UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

DEPARTAMENTO DE INFORMÁTICA

MESTRADO EM INFORMÁTICA

VERUSKA CARRETTA ZAMBORLINI

ESTUDO DE ALTERNATIVAS DE MAPEAMENTO DE ONTOLOGIAS DA LINGUAGEM ONTOUML PARA OWL: ABORDAGENS PARA REPRESENTAÇÃO DE INFORMAÇÃO TEMPORAL

VITÓRIA

2011

VERUSKA CARRETTA ZAMBORLINI

ESTUDO DE ALTERNATIVAS DE MAPEAMENTO DE ONTOLOGIAS DA LINGUAGEM ONTOUML PARA OWL: ABORDAGENS PARA REPRESENTAÇÃO DE INFORMAÇÃO TEMPORAL

Dissertação submetida ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Mestre em Informática, na área de concentração Inteligência Computacional.
Orientador: Prof. Giancarlo Guizzardi, Ph.D.

VITÓRIA

2011

VERUSKA CARRETTA ZAMBORLINI

Estudo de Alternativas de Mapeamento de Ontologias da Linguagem OntoUML para OWL

Abordagens Representação de Informação Temporal

Dissertação submetida ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Mestre em Informática, na área de concentração Inteligência Computacional.

Aprovada em 20 de janeiro de 2011.

COMISSÃO EXAMINADORA

Prof. Giancarlo Guizzardi, Ph.D.
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. Flávio Miguel Varejão
Universidade Federal do Espírito Santo

Prof. Dr. Fernando Náufel do Amaral
Universidade Federal Fluminense

Prof. Daniel Schwabe, Ph.D.
Pontifícia Universidade Católica do Rio de Janeiro

Ao Mestre.

À família.

Aos mestres.

Aos amigos.

AGRADECIMENTOS

Algumas pessoas marcam a nossa vida para sempre,
umas porque nos vão ajudando na construção,
outras porque nos apresentam projetos de sonho
e outras ainda porque nos desafiam a construí-los.
[autor desconhecido]

... a Deus e à minha família, que são meu refúgio e minha fortaleza, minha fonte da alegria de viver, da coragem pra encarar os desafios, enfim, do amor...

... à minha família, meus pais, Toninho e Rita, meus irmãos, Léo e Mila, meus cunhados Alê e Lucas, meus sobrinhos, Maria Julia, Antonio e Maria Rita, pela paciência, pelo apoio, pelo carinho, por aceitarem minha ausência e por fazerem questão da minha presença, pela companhia tão preciosa, da qual tantas vezes tive que abdicar e senti imensa falta ...

... em especial aos meus pais, meus maiores mestres, que com muito amor fizeram todo sacrifício necessário para me dar condições de chegar até aqui ...

... em especial e à minha irmã pelas revisões, dicas e conselhos valiosos, e ao meu irmão pela paciência de me ajudar na revisão e impressão da disseração ...

... muito obrigada!

... ao orientador Giancarlo Guizzardi, pelo conhecimento compartilhado, pelos poucos minutos que valeram por vários, pela paciência com meu perfeccionismo e minha inconstância, por acreditar em mim há tanto tempo... muito obrigada!

... à amiga/professora Rosane de Oliveira, pelas conversas, pelo incentivo, pelo carinho, pelos ensinamentos acadêmicos e pessoais, enfim, pelos momentos felizes ou tristes em que pude contar com seu carinho e seu apoio... muito obrigada!

... aos outros professores/colegas/amigos do Departamento de Informática, pelas conversas, conselhos, livros, links e por participarem e incentivarem a minha formação acadêmica desde a graduação... muito obrigada!

... aos professores, amigos e colegas do Laboratório NEMO, pelos momentos juntos de trabalho, de estudo e de lazer, pelas discussões e reflexões sobre os mais variados assuntos, pelas contribuições profissionais e pessoais na minha vida. Devo destacar um agradecimento ao professor João Paulo Almeida pelas vezes em que recorri, e fui prontamente atendida, em especial pelo apoio na implementação dos mapeamentos, que foi possível graças à colaboração do colega Roberto Carraretto. Agradeço especialmente aos queridos amigos, Alessander, Bernardo Braga, Calhau, Evellin, Lucas e Tina, que acompanharam minha saga, por todo apoio que me ofereceram... muito obrigada!

... ao professor Fernando Náufel pela valiosa ajuda em questões sobre OWL e DL e pela feliz oportunidade, infelizmente tardia, de participar de reuniões periódicas com ele, juntamente com o prof. Giancarlo, e com os amigos Alessander e Bernardo Braga... muito obrigada!

... aos meus familiares e amigos em geral, aos da UFES e aos "normais", pelo carinho, palavras de incentivo, por não desistirem de mim mesmo quando eu nunca tinha tempo, e quando meus únicos assuntos eram mestrado e ontologias. Em especial, agradeço ao amigo Bernardo Gonçalves, pelas valiosas e variadas discussões, inclusive sobre Eletrocardiologia... muito obrigada!

... a CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), pelo apoio financeiro, viabilizado por meio de uma bolsa de mestrado.

RESUMO

O uso de ontologias para representação do conhecimento acerca de um domínio tem crescido na computação, especialmente no contexto da Web Semântica. Entretanto, observa-se uma forte tendência em priorizar a eficiência computacional em detrimento da qualidade da representação, deixando de lado importantes características do domínio, como aspectos temporais. De fato, a linguagem OWL, padrão adotado na Web Semântica, não permite, em princípio, a representação de informação temporal. As principais estratégias propostas na literatura para resolver este problema, as abordagens Perdurantista 4D e de Reificação Temporal, não provêem suporte para apoiar as decisões do modelagem, deixando a responsabilidade nas mãos do usuário. Neste contexto, este trabalho objetiva, através da aplicação de resultados vindos da disciplina filosófica de Ontologia Formal, fundamentar tais abordagens para representação de informação temporal em OWL, provendo uma estrutura-base ontologicamente fundamentada e diretrizes para guiar sua utilização, avaliando e comparando as alternativas propostas. Ademais, uma vez que não é tarefa trivial a aplicação direta das noções ontológicas, observa-se que, utilizando-se uma linguagem de modelagem que as incorpore, é possível definir mapeamentos sistemáticos a partir desta linguagem para OWL de tal forma que se permita a geração automática de modelos num processo de Engenharia de Ontologias. Assim, outro objetivo deste trabalho é propor alternativas de mapeamentos entre a linguagem nível ontológico OntoUML e a linguagem de nível epistemológico OWL. Realiza-se ainda um estudo de caso no domínio de Eletrocardiografia, aplicando-se uma das alternativas para codificar automaticamente uma ontologia de referência de ECG, seguindo-se um processo de Engenharia de Ontologias, permitindo representar a informação temporal. Por fim, os resultados obtidos neste trabalho são relevantes no contexto de Engenharia de Ontologias por prover mapeamentos sistemáticos que, uma vez implementados, permitem que a codificação da ontologia de referência seja realizada de forma automatizada e eficiente, preservando a qualidade do modelo de referência e evitando erros decorrentes do processo de codificação manual; e, além disso, permitem contornar a limitação da linguagem OWL quanto à representação da informação temporal.

Palavras-chave: Ontologia. Codificação de Ontologias. Mapeamento. Representação de Informação Temporal em OWL.

ABSTRACT

The application of ontologies for domain knowledge representation has gained much interest in the later years, specially in the context of Semantic Web. However, it is been observed a strong tendency a strong tendency to prioritize the computational efficiency over quality of representation, leaving out important features of the domain, such as temporal aspects. Indeed, the OWL, standard adopted for the Semantic Web, does not allow, in principle, the representation of temporal information. The main strategies proposed in the literature to solve this problem, namely, the Perdurantistic and Temporal Reification approaches, do not provide support for the modeling support the decisions, leaving the responsibility for the user. In this context, this research aims, through the application of results from the philosophical discipline of Formal Ontology, to provide for such approaches a base structure ontologically founded and guidelines to guide its use, evaluating and comparing the proposed alternatives. Moreover, since not a trivial task the direct application of the ontological notions, it is observed that using a modeling language that incorporates them, it is possible to define systematic mappings from this language to OWL so that it allows automatic generation of models in an Ontology Engineering process. Thus, another goal is to propose alternative mappings between the ontological level language OntoUML and epistemological level language OWL. Moreover, one of these alternatives have been applied in a case study in the Electrocardiography domain for automatically codifying an ECG reference ontology following an Ontology Engineering process, allowing to represent the temporal information. Finally, the results of this research are relevant in the context of Ontology Engineering for providing systematic mappings which, once implemented, allows the reference ontology codification to be conducted in an automated and efficient way, while preserving the quality of the reference model and avoiding errors arising from manual coding process and, moreover, allowing to circumvent the limitation of OWL on the representation of temporal information.

Keywords: Ontology. Ontology Codification. Mapping. Representation of Temporally Changing Information in OWL.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIAÇÕES

BWW – Bunge-Wand-Weber

DL – *Description Logics*

DOLCE – Ontologia Descritiva para Engenharia Linguística e Cognitiva

ECG – Eletrocardiograma

GFO – *General Formal Ontology*

GOL – *General Ontology Language*

HMA – Hipótese de Mundo Aberto

HNNU – Hipótese de Nomes Não-Únicos

HNU – Hipótese de Nomes Únicos

IA – Inteligência Artificial

IC – *Individual Concept*

LPO – Lógica de Primeira Ordem

OntoUML – *Ontological Unified Modelling Language*

OWL – *Web Ontology Language*

RDF – *Resource Description Framework*

RC – Representação do Conhecimento

TS – *Time Slice*

UFO – *Unified Foundational Ontology*

UML – *Unified Modelling Language*

W3C – *World Wide Web Consortium*

WS – *Web* Semântica

XML – *eXtensible Markup Language*

SUMÁRIO

1. Introdução

1. INTRODUÇÃO

Ontologia, no seu sentido literal de "estudo da existência", existe (pelo menos) desde o século IV a.C., quando Aristóteles a descreveu como "a ciência do ser enquanto ser". O termo em si foi cunhado no meio filosófico no século XVII, e é usado neste meio desde então para se referir tanto a uma disciplina quanto a sistemas de categorias independentes de linguagem. Na computação, o termo foi usado pela primeira vez em 1967 na área de Processamento de Dados, e desde então tem sido aplicado em outras áreas da computação, porém com diferentes significados e propósitos (GUIZZARDI, 2005).

De modo geral, em áreas como Inteligência Artificial (IA) e Web Semântica (WS), o termo ontologia tem sido empregado para designar modelos construídos com base em algum formalismo. O que tem variado é a expressividade desses formalismos e, consequentemente, a qualidade dos modelos produzidos (GUIZZARDI, 2005). De fato, quanto mais características de um dado domínio se pretende representar, mais expressiva deve ser a linguagem conceitual subjacente ao modelo. Entretanto, aumentar o poder de expressividade geralmente implica também no aumento da complexidade computacional (BRACHMAN; LEVESQUE, 2004).

Este conflito de expressividade *versus* eficiência, clássico na área de Representação do Conhecimento (RC), é particularmente agravado no contexto de representação de ontologias, uma vez que idealmente uma linguagem para tal intento deve ser suficientemente expressiva para caracterizar formalmente as distinções ontológicas. Embora seja uma preocupação autêntica da computação garantir a eficiência computacional, de acordo com Guarino, "importantes questões a respeito de diferentes hipóteses ontológicas subjacentes ao nosso uso de termos têm sido simplesmente deixadas de lado enquanto busca-se simplificação lógica e tratabilidade computacional" (2009, p. 1).

Guarino (1994, 2009) propõe a existência de um nível ontológico para linguagens de modelagem, que contempla aquelas cujo foco principal é a qualidade do modelo produzido, sem considerar questões computacionais. Por outro lado, linguagens que tenham como foco questões computacionais, como decidibilidade e eficiência, recomendáveis para realizar raciocínio automatizado, são ditas de nível lógico/epistemológico,.

Assim, este trabalho se insere no contexto de lidar com tal conflito, buscando alternativas de se representar ontologias em uma linguagem de nível epistemológico, contudo sem deixar de lado as

distinções ontológicas. Ao contrário, tais distinções são consideradas imprescindíveis para guiar a modelagem de tal forma que se possa garantir a qualidade do modelo e ainda contornar algumas limitações da linguagem de representação.

1.1. MOTIVAÇÃO E OBJETIVOS DA PESQUISA

Diante das perspectivas apresentadas, tem-se observado uma forte tendência em priorizar a eficiência computacional em detrimento da qualidade da ontologia (GUARINO, NICOLA, 2009; GUIZZARDI, 2007). Uma classe particular de linguagens lógicas em RC, chamada de Lógicas de Descrição (DL – *Description Logics*), tem ganhado muito interesse nos últimos anos, principalmente por causa da sua crescente popularidade na iniciativa da Web Semântica. Este é o formalismo que define a semântica da linguagem para representação de ontologias na Web, OWL (*Web Ontology Language*), padrão de fato recomendado pela W3C (*World Wide Web Consortium*) (HORROCKS ET AL., 2006).

Uma vez que DL consiste de uma família de subconjuntos da lógica clássica de primeira ordem tipicamente projetados para serem decidíveis, esta família de linguagens lógicas, bem como as linguagens nela baseadas, apresentam-se inexpressivas para representar algumas características importantes do domínio. Também por adotarem a hipótese de mundo aberto e monotonicidade, tais linguagens são apropriadas para representação de informação estática, ou seja, informação cujo valor verdade é fixo. Por um lado, a hipótese de mundo aberto implica que só é verdadeira a informação explicitamente declarada ou deduzida a partir dela, e, por outro lado, a monotonicidade implica que o valor verdade da informação deve permanecer inalterado independentemente da adição de informação no modelo. Em outras palavras, tem-se que a informação sobre o domínio pode ser completada, porém não pode ser efetivamente modificada (HOEKSTRA, 2009).

Sendo assim, não é possível, em princípio, representar em OWL informação dinâmica, ou seja, informação cujo valor verdade não é fixo, mas pode mudar de um estado/momento para outro. Em particular, uma vez que a dinamicidade da informação é indispensável para capturar a modalidade subjacente às distinções ontológicas, tem-se que estas não podem ser (trivialmente) capturadas em OWL.

Uma alternativa para se representar a informação dinâmica é a interpretação temporal, ou seja, associar um intervalo de tempo ao valor verdade da informação. Neste contexto, vários autores têm investigado estratégias para estender a expressividade de DL permitindo representar a

informação que pode mudar temporalmente (ou, simplificadamente, informação temporal) (LUTZ ET AL., 2008). Porém, tais estratégias trazem consigo, além do aumento da complexidade computacional por conta do aumento do poder de expressividade da linguagem, um problema de incompatibilidade com as linguagens clássicas adotadas na Web Semântica, e, por conseguinte, com os editores e máquinas de inferência já desenvolvidos e popularizados.

Numa direção diferente de pesquisa, outros autores estão especialmente interessados em manter a compatibilidade com as DLs clássicas adotadas na Web Semântica propondo estruturas para representar tal informação temporal em OWL, mantendo-se um histórico de informação. Alguns adotam uma abordagem baseada na visão perdurantista 4D, em que um objeto do domínio é visto como uma minhoca temporal, ou seja, um objeto 3D que se estende numa quarta dimensão temporal (KRIEGER, 2008; WELTY, CHRIS; FIKES, 2006). Por outro lado, outros autores partem para uma abordagem baseada na estratégia de reificação temporal, em que informações mutáveis de um objeto são reificadas/objetificadas, atribuindo-se a elas um intervalo de tempo no qual elas são válidas (GANGEMI, ALDO, 2005; O'CONNOR; DAS, 2011). Tais abordagens, entretanto, não provêem ao modelador orientação quanto às decisões de modelagem e suas consequências.

Dessa forma, um **objetivo** deste trabalho é aplicar resultados vindos da disciplina filosófica de Ontologia Formal para propor abordagens ontologicamente bem-fundamentadas para representação de informação temporal em OWL segundo as abordagens perdurantista 4D e de reificação temporal, dando suporte às decisões de modelagem. Para tanto, os seguintes objetivos específicos são estabelecidos: (i) analisar a fundamentação ontológica da visão filosófica Perdurantista 4D e da visão de Reificação Temporal; (ii) para cada teoria, propor estruturas ontologicamente fundamentadas de representação de informação temporal em OWL, com diretrizes para guiar sua utilização; (iii) para cada caso, avaliar e comparar as alternativas propostas;

Uma vez que não é tarefa trivial a aplicação direta de tais noções ontológicas, observa-se que, utilizando-se uma linguagem de modelagem que as incorpore, é possível definir mapeamentos sistemáticos a partir desta linguagem para OWL de tal forma que se permita a geração automática de modelos num processo de Engenharia de Ontologias. Este processo consiste de três fases:,(i) na fase de análise, a ontologia é descrita numa linguagem de nível ontológico, e, (ii) conforme decisões tomadas na fase de projeto, (ii) codifica-se (automaticamente) a ontologia para uma linguagem de nível epistemológico na fase de implementação. (GUIZZARDI; HALPIN, 2008).

Assim, outro **objetivo** deste trabalho é propor alternativas de mapeamentos entre a linguagem nível ontológico OntoUML (*Ontological Unified Modeling Language*) (GUIZZARDI, 2005) e a linguagem de nível epistemológico OWL. A escolha da linguagem OntoUML se dá por ser objeto de estudo do grupo de pesquisa no qual este trabalho se insere, além de ser (até onde sabemos) o único sistema de linguagem que incorpora explicitamente as distinções modais de nível ontológico no metamodelo da linguagem. Por outro lado, a linguagem OWL foi escolhida por ser o padrão de fato recomendado pela W3C que tem sido usado por diversos grupos dentro e fora do Brasil para criar ontologias.

Neste contexto, os seguintes objetivos específicos são estabelecidos: (i) especificar mapeamentos para cenário estático, ou seja, mapear as entidades do domínio considerando-se sua natureza ontológica, porém sem considerar os aspectos modais/temporais; (ii) especificar mapeamentos para cenário dinâmico, seguindo propostas ontologicamente fundamentadas para representação de informação temporal em OWL, ou seja, mapear as entidades do domínio considerando-se sua natureza ontológica com os aspectos modais/temporais; (iii) implementar os mapeamentos; e, finalmente, (iv) escolher e aplicar uma alternativa de mapeamento em estudo de caso no domínio de Eletrocardiografia, seguindo-se um processo de Engenharia de Ontologias.

Enfim, os objetivos supracitados compõem o **objetivo principal** desta pesquisa, que é aplicar resultados advindos da disciplina filosófica de Ontologia Formal para fundamentar a representação de ontologias em OWL. O Quadro 1 sintetiza a problematização da pesquisa apresentando em linhas gerais o tema, o problema, a hipótese, o recorte e os objetivos da pesquisa.

Tema	Representação e raciocínio automático sobre ontologias.
Problema	O conflito poder de expressividade *versus* eficiência computacional tem levado à priorização deste último, em detrimento da qualidade da ontologia, dificultando ou impossibilitando a representação de importantes características do domínio.
Hipótese	É possível contornar limitações de expressividade de linguagens computacionalmente eficientes utilizando-se estratégias fundamentadas em resultados advindos da disciplina filosófica de Ontologia Formal.
Recorte	Uso de noções ontológicas para guiar a representação de ontologias na linguagem OWL, visando a qualidade do modelo, capturando importantes características do domínio como a informação que muda temporalmente.
Objetivos	Aplicar resultados advindos da disciplina filosófica de Ontologia Formal para fundamentar a representação de ontologias em OWL:
	Fundamentar ontologicamente as abordagens Perdurantista 4D e de Reificação Temporal para representação de informação temporal em OWL.
	Propor alternativas de mapeamento sistemático de modelos descritos na linguagem de nível ontológico OntoUML para modelos descritos na linguagem OWL.

Quadro 1. Síntese da problematização da pesquisa.

1.2. HISTÓRICO DO DESENVOLVIMENTO DO TRABALHO

A motivação prática deste trabalho surgiu a partir de trabalhos anteriores que visavam implementar em OWL uma ontologia de Eletrocardiografia descrita em OntoUML (ZAMBORLINI, 2008; GONÇALVES ET AL., 2008; ZAMBORLINI ET AL., 2008; GONÇALVES; ZAMBORLINI; GUIZZARDI; PEREIRA FILHO, 2009; GONÇALVES; ZAMBORLINI; GUIZZARDI, 2009). O objetivo era verificar indícios de funcionamento do coração a partir de registros de ECG (Eletrocardiograma).

Contudo, verificou-se que era possível apenas a representação de cenários estáticos, ou seja, de momentos/instantes da realidade, em vista do limitado poder de expressividade de OWL. Por exemplo, se para um dado ciclo/batimento infere-se que o coração está funcionando, sendo sua representação estática, esta informação permanecerá verdadeira mesmo que outro ciclo, referente a um outro momento, signifique o contrário. Assim, não fazia sentido representar vários ciclos simultaneamente, uma vez que a informação inferida não poderia ser "apagada" por outra informação. Dessa forma, era necessário re-instanciar o modelo a cada ciclo do ECG que se pretendia avaliar, perdendo-se o resultado da avaliação anterior.

Percebeu-se então que a impossibilidade de se representar a informação temporal inviabilizava a avaliação do funcionamento do coração de forma sequencial, e, consequentemente, a utilização de tal modelo para identificação de (suspeita de) falhas. De fato, para tanto é necessário observar variações no padrão da freqüência e amplitude de ciclos/batimentos sequenciais, não adiantando observar um único ciclo de ECG isoladamente. Na verdade, este problema de representação da informação temporal, decorrente do limitado poder de expressividade de OWL, está presente em qualquer ontologia ou modelo conceitual que tenha por objetivo descrever um domínio dinâmico, ou seja, um domínio em que os estados do mundo (*state of affairs*) podem mudar. Enfim, diante da diferença de expressividade entre as linguagens, percebeu-se não ser possível (trivialmente) capturar em OWL os aspectos modais/temporais subjacentes às distinções ontológicas embutidas na linguagem OntoUML.

Outra dificuldade encontrada foi a ausência de uma proposta de mapeamento sistemático entre as linguagens OntoUML e OWL, de forma que se realizou um mapeamento manual e *ad-hoc*, tornando o modelo resultante passível de introdução de erros no processo de mapeamento. Enfim, identificou-se a necessidade de se estudar mapeamentos sistemáticos a partir da linguagem

OntoUML, que permitissem realizar automaticamente a codificação segundo um processo de Engenharia de Ontologia, e também representar satisfatoriamente domínios dinâmicos.

Para atingir os objetivos desta pesquisa, realiza-se numa primeira etapa, a estruturação de um referencial teórico com base em produções científicas que versam sobre o significado de ontologias e seu uso na computação, bem como sobre noções ontológicas básicas. Dá-se atenção especialmente àquelas reunidas na UFO (*Unfied Foundational Ontology*), ontologia que fundamenta a linguagem OntoUML, ambas propostas por Guizzardi (2005). Ademais, busca-se o aprofundamento do entendimento das linguagens OntoUML e OWL, e da diferença de expressividade entre elas.

Realiza-se também uma revisão bibliográfica sobre abordagens para representação de informação temporal em OWL, em que foram escolhidas duas abordagens comuns na literatura para representação da informação em OWL. Tais abordagens são a perdurantista e a de reificação temporal, cujas teorias também foram investigadas com o objetivo de melhor compreendê-las e relacioná-las com as noções ontológicas. Como resultado desta etapa, são propostas duas abordagens ontologicamente fundamentadas para representação de informação temporal em OWL, baseadas nas duas teorias supracitadas. O resultado relativo à abordagem perdurantista foi publicado sob o título "*On the representation of temporally changing information in OWL*" (ZAMBORLINI; GUIZZARDI, 2010a). Já o resultado relativo à abordagem de reificação temporal está descrito no relatório técnico "*An Ontologically-founded Reification Approach for Representing Temporally Changing Information in OWL*" (ZAMBORLINI; GUIZZARDI, 2010b) que ainda será submetido para avaliação e eventual publicação.

Numa segunda etapa de desenvolvimento, uma vez conhecidas as características da linguagem de origem e da linguagem alvo, e tendo em mãos duas abordagens ontologicamente fundamentadas para representação de informação temporal em OWL, são propostas alternativas sistemáticas de mapeamento de modelos descritos em OntoUML para modelos descritos em OWL, permitindo-se representar tanto cenários estáticos quanto dinâmicos.

Na terceira e última etapa, as alternativas propostas são implementadas numa infra-estrutura de OntoUML e, como estudo de caso aplica-se o mapeamento automatizado de uma das alternativas para uma ontologia no domínio de Eletrocardiografia, segundo um processo de Engenharia de Ontologias,.

1.3. ESTRUTURA DA DISSERTAÇÃO

A estrutura da pesquisa está organizada em seis capítulos, sendo o primeiro este capítulo introdutório, e os demais conteúdos desdobrados da seguinte forma:

O **segundo capítulo** apresenta o referencial teórico a respeito de Ontologias, do seu propósito geral e das suas aplicações em computação. São apresentadas algumas noções advindas da disciplina filosófica de Ontologia Formal que servem de base para as abordagens e mapeamentos propostos neste trabalho. Mais especificamente, é apresentada a ontologia de fundamentação UFO que combina várias teorias da disciplina de Ontologia Formal.

Nesse capítulo também atenta-se para a existência de diferentes níveis de linguagem, com diferentes propósitos. Destaca-se uma abordagem de engenharia de ontologias que propõe o uso de duas classes de linguagens. Então, são apresentadas as linguagens utilizadas neste trabalho, a OntoUML, de nível ontológico fundamentada na UFO, e OWL, de nível epistemológico fundamentada em DL. Cabe ressaltar que parte do referencial teórico está apresentada no decorrer do capítulo 3, visando uma composição sequencial mais adequada à compreensão do leitor.

No **terceiro capítulo** revisita-se duas estratégias usadas para representação de informação temporal em OWL, a saber, a Abordagem Perdurantista 4D e a Abordagem de Reificação Temporal. Propõe-se, então, duas abordagens baseadas nessas estratégias, fundamentado-as em resultados de Ontologia Formal. As duas abordagens são comparadas e relacionadas, mostrando-se a possibilidade de abordagens híbridas.

No **quarto capítulo** são propostos mapeamentos sistemáticos de modelos OntoUML em modelos OWL, baseados tanto na UFO quanto nas abordagens propostas no capítulo 3. Também é apresentado um sistema de mapeamento implementado na linguagem Java, como complemento à infra-estrutura OntoUML proposta por Carraretto (2010).

O **quinto capítulo**, por sua vez, apresenta um estudo de caso no domínio de Eletrocardiografia, seguindo-se um processo de Engenharia de Ontologia para codificar em OWL uma ontologia de ECG descrita em OntoUML, aplicando-se um dos mapeamento propostos para representar informação temporal.

Finalmente, o **sexto capítulo** apresenta as considerações finais deste trabalho, destacando-se as conclusões, as limitações, as dificuldades encontradas, além de propostas e perspectivas para trabalhos futuros.

2. Ontologias

2. ONTOLOGIAS

Ontologia é uma palavra de origem latina cujo significado pode ser entendido como "o estudo da existência". Etimologicamente, *ont-* vem do particípio presente do verbo grego *enai*, que significa "ser", e *logia* em latim, que significa estudo. Como a "ciência do ser enquanto ser", seu estudo remete a Aristóteles, no século IV a.C., embora o termo tenha sido cunhado apenas no século XVII no meio filosófico. Desde então tem sido usado na filosofia para designar tanto uma disciplina – utilizando-se o termo com "O" maiúsculo – quanto um sistema de categorias independente de linguagem apropriado para conceituação de teorias científicas (GUIZZARDI, 2005).

No início do século XX, o filósofo germânico Edmund Husserl definiu o termo **Ontologia Formal** para se referir a uma parte específica da disciplina filosófica de Ontologia. Fazendo-se uma analogia com a Lógica Formal, enquanto esta lida com estruturas lógicas formais (e.g. verdade, validade, consistência) independentemente de sua veracidade, a Ontologia Formal lida com estruturas ontológicas formais (e.g. teoria das partes, teoria do todo, tipos e instanciação, identidade, dependência, unidade), isto é, com aspectos formais de objetos independentemente da sua natureza particular (GUIZZARDI, 2005).

A primeira menção ao termo na computação data de 1967 por G. H. Mealy, num trabalho sobre os fundamentos da modelagem de dados, na área de processamento de dados (MEALY, 1967 apud GUIZZARDI, 2005). Desde então tem sido aplicado em diversas áreas, como Sistemas de Informação, Engenharia de Software, Inteligência Artificial e Web Semântica, porém com diferentes significados e propósitos. Nas duas primeiras áreas, ontologia é comumente utilizada em conformidade com seu significado em Filosofia, ou seja, como um sistema de categorias independente de linguagem. Em contrapartida, em outras áreas como IA e WS, tal palavra é usada, em geral, para designar um artefato concreto, projetado para um propósito específico, e representado em uma linguagem específica. A Figura 1 ilustra uma grande variedade de artefatos classificados com ontologias na literatura de computação, desde simples catálogos (lista de termos) até teorias lógicas formais que permitem raciocínio automatizado (GUIZZARDI, 2005, 2007).

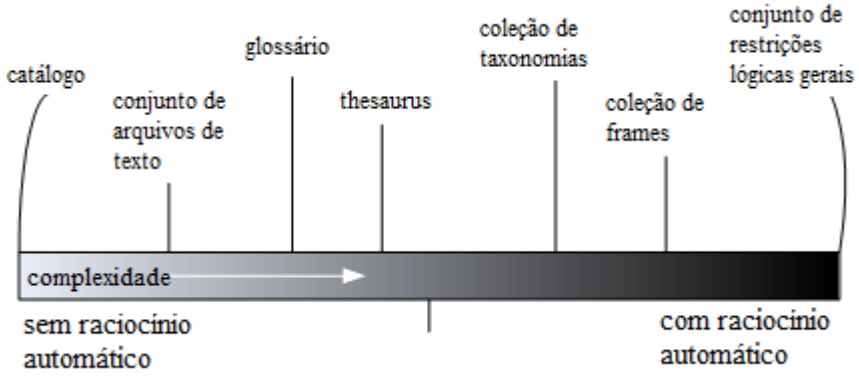


Figura 1. Diferentes tipos de especificações classificadas como ontologies na literatura de ciência da computação. **Fonte**: Smith e Welty (2001) apud Guizzardi (2005, p. 78) (adaptado).

Enquanto estes exemplos referem-se a modelos que descrevem entidades específicas de domínio, um outro tipo de ontologia, chamado **Ontologia de Fundamentação**, reúne resultados de Ontologia Formal num sistema de categorias independentes de domínio (e.g. conceitos como parte, todo, papel e evento), usadas para articular conceituações dos diversos domínios. Um exemplo conhecido na literatura de computação é a Ontologia Bunge-Wand-Weber (BWW), proposta por Wand and Weber, baseada na teoria metafísica original desenvolvida por Bunge (WAND; WEBER, 1995) .

Em particular, Guizzardi e colegas propõem em uma série de publicações (GUIZZARDI, 2005; GUIZZARDI ET AL., 2006; GUIZZARDI; WAGNER, 2010; MASOLO, CLAUDIO ET AL., 2005) uma ontologia de fundamentação chamada **UFO** (*Unified Foundational Ontology*). Neste trabalho interessa em especial uma parte denominada UFO-A, que define termos relacionados a aspectos estruturais como conceitos gerais de objetos, suas propriedades intrínsecas e relacionais, os tipos que eles instanciam, os papéis que eles desempenham, etc. Esta teoria é proposta por Guizzardi (2005), e usada como fundamentação ontológica na aplicação de um método de avaliação à Linguagem de Modelagem Conceitual UML 2.0 (*Unified Modeling Language*). Tal processo deu origem à linguagem chamada de **OntoUML**, que é então uma versão estendida da UML 2.0 ontologicamente bem-fundamentada.

De fato, Guarino (2009) afirma que, apesar de não haver um consenso geral sobre ter as distinções ontológicas embutidas na linguagem de representação, a OntoUML é uma proposta concreta nesta direção. Ainda segundo o autor (1994, 2009), as linguagens de representação do conhecimento podem ser divididas em **níveis de linguagem**. As de nível ontológico são justamente aquelas que incorporam as distinções ontológicas, tendo como foco principal a qualidade do modelo produzido sem se preocupar com questões computacionais. Por outro lado, linguagens com foco em questões

computacionais, que permitam raciocínio automatizado, são ditas de nível lógico/epistemológico segundo o autor. Uma linguagem deste nível é a **OWL** (*Web Ontology Language*), padrão de fato recomendado pela W3C para representação de ontologia na Web.

Neste sentido, nota-se um conflito entre focar na qualidade da representação ou em questões computacionais, derivado do clássico conflito da área de RC entre poder de expressividade e eficiência computacional. Assim, Guizzardi defende a necessidade uma abordagem de **Engenharia de Ontologia** para lidar com esse conflito (GUIZZARDI, 2007, 2010a; GUIZZARDI; HALPIN, 2008).

De modo geral, o termo ontologia é aplicado neste trabalho de forma flexível, com o intuito de evitar um preciosismo prejudicial à sua compreensão, embora sua fundamentação siga a idéia de autores como Guarino, Guizzardi e outros, de que ontologia é essencialmente mais do que uma especificação descrita num formalismo qualquer.

2.1. ONTOLOGIA FORMAL

A Ontologia Formal se refere a uma parte específica da disciplina filosófica de Ontologia que lida com estruturas ontológicas formais, isto é, com aspectos formais de entidades independentemente da sua natureza particular. Esta seção apresenta algumas noções ontológicas básicas que se refletem no sistema de categorias da UFO-A, bem como este mesmo sistema.

2.1.1. Noções Ontológicas Básicas

As principais primitivas de modelagem consideradas neste trabalho são conceitos e relações binárias, que correspondem respectivamente a propriedades unárias e binárias em Representação do Conhecimento. Conforme defendido por Guarino (1994, 2009) e Guizzardi (2005), tais propriedades apresentam diferentes meta-propriedades formais, e, portanto, desempenham diferentes papéis, conforme apresentado nos quadros a seguir. O Quadro 2 mostra as meta-propriedades básicas das **propriedades unárias** usadas neste trabalho que são três: **identidade, rigidez** e **dependência**. Por outro lado, o Quadro 3 mostra as meta-propriedades básicas das **propriedades binárias** usadas neste trabalho que são duas: **formal** e **material**.

	DESCRIÇÃO	EXEMPLO
IDENTIDADE	Alguns conceitos têm a característica de prover um **critério ou princípio de identidade** para seus indivíduos que permita distingui-los e contá-los. Uma heurística útil para chegar a tal conclusão é verificar se é possível contar os indivíduos.	O conceito **Maçã** provê um critério de identidade aos indivíduos por ele classificados se faz sentido perguntar "quantas maçãs há nesta caixa?". Em contrapartida, o conceito **Vermelho** não provê tal critério se não faz sentido perguntar "quantos vermelhos há nesta caixa?".
RIGIDEZ	**Conceitos rígidos:** têm a característica de serem necessariamente aplicáveis a seus indivíduos enquanto eles existirem.	O conceito **Pessoa** é rígido se todos os indivíduos por ele classificados, e.g. **João**, não podem deixar de ser **Pessoa** enquanto existirem.
	Conceitos anti-rígidos: têm a característica de serem aplicáveis a seus indivíduos de maneira contingente.	O conceito **Estudante** é anti-rígido se todos os indivíduos por ele classificados, e.g. **João**, podem deixar de ser **Estudante** e ainda continuar existindo.
	Conceitos semi-rígidos: têm a característica de serem eventualmente aplicáveis a alguns de seus indivíduos, e necessariamente aplicáveis a outros.	O conceito **Sentável** é semi-rígido se é aplicado necessariamente a alguns dos indivíduos por ele classificados, e.g. uma **cadeira**, e também é aplicado eventualmente a outros, e.g. um **caixote**.
DEPENDÊNCIA	Um conceito C_1 é relacionalmente **dependente** de outro C_2 se para instanciar C_1 os indivíduos precisam participar de uma determinada relação com instâncias de C_2. Neste trabalho, consideram-se os seguintes tipos de dependência:	
	Dependência Genérica: se a relação que caracteriza a dependência pode mudar.	Um **coração** é genericamente dependente do tipo **Pessoa**, se ele deve sempre instanciar a relação **parte-de**, porém com um indivíduo qualquer do tipo **Pessoa**, que pode mudar. Neste caso, considera-se a possibilidade de transplante.
	Dependência Específica: se a relação que caracteriza a dependência não pode mudar.	Um **coração** é especificamente dependente de uma **pessoa**, se ele deve instanciar a relação **parte-de** sempre com um mesmo indivíduo do tipo **Pessoa**, que não pode mudar. Neste caso, desconsidera-se a possibilidade de transplante.
	Dependência Existencial: um tipo de dependência específica, em que a relação que caracteriza a dependência é tal que o indivíduo dependente existe somente se o outro indivíduo do qual aquele depende também existir.	O exemplo anterior é de dependência existencial se o **coração**, para existir, deve sempre instanciar tal relação. Noutro caso, considera-se a possibilidade de que o **coração** possa continuar existindo fora da **pessoa**.

Quadro 2. Meta-propriedades básicas das propriedades unárias.

	DESCRIÇÃO	EXEMPLO
FORMAL	**Relações formais** valem pela simples existência dos indivíduos que ela relaciona, ou seja, não dependem de uma entidade interventora para valer. Podem ser classificadas como **internas ou externas**.	
	Relações formais internas implicam em dependência existencial entre os indivíduos relacionados. Assim, a relação vale sempre que o indivíduo dependente existir.	A relação **parte-de** (geralmente[1]) implica em dependência existencial de uma das partes. Por exemplo, se um **coração** é **parte essencial de** uma **pessoa**, significa que este **coração** é existencialmente dependente desta **pessoa**, e que esta relação vale enquanto o **coração** existir.
	Relações formais externas são tipicamente relações de comparação entre propriedades dos indivíduos relacionados. Assim, valem sempre que os indivíduos e as referidas propriedades existirem.	A relação **mais-pesado-que** entre **João** e **José** é verdade enquanto os dois indivíduos existirem e o peso de **João** for maior que o de **José**.
MATERIAL	**Relações materiais** são aquelas que dependem de uma entidade interventora para valer.	A relação **casado-com** entre **João** e **Maria** é verdade enquanto existir um indivíduo interventor **casamento** mediando-os.

Quadro 3. Meta-propriedades das propriedades binárias.

2.1.2. UFO – *Unified Foundational Ontology*

A UFO é uma ontologia de fundamentação baseada em várias teorias de Ontologia Formal, Lógica filosófica, Filosofia da Linguagem, Linguística e Psicologia Cognitiva, e desenvolvida com o intuito de prover uma fundamentação ontológica para linguagens gerais de modelagem conceitual. Ela sintetiza resultados de outras ontologias de fundamentação, a GFO/GOL[2] e a OntoClean/DOLCE[3] que, embora apresentem propriedades interessantes, têm limitações consideráveis na habilidade de capturar os conceitos básicos de linguagens de modelagem

[1] Guarino (2009) indica que alguns tipos de relações que são tipicamente entendidas como formais internas, como todo-parte e constituição, em alguns casos podem valer apenas por um intervalo de tempo qualquer. Isso significa que, nestes casos, elas não são genuinamente formais internas, uma vez que não implicam em dependência existencial. Ele conclui que o problema de como caracterizar formalmente essas relações ainda está sendo investigado.

[2] *General Formal Ontology* (GFO) é ontologia subjacente à *General Ontology Language* (GOL), que é uma linguagem desenvolvida pelo grupo de pesquisa OntoMed da Universidade de Leipzig (DEGEN ET AL., 2001 apud GUIZZARDI; WAGNER, 2005)

[3] Junção da ontologia OntoClean com a Ontologia Descritiva para Engenharia Linguística e Cognitiva (DOLCE), desenvolvida pelo grupo de pesquisa ISTC-CNR-LOA (WELTY; GUARINO, 2001 apud GUIZZARDI; WAGNER, 2005).

conceitual. Assim, a proposta da UFO é justamente unificar essas ontologias aproveitando suas características positivas e sanando as limitações detectadas (GUIZZARDI; WAGNER, 2005, 2010).

Este trabalho aborda uma parte específica da UFO, denominada UFO-A, que trata particularmente de entidades chamadas endurantes (*endurants*), ou, mais intuitivamente, objetos ordinários do cotidiano que persistem no tempo, como uma pessoa, um buraco, um monte de areia. Esta ontologia é proposta por Guizzardi (2005), consistindo de conceitos e relações em diagramas estilo UML, além de uma caracterização formal de cada um deles. Devido à dificuldade de se traduzir alguns termos, eles são mantidos em língua inglesa nos diagramas, como foram concebidos, porém alguns são traduzidos no texto quando convém. Ademais, uma vez que toda esta seção é baseada nessa referência principal da UFO-A, toma-se a liberdade de omitir as citações no restante do texto.

As entidades da UFO-A relevantes a este trabalho são as Categorias de Universais e de Indivíduos, apresentadas nesta seção seguindo a abordagem em profundidade a seguir:

As Categorias de Universais e as Categorias de Indivíduos

I - As Categorias de Universais Monádicos e as Categorias de Indivíduos Singulares

i - Distinção entre *Substantials* e *Moments*

ii - *Substantials*

iii - M*oments*

II - As Categorias de Universais de Relação e as Categorias de Instâncias de Relações

As Categorias de Universais e as Categorias de Indivíduos

A UFO pode ser vista como uma teoria sobre categorias de **universais** e **indivíduos**. Em modelagem conceitual, tipicamente, os universais são conhecidos como conceito/classe ou relação/associação, enquanto as instâncias destes representam os indivíduos. Assim, uma vez que as categorias da UFO classificam esses elementos de modelagem conceitual, a categoria fundamental, chamada de **Entidade** ou **Coisa** (*Thing*), é ramificada nas categorias/tipos **Universal** (*Universal*) e **Indivíduo** (*Individual*), conforme apresentado no diagrama da Figura 2.

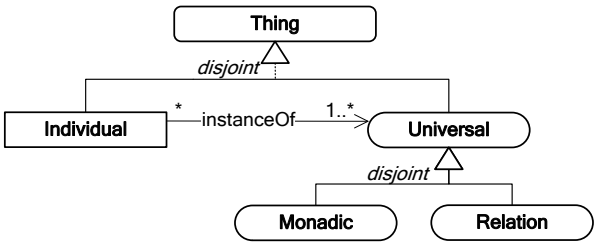


Figura 2. Fragmento da UFO - tipos de Universais e Indivíduos.

Assim, *Universal* é a categoria ou tipo geral que representa os padrões de características presentes em diferentes indivíduos. Por exemplo, este tipo se aplica aos conceitos ou categorias de indivíduos **Pessoa, Adulto** e **Cachorro**, que são padrões de características comuns presentes em certos indivíduos. Por outro lado, *Individual* é a categoria ou tipo geral que se aplica aos indivíduos, que são entidades que existem na realidade e possuem uma identidade única. Por exemplo, este tipo se aplica aos indivíduos **João** e **Rex**. Além disso, cada indivíduo num domínio deve ser instância de pelo menos um universal. Por exemplo, **João** é um indivíduo do domínio que instancia os universais **Pessoa** e **Adulto**, enquanto **Rex** instancia o universal **Cachorro**.

Classificam-se ainda os universais como **Monádico** (*Monadic Universal*) ou **de Relação** (*Relation Universal*). O tipo *Monadic* é a categoria que se aplica aos conceitos, que são padrões aplicados a indivíduos singulares, enquanto o tipo *Relation* se aplica às relações, que são padrões aplicados a grupos de dois ou mais indivíduos.

Os seguintes padrões de apresentação são adotados nos diagramas da UFO presentes nesta seção: (i) as categorias de universais são representadas com cantos arredondados, para fins de legibilidade, permitindo-se assim omitir o termo *Universal* após cada entidade; (ii) a relação de especialização, quanto pontilhada, significa que algumas entidades da hierarquia original da UFO foram omitidas.

I - As Categorias de Universais Monádicos e as Categorias de Indivíduos Singulares

A Figura 3 apresenta um esquema divido em três níveis que visa ilustrar como as entidades de um modelo conceitual específico de domínio, bem como suas instâncias, se relacionam com as entidades independentes de domínio previstas na UFO. As relações fundamentais entre as entidades são a de instanciação e de especialização. Enquanto as de instanciação valem entre uma entidade de um nível e outra do nível imediatamente superior, as de especialização valem entre entidades do mesmo nível, estabelecendo uma hierarquia entre elas. Assim, os indivíduos do domínio, representados no nível de Indivíduos, são instâncias dos conceitos do domínio, representados no nível Conceitual. Os conceitos do domínio, por sua vez, especializam as

categorias de indivíduos da UFO, representadas no mesmo nível, e instanciam as categorias de universais da UFO, representadas no nível Meta-Conceitual.

O esquema da Figura 3 mostra que os conceitos do domínio são genericamente classificados pela categoria *Monadic Universal* da UFO, enquanto suas instâncias são classificadas pela categoria *Individual*. Este esquema será expandido e apresentado em figuras sequenciais à medida que outras entidades da UFO são explicadas.

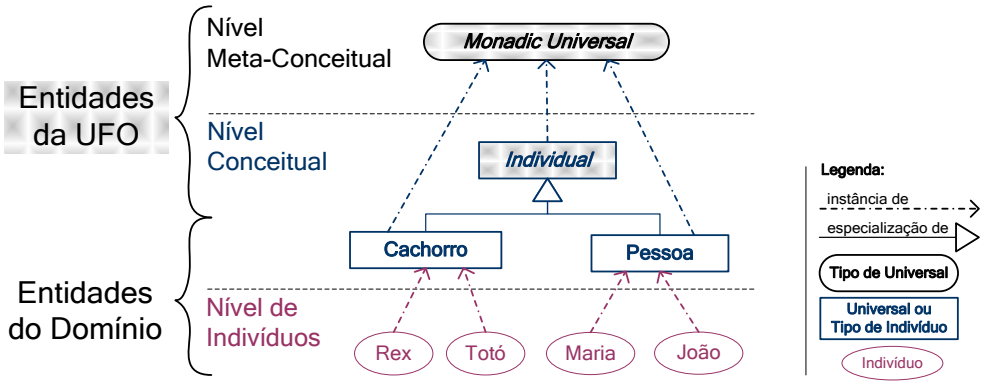


Figura 3. Fragmento da UFO em esquema de níveis - Universais Monádicos e Indivíduos.

As entidades da UFO, pertencentes aos níveis meta-conceiual e conceitual, são representadas com cor de fundo acinzentada, enquanto as de domínio, pertencentes aos níveis conceitual e de indivíduos, são representadas com cor de fundo branca. Por sua vez, as relações de instanciação são representadas por uma seta aberta e pontilhada, enquanto as de especialização são representadas por uma seta fechada.

No nível Meta-Conceitual está o tipo geral de universais monádicos chamado *Monadic Universal*. No nível Conceitual estão os tipos de indivíduos ou conceitos, que são instâncias do meta-conceito supracitado: o tipo geral de indivíduos, chamado *Individual*, e dois conceitos específicos de domínio: **Cachorro** e **Pessoa**. Finalmente, no nível de Indivíduos estão os indivíduos **Rex** e **Totó**, instâncias diretas de **Cachorro**, e **João** e **Maria**, instâncias diretas de **Pessoa**. É importante observar que as relações de instanciação se propagam através da relação de especialização (no sentido para cima na hierarquia), por exemplo, **João** é instância de **Pessoa** que por sua vez é especialização de *Individual*, portanto, **João** é instância de *Individual*.

i - Substantials e Moments

A principal distinção de tipos de universais (monádicos) e tipos de indivíduos em UFO é a distinção entre ***substantials*** e ***moments***. Este último é derivado da palavra *Momente* em alemão,

dos escritos de Husserl, e denota, em termos gerais, o que é por vezes chamado na literatura de tropo (*trope)*, indivíduo abstrato, ou instância de propriedade. Assim, ***moments*** podem ser entendidos como indivíduos que representam propriedades objetificadas de outros indivíduos, e que são inerentes a eles. Por exemplo, a **idade de João** é um *moment*, ou seja, um indivíduo que objetifica sua idade enquanto proporiedade abstrata, e é inerente a ele (**João**). Ou ainda, a **intensidade** é um *moment* da **dor de cabeça de João** que é um *moment* de **João**, sendo assim, a **intensidade** é inerente à **dor de cabeça**, que é inerente a **João**.

Observa-se que essa relação de inerência (*inherence*) forma uma cadeia de indivíduos existencialmente dependentes, que termina em um tipo de indivíduo particular, chamado *substantial* (no exemplo, **João**), de forma que este não é inerente a nenhum outro indivíduo. Assim a categoria geral de indivíduos é ramificada em *Substantial* e *Moment*, conforme apresentado no diagrama[4] da Figura 4.

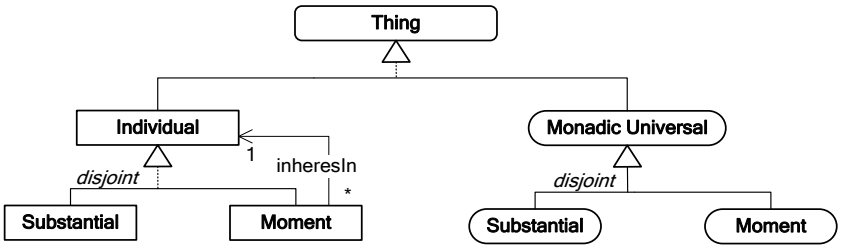


Figura 4. Fragmento da UFO – *Substantials* e *Moments*.

Essa mesma distinção se reflete na categorização dos universais, ou seja, os universais do tipo ***Substantial*** são aqueles que agrupam os indivíduos do tipo *substantial*, e, da mesma forma, os universais do tipo ***Moment*** agrupam os indivíduos do tipo *Moment*. Por exemplo, o conceito **Pessoa** é um universal do tipo *Substantial* porque seus indivíduos são do tipo *Substantial*, como **João**. Analogamente, o conceito **Dor de Cabeça** é um universal do tipo *Moment* porque seus indivíduos são do tipo *Moment* (**dor de cabeça de João**). A Figura 5 apresenta um esquema que estende aquele da Figura 3 com as entidades *Substantial* e *Moment* nos níveis meta-conceitual e conceitual.

Neste esquema, os conceitos **Cachorro** e **Pessoa** especializam o conceito *Substantial*, bem como instanciam o meta-conceito *Substantial*. Analogamente, o conceito **Dor de Cabeça** especializa o conceito *Moment* e instancia o meta-conceito *Moment*. Uma instância do conceito **Dor de Cabeça** é a **dor de cabeça do João**.

[4] Seguindo o padrão sugerido, as entidades nomeadas como *Substantial* no diagrama se referem a *Substantial Individual