORBA Components ptc/99-10-05, orbos/99-07-02). Object Management Group, Needham, MA USA, 1999.

[OMG99c] OMG. White Paper on the Profile mechanism, Version 1.0, OMG Document ad/99-04-07, Object Management Group, Needham, MA USA, 1999.

[OUZ01] OUZOUNIS, V.; TSCHAMMER, V.; Towards Dynamic Virtual Enterprise. Towards the E-Society : E-commerce, E-Business, and E-Government, IFIP Academic Series, Kluwer Academic Publishers, October 2001.

[PUL01] PULVERMÜLLER, E.; SPECK, A.; COPLIEN, J. O.; D'HONDT, M.; DEMEUTER, W. Position Paper: Feature Interaction in Composed Systems. proceedings of ECOOP 2001, pg 3, 2001.

[RAT97] Rational Software et al. UML Notation Guide, Version 1.1, Object Management Group, http://www.omg.org, September 1997.

[REI98] M. REICHERT AND P. DADAM. ADEPTflex—Supporting Dynamic Changes of Workflows Without Loosing Control. Journal of Intelligent Information Sys-tems, 10(2), 1998.

[RIC02] RICCIOPPO, A.M. Construção de um repositório de definições de Processos baseado na Meta-Object Facility. Tese de mestrado, Orientação Prof. José Coelho, co-orientação Prof. Manuel de Jesus Mendes, DCA-FEEC, UNICAMP, Campinas, junho de 2002.

[RID99] RIDEAU, F.-R. Métaprogrammation et libre disponibilité des sources. In Actes de la conférence «Autour du Libre 1999», January 1999. http://tunes.org/~ fare/articles/ll99/index.fr.html.

[RIO02] O'RIORDAN, D. Business Process Standards for Web Services. TECT, extracted from Web Services Business Strategies and Architectures. by APSHANKAR, K. et al., Expert Press, 2002.

[ROS98] M. Rosenthal. Ein graphischer Editor zur Erstellung von Workflow-Spez-ifikationen. Diploma thesis, Department of Information Technology, University of Zurich, January 1998.

[SAC99] SACCONI, L.A. Nossa Gramática, Teoria e Prática, Editora Atual, 25ª. Edição, 1999.

[SIL00] SILVA, C.R.M.; SOTO, J. A.; MENDES, M. J.; CARVALHO, M.; SILVA, O. Integration of Workflow Management Systems with Planning Suites in Supply Chains. In: Proceedings 16th International Conference on CAD/CAM Robotics and Factories of the Future (CARS & FOF 2000), St, Augustine, Trinidad and Tobago, June 26-28, 2000.

[SIL02a] SILVA, C.R.M.; MEJIA, J. A. S.; MENDES, M. J. Integrating a Workflow Engine and a MOF Repository to an Open Service Platform. In: Proceedings of the 3rd. IFIP Working Conference on Infrastructures for Virtual Enterprise (PRO-VE 2002), Sesimbra, Portugal, May 1-3, 2002.

[SIL02b] SILVA, C.R.M.; MENDES, M.J. Extending Enterprise Portal Capabilities with a Workflow Meta-Component Framework. In: Proceedings of the 2nd. IFIP Conference on E-Commerce, E-Business and E-Government (I3E 2002), Lisbon - Portugal, October 7-9, 2002.

[SIT01] SITESCAPE. Collaboration + Workflow = e-Process. September 2001, SiteScape, Inc. Maynard, Massachusetts.

[SMI82] SMITH, B. C. Reflection and Semantics in a Procedural Language. MIT Laboratory of Computer Science (MIT Technical Report 272), Massachusetts Institute of Technology, Boston-Massachusetts, 1982 (Ph.D. Thesis).

[SOL99] SOLTES, D. Metadata and Meta-information - Old Concepts and new Challenges. IASSIST Quarterly, vol. 23, number 3, pg. 12-24, Fall 1999.

[STE03] STEINHAU, R., Model Driven Architecture Definition and Methodology, Deliverable 3.1, IST-2001-37785, MODA-TEL Consortium 2003.

[SUN01] SUN. JSR #40 – JMI - Java Metadata Interface API Specification. Sun Microsystems, Inc., http://java.sun.com/aboutJava/communityprocess/jsr/jsr_040_mof.html, 2001.

[SUN02] SUN. JSR #26 – UML Profile for EJB - Draft Specification. Sun Microsystems, Inc., http://java.sun.com/aboutJava/communityprocess/jsr/jsr_026, 2001.

[SUN99] SUN. RMI-IIOP Programmer's Guide. Sun Microsystems, Inc. http://www-106.ibm.com/developerworks/java/rmi-iiop/docs/aix130/rmi_iiop_pg.html, 1999.

[TIL02] TILKOV. S. "MDA from a Developer's Perspective", White paper in the site: http://www.theserverside.com//resources/article.jsp?l=MDA, december, 2002.

[URT98] C. URTADO AND C. OUSSALAH. Complex Entity Versioning at two Granularity Levels. Information Systems, 23(3/4), 1998.

[VAN98] VANEGAS, R.; ZINKY, J.; LOYALL, J.P.; Karr, D.; SCHANTZ, R.E.; BAKKEN, D.E. QuO's Runtime Support for Quality of Service in Distributed Objects. Proceedings of Middleware 98, the IFIP International Conference on Distributed Systems Platform and Open Distributed Processing, September 1998.

[W3C00] W3C. Extensible Markup Language (XML) 1.0, Second Edition (W3C Recommendation). World Wide Web Consortium, 2000, http://www.w3.org/TR/REC-xml.

[WAD02] J.P. Wadsack, J.P. and Jahnke, J.H. Towards Model-Driven Middleware Maintenance. In Proc. of the OOPSLA 2002 Workshop on Generative Techniques in the context of Model-Driven Architecture, Seattle, USA. November 2002.

[WAN00] WANG, W; HIDVÉGI, Z; BAILEY, A.D.; WHINSTON, A.B. E-Process Design and Assurance Using Model Checking. IEEE Computer Magazine, Cover Feature, October, 2000.

[WAT98] WATANABE, T.; YONEZAWA, A. Reflection in an Object-Oriented Concurrent Language. Proc. ACM Conference on Object-Oriented Programming, Systems, Languages and Applications (OPSLA' 88) , pp.306-315, San Diego, CA, USA, September 1988.

[WFM00a] WfMC. Workflow Interoperability – MIME Bindings, WfMC standard number TC 1018, version 1.2, January, 2000.

[WFM00b] WfMC. Workflow Interoperability – Wf-XML Bindings, WfMC standard number TC 1023, version 1.0, May, 2000.

[WFM95] WfMC. WAPI – Naming Conventions, WfMC standard number TC 1013, November, 1995.

[WFM95] WFMC. Workflow Reference Model. Workflow Management Coalition, Document WfMC-TC-1003, v1.1, january, 1995.

[WFM98a] WfMC. Workflow Client API Specifications (WAPI), WfMC standard number TC 1002, july, 1998.

[WFM98b] WfMC. Workflow Audit Data Specification, WfMC standard number TC 1015, version 1.1, September, 1998.

[WFM99a] WfMC. Terminology & Glossary , standard number TC 1011, version 3.0, February, 1999.

[WFM99b] WfMC. Process Definition Meta Model & WPDL, WfMC standard number TC 1016, version 1.1, October, 1999.

[WFM99c] WfMC. Workflow Interoperability – Abstract specifications, WfMC standard number TC 1012, version 2.0, December, 1999.

[ZIA02] ZIADE, T.; TRAVERSON, B.; JEZEQUEL, J. "From a UML Platform Independent Component Model to Platform Specific Component Models", WiSME@UML'2002,

Workshop in Software Model Engineering, Tuesday October 1st 2002, Dresden, Germany.

Apêndice A - Meta-modelo CCA

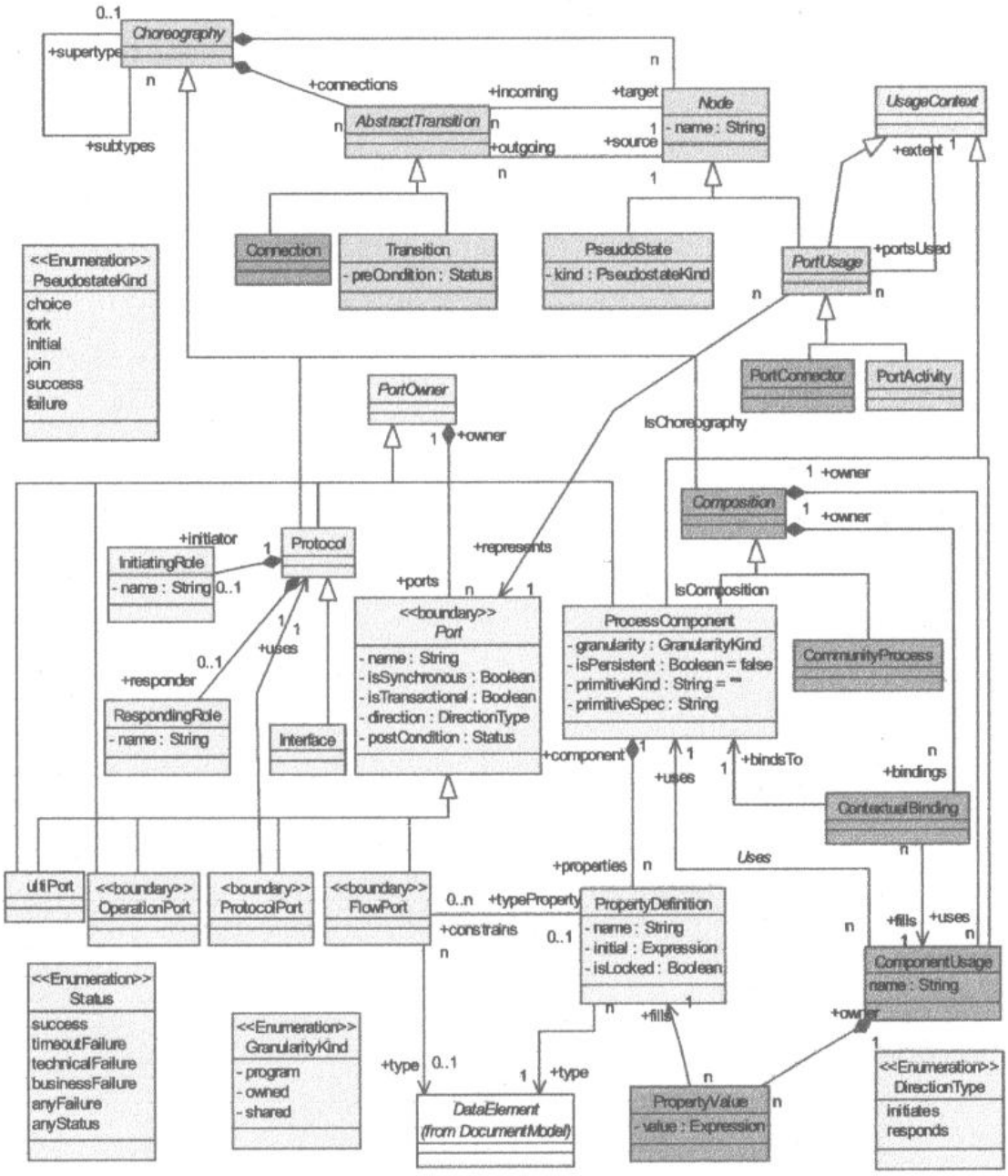

Figura A-1 - Meta-modelo MOF para o Perfil EDOC – CCA [OMG02b].

Apêndice B - Arquivo IDL para BPRI

WF-BPRI. IDL

```
#include <EDOC.idl>

module BPRI {

  // Forward declarations
  //-------------------------------------------------------------------
  interface WfRepository;                // New Interface
  interface WfExecutionObject;
  interface WfProcess;
  interface WfProcessExt;                // New Interface
  interface WfProcessDefExt;             // New Interface
  interface WfProcessIterator;
  interface WfRequester;
  interface WfProcessMgr;
  interface WfProcessMgrExt;             // New Interface
  interface WfActivity;
  interface WfActivityExt;               // New Interface
  interface WfActivityIterator;
  interface WfResource;
  interface WfAssignment;
  interface WfAssignmentExt;             // New Interface
  interface WfAssignmentIterator;
  interface WfEventAudit;
  interface WfEventAuditIterator;
  interface WfCreateProcessEventAudit;
  interface WfStateEventAudit;
  interface WfAssignmentEventAudit;


  // DataTypes
  //-------------------------------------------------------------------
  struct NameValueInfo{
    string attribute_name;
    string type_name;
  };
 struct ProcessDefinition {    // new
    string name;
    string version;
    string procDefID;
};
```

```
typedef sequence<NameValueInfo> NameValueInfoSequence;
struct NameValue{
  string the_name;
  any the_value;
};
typedef sequence <NameValue> NameValueSequence;
typedef sequence <string> NameSequence;
struct BaseError {
  long exception_code;
  string exception_source;
  any exception_object;
  string exception_reason;
};
typedef sequence <BaseError> BaseErrorSequence;

// DataTypes
//-------------------------------------------------------------------
typedef sequence<WfProcessExt> WfProcessSequence;                  // New
typedef sequence<WfActivityExt> WfActivitySequence;                // New
typedef sequence<WfAssignmentExt> WfAssignmentSequence;            // New
typedef sequence<WfEventAudit> WfEventAuditSequence;
typedef NameValueInfoSequence ProcessDataInfo;
typedef NameValueSequence ProcessData;

enum workflow_stateType{ open, closed };
enum while_openType{not_running, running };
enum why_not_runningType{ not_started, suspended };
enum how_closedType{ completed, terminated, aborted };
enum process_mgr_stateType{enabled, disabled };
enum assignment_stateType{potential, active};                      // New


// Exceptions
//-------------------------------------------------------------------
exception InvalidPerformer{};
exception InvalidState{};
exception InvalidData{};
exception TransitionNotAllowed{};
exception CannotResume{};
exception CannotSuspend{};
exception AlreadySuspended{};
exception CannotStop{};
exception NotRunning{};
exception HistoryNotAvailable{};
exception NotEnabled{};
exception AlreadyRunning{};
exception CannotStart{};
exception ResultNotAvailable{};
exception CannotComplete{};
exception NotAssigned{};
exception SourceNotAvailable{};
```

```
  exception RequesterRequired{};
  exception NotSuspended{};
  exception CannotChangeRequester{};
  exception InvalidResource{};
  exception UpdateNotAllowed{};
  exception InvalidRequester{};
  exception NameMismatch{};
  exception InvalidQuery{};
  exception GrammarNotSupported{};
  exception BaseException {
    BaseErrorSequence errors;
  };

  // Interfaces
  //--------------------------------------------------------------------
  interface WfRepository {       // New
    EDOC::BusinessProcesses::BusinessProcessesPackage get_pd(in string ProcessId) raises
(BaseException);
    void subscribe(in string ProcessId, in WfProcessMgrExt manager) raises (BaseException);
    void  unsubscribe(in string ProcessId, in WfProcessMgrExt manager) raises (BaseException);
  };

  interface WfRequester {
    long how_many_performer()  raises (BaseException);
    WfProcessIterator get_iterator_performer()  raises (BaseException);
    WfProcessSequence get_sequence_performer(  in long max_number )  raises (BaseException);
    boolean is_member_of_performer(in WfProcess member )  raises (BaseException);
    void receive_event(in WfEventAudit event) raises (BaseException, InvalidPerformer);
  };

  interface WfExecutionObject   {
    workflow_stateType workflow_state()  raises (BaseException);
    while_openType while_open()  raises (BaseException);
    why_not_runningType why_not_running()  raises (BaseException);
    how_closedType how_closed()  raises (BaseException);
    NameSequence valid_states()  raises (BaseException);
    string state()  raises (BaseException);
    void change_state(in string new_state) raises (BaseException, InvalidState,  TransitionNotAllowed);
    string name()  raises(BaseException);
    void set_name( in string new_value)  raises (BaseException);
    string key()  raises(BaseException);
    string description()  raises(BaseException);
    void set_description(in string new_value)  raises (BaseException);
    ProcessData process_context()  raises(BaseException);
    void set_process_context(in ProcessData new_value)  raises (BaseException, InvalidData,
UpdateNotAllowed);
    unsigned short priority()  raises(BaseException);
    void set_priority(in unsigned short new_value)  raises (BaseException);
    void resume()  raises (BaseException, CannotResume,  NotRunning, NotSuspended);
    void suspend()  raises (BaseException, CannotSuspend,  NotRunning, AlreadySuspended);
    void terminate()  raises (BaseException, CannotStop, NotRunning);
```

```
    void abort()  raises (BaseException, CannotStop, NotRunning);
    long how_many_history()  raises (BaseException, HistoryNotAvailable);
    WfEventAuditIterator get_iterator_history(in string query,in NameValueSequence names_in_query)
raises(BaseException, HistoryNotAvailable);
    WfEventAuditSequence get_sequence_history(in long max_number)  raises(BaseException,
HistoryNotAvailable);
    boolean is_member_of_history(in WfExecutionObject member)  raises(BaseException);
    long last_state_time() raises(BaseException);
  };

  interface WfProcessMgr  {
    long how_many_process()  raises (BaseException);
    WfProcessIterator get_iterator_process()  raises (BaseException);
    WfProcessSequence get_sequence_process(in long max_number )  raises (BaseException);
    boolean is_member_of_process(in WfProcess member )  raises (BaseException);
    process_mgr_stateType process_mgr_state()  raises(BaseException);
    void set_process_mgr_state(in process_mgr_stateType new_state)  raises(BaseException,
TransitionNotAllowed);
    string name()  raises(BaseException);
    string description()  raises(BaseException);
    string category()  raises(BaseException);
    string version()  raises(BaseException);
    ProcessDataInfo context_signature()  raises (BaseException);
    ProcessDataInfo result_signature()  raises (BaseException);
    WfProcess create_process(in WfRequester requester)  raises (BaseException, NotEnabled,
InvalidRequester, RequesterRequired);
  };

  interface WfProcessMgrExt : WfProcessMgr  {  // New
     struct ProcessDefinition {
        string name;
        string version;
        string ProcessDefRef;
      };
     void update()   raises(BaseException);
  };

  interface WfProcess : WfExecutionObject {
    WfRequester requester()  raises(BaseException);
    void set_requester( in WfRequester new_value)  raises (BaseException, CannotChangeRequester);
    long how_many_step()  raises (BaseException);
    WfActivityIterator get_iterator_step()  raises (BaseException);
    WfActivitySequence get_sequence_step(in long max_number )  raises (BaseException);
    boolean is_member_of_step(in WfActivity member )  raises (BaseException);
    WfProcessMgr manager()  raises(BaseException);
    ProcessData result()  raises (BaseException, ResultNotAvailable);
    void start()  raises (BaseException, CannotStart,  AlreadyRunning);
    WfActivityIterator get_activities_in_state(in string state)  raises(BaseException, InvalidState);
  };
  interface WfProcessExt : WfProcess {  // New
     struct Subscription{
```

```
        WfRequester the_subscriber;
        string          the_event_type;
    };
    typedef sequence<Subscription> SubscriptionList;

  void subscribe_event(in WfRequester requester, in string eventType)  raises (BaseException, NotEnabled,
InvalidRequester, RequesterRequired);
  void unsubscribe_event(in WfRequester requester)  raises (BaseException, NotEnabled, InvalidRequester,
RequesterRequired);
 };

 interface WfProcessDefExt {  // New
   typedef sequence<EDOC::BusinessProcesses::ProcessPortConnector> ProcessPortConnectorSequence;
   typedef sequence<EDOC::BusinessProcesses::ProcessFlowPort> ProcessFlowPortSequence;
   typedef sequence<EDOC::BusinessProcesses::Activity> ActivitySequence;
   typedef sequence<EDOC::BusinessProcesses::DataFlow> DataFlowSequence;
   typedef sequence<EDOC::BusinessProcesses::InputGroup> InputGroupSequence;
   typedef sequence<EDOC::BusinessProcesses::OutputGroup> OutputGroupSequence;
   struct ControlFlowData {
    ProcessPortConnectorSequence the_connectors;
    ActivitySequence            the_activities;
   };
   struct InputData {
    InputGroupSequence      the_input_group;
    ProcessFlowPortSequence  the_flow;
   };
   struct OutputData {
    OutputGroupSequence     the_output_group;
    ProcessFlowPortSequence  the_flow;
   };
   string ext_pd_type() raises (BaseException);
   Reflective::RefObject extract_artifact(in string artifactName)  raises (BaseException);      ///?????????
   DataFlowSequence extract_data_flow() raises (BaseException);
   ControlFlowData extract_control_flow() raises (BaseException);
   InputData extract_input_data()  raises (BaseException);
   OutputData extract_output_data()  raises (BaseException);
   EDOC::BusinessProcesses::CompoundTask  extract_compound_task() raises (BaseException);
 };

 interface WfProcessIterator  {
  WfProcessExt get_next_object ()  raises (BaseException);
  WfProcessExt get_previous_object()   raises (BaseException);
  WfProcessSequence get_next_n_sequence(in long max_number)  raises (BaseException);
  WfProcessSequence get_previous_n_sequence(in long max_number)  raises (BaseException);
 };

 interface WfActivity : WfExecutionObject {
  long how_many_assignment()  raises (BaseException);
  WfAssignmentIterator get_iterator_assignment()  raises (BaseException);
  WfAssignmentSequence get_sequence_assignment(in long max_number ) raises (BaseException);
  boolean is_member_of_assignment(in WfAssignment member ) raises (BaseException);
```

```
    WfProcess container()  raises(BaseException);
    ProcessData result()  raises(BaseException, ResultNotAvailable);
    void set_result(in ProcessData result)  raises (BaseException, InvalidData);
    void complete()  raises (BaseException, CannotComplete);
  };
  interface WfActivityExt : WfActivity {  // New
    void receive_event(in WfEventAudit event) raises (BaseException, InvalidPerformer);
  };

  interface WfActivityIterator {
    WfActivityExt get_next_object ()  raises (BaseException);
    WfActivityExt get_previous_object()  raises (BaseException);
    WfActivitySequence get_next_n_sequence(in long max_number)  raises (BaseException);
    WfActivitySequence get_previous_n_sequence(in long max_number)  raises (BaseException);
  };

  interface WfAssignment {
    WfActivity activity()  raises(BaseException);
    WfResource assignee()  raises(BaseException);
    void set_assignee(in WfResource new_value)  raises (BaseException, InvalidResource);
  };

  interface WfAssignmentExt : WfAssignment {  // New
    string get_assignment_sate()   raises(BaseException);
    WfAssignment get_assignment(in string processRole, in string selectionRule, in string creationRule)
raises(BaseException);
  };

  interface WfAssignmentIterator {
    WfAssignment get_next_object ()  raises (BaseException);
    WfAssignment get_previous_object()  raises (BaseException);
    WfAssignmentSequence get_next_n_sequence(in long max_number)  raises (BaseException);
    WfAssignmentSequence get_previous_n_sequence(in long max_number)  raises (BaseException);
  };

  interface WfResource {
    long how_many_work_item()  raises (BaseException);
    WfAssignmentIterator get_iterator_work_item()  raises (BaseException);
    WfAssignmentSequence get_sequence_work_item(in long max_number )  raises (BaseException);
    boolean is_member_of_work_items(in WfAssignment member )  raises (BaseException);
    string resource_key()  raises(BaseException);
    string resource_name()  raises(BaseException);
    void release(in WfAssignment from_assigment,in string release_info)  raises (BaseException,
NotAssigned);
  };

  interface WfEventAudit {
    WfExecutionObject source()  raises(BaseException, SourceNotAvailable);
    long time_stamp()  raises(BaseException);
    string event_type()  raises(BaseException);
    string activity_key()  raises(BaseException);
```

```
  string activity_name()  raises(BaseException);
  string process_key()  raises(BaseException);
  string process_name()  raises(BaseException);
  string process_mgr_name()  raises(BaseException);
  string process_mgr_version()  raises(BaseException);
};

interface WfEventAuditIterator {
  WfEventAudit get_next_object ()  raises (BaseException);
  WfEventAudit get_previous_object()  raises (BaseException);
  WfEventAuditSequence get_next_n_sequence(in long max_number)  raises (BaseException);
  WfEventAuditSequence get_previous_n_sequence(in long max_number)  raises (BaseException);
};

interface WfCreateProcessEventAudit : WfEventAudit{
  string p_activity_key()   raises(BaseException);
  string p_process_key()   raises(BaseException);
  string p_process_name()   raises(BaseException);
  string p_process_mgr_name()   raises(BaseException);
  string p_process_mgr_version()  raises(BaseException);
};

interface WfStateEventAudit : WfEventAudit {
  string old_state()   raises(BaseException);
  string new_state()   raises(BaseException);
};

interface WfDataEventAudit : WfEventAudit {
  ProcessData old_data() raises(BaseException);
  ProcessData new_data() raises(BaseException);
};

interface WfAssignmentEventAudit : WfEventAudit{
  string old_resource_key() raises(BaseException);
  string old_resource_name()  raises(BaseException);
  string new_resource_key()  raises(BaseException);
  string new_resource_name() raises(BaseException);
};

}; // end module BPRI
```

Apêndice C - Glossário

Modelo	É a representação de uma parte da funcionalidade, estrutura e/ou comportamento de um sistema. Deve ter uma descrição formal, no sentido de que cada elemento tenha uma semântica bem definida e esteja definido em conformidade com a especificação MOF.
Meta-modelo	É um modelo usado para representar um outro modelo. Corresponde a uma linguagem de modelagem. Por exemplo, a linguagem de modelagem UML possui um meta-modelo que representa cada um de seus elementos, sejam eles gráficos ou não.
Infra-estrutura para MDA	É um ambiente de modelagem de sistemas que deve incluir uma linguagem para definir linguagens (meta-linguagem) e mecanismos para definir, integrar e transformar modelos.
Meta-linguagem	É uma linguagem usada para definir linguagens de modelagem. Por exemplo, o meta-meta-modelo MOF é uma meta-linguagem usada para definir meta-modelos correspondentes a linguagens de modelagem genéricas.
Sistema	No contexto deste trabalho, um sistema é visto como uma aplicação de software que é modelada através dos princípios e metodologia da arquitetura MDA. Por exemplo, o ambiente de execução de processos de negócios é um sistema modelado através de artefatos EDOC.
Metadados	Podem ser definidos genericamente como dados sobre dados e são usados para definir, relacionar e manipular objetos de dados. São exemplos de metadados: os significados de atributos e métodos de um objeto, esquemas de bancos de dados, artefatos de

	modelos UML, etc.
Meta-informação	Na maioria dos casos o conceito de meta-informação é visto como sinônimo do conceito de metadado. Há casos, entretanto, em que algumas distinções são feitas entre estes conceitos com base nas diferenças que existem entre os termos *dado (um fato)* e *informação (fato + significado)*. Quando estas distinções são consideradas, o termo meta-informação é visto como o conteúdo semântico de uma descrição de dados [SOL99] ou como a representação de aspectos estruturais e comportamentais de entidades de primeira classe [COS01].
Linguagem de Modelagem	É uma linguagem que define os elementos de um modelo e seus relacionamentos. Por exemplo, a linguagem UML é uma linguagem de modelagem que permite a definição de modelos que representam artefatos de sistemas de software genéricos.
Infra-estrutura de Software	É um conjunto de partes de software e/ou hardware assumidos como presentes por usuários quando estes desenvolvem um artefato de software.
Abstração	É a descrição de algo que omite alguns detalhes irrelevantes para um contexto específico.
Refinamento	É uma descrição detalhada que está em conformidade com uma abstração correspondente.
Plataforma	É uma infra-estrutura de software implementada com uma tecnologia específica (Unix, CORBA, Windows, etc) sobre uma tecnologia de hardware específica.
Visão	É uma representação de um sistema como um todo, a partir da perspectiva de um conjunto relacionado de interesses (*concerns*).
Ponto de Vista	É a especificação de convenções para a construção e uso de uma *Visão*, ou seja, um *template,* a partir do qual se desenvolvem visões pelo estabelecimento de propósitos e audiência, bem como as técnicas para sua criação e análise.

Mapeamento	É um conjunto de regras e técnicas usados para modificar um modelo e criar um outro modelo a partir do anterior [OMG03d].
Modelo Independente de Plataforma	Em inglês: PIM (*Platform Independent Model*). Fornece especificações formais da estrutura e funcionalidade de um sistema se abstraindo de detalhes técnicos [OMG03d].
Modelo Específico de Plataforma	Em inglês: PSM (*Platform Specific Model*). É expresso em termos do modelo de especificação da plataforma específica. Um PSM usa os conceitos próprios de uma plataforma específica, tais como: modelo de componentes, mecanismos de exceção, tipos de parâmetros, etc [OMG03d].
Framework	É um conjunto de classes que incorpora um projeto abstrato de solução que pode ser aplicada a uma família de problemas relacionados [JOH88].
Federação de repositórios	É uma federação de sistemas de gerência de meta-informações que compartilham os mesmos esquemas de bancos de dados e interagem entre si para permitir acesso uniforme e transparente a uma série de recursos distribuídos nos repositórios da federação. É um conceito similar ao de federações de bancos de dados, pois os gerentes de repositórios são aplicações de bancos de dados [HAA02].
Catálogos de Coleções	São espaços de nomes (*namespaces*) para as referências que representam os meta-objetos que compõem uma coleção.
Processo de Negócio	É um conjunto de procedimentos ou atividades que coletivamente realizam um objetivo de negócio dentro do contexto de uma estrutura organizacional que define papéis e relacionamentos funcionais para os participantes do processo [WFM99].
Workflow	É a automação de um processo de negócio, no todo ou em parte, durante a qual documentos, informações ou tarefas são passados de um participante para outro,

	para que ações sejam tomadas de acordo com um conjunto de regras procedimentais [WfM99].
Sistema de Gerência de Workflows	É um sistema que define, cria e gerencia a execução de workflows através do uso de sistemas de software que são capazes de interpretar definições de processos, interagir com participantes dos workflows e, onde for necessário, fazer uso de ferramentas e aplicações externas [WfM99].
Participante de Workflow	É um recurso que realiza o trabalho representado por uma instância de atividade de workflow.
Item de trabalho	É a representação do trabalho a ser processado por um participante de workflow no contexto de uma atividade de uma instância do processo em questão.
Automação de Processos de Negócios	É a utilização de sistemas de software para projetar, executar e acompanhar processos de negócios manuais com o objetivo de reduzir custos e aumentar a eficiência e a consistência dos processos considerados.
Definição de Processo	É a representação de um processo de negócio de forma a permitir a manipulação automatizada de um processo. Uma definição de processo consiste de uma rede de atividades e seus relacionamentos, critérios para indicar o início e término do processo e informações sobre as atividades individuais, tais como: participante responsável, recursos e aplicações necessárias, etc [WfM99].
Instância de Processo	É a representação de um processo dentro de um ambiente de execução definido por um sistema de gerência de workflows.
Aplicação de Workflow	É um termo geral para um programa de software que interage com um ambiente de execução de workflows para executar parte do processamento necessário para a conclusão de uma atividade de workflow.
Lista de Trabalhos	É uma lista de itens de trabalhos associados com um dado participante de workflow.

Coreografia	É um conceito abstrato, próprio do perfil CCA que está relacionado com a ordem com que as portas são executados em um protocolo ou processo [OMG02b].
Backend	É um termo usado para referenciar serviços transparentes associados com um serviço que é utilizado diretamente por um cliente.

Apêndice D - Publicações

- MENDES, M. J.; CARVALHO, M.; SILVA, O.; **SILVA, C.R.M**.; SOTO, J. A.; "Integration of Workflow Management Systems with Planning Suites in Supply Chains". In: Proc. of the 16th International Conference on CAD/CAM Robotics and Factories of the Future (CARS & FOF 2000), St, Augustine, Trinidad and Tobago, June 26-28, 2000.

- **SILVA, C.R.M**.; MEJIA, J. A. S.; MENDES, M. J. "Integrating a Workflow Engine and a MOF Repository to an Open Service Platform". In: Proc. of the 3rd. IFIP Working Conference on Infrastructures for Virtual Enterprise (PRO-VE 2002), Sesimbra, Portugal, May 1-3, 2002.

- **SILVA, C.R.M**. and MENDES, M.J. "Extending Enterprise Portal Capabilities with a Workflow Meta-Component Framework". In: Proc. of the 2nd. IFIP Conference on E-Commerce, E-Business and E-Government (I3E 2002), Lisbon - Portugal, October 7-9, 2002.

- MENDES, M. J., RODRIGUES, M, FIGUEIREDO, A, KAMADA, A. DAMASCENO, L. RIZZO, N. **SILVA, C.R.M**., "Federated Web Services", In. Proc. of the XXII International Conference of the Chilean Computer Science Society, SCCC 2002, Copiapó, Atacama, CHILE , 6-8 November 2002.

- **SILVA, C.R.M** and MENDES, M.J. "A MOF-based Approach to Business Process in Model-driven Architectures", In: Proc. of the 3rd. IFIP Conference on E-Commerce, E-Business and E-Government (I3E 2003), Guarujá – SP - Brazil, September 22-24, 2003.

Índice Analítico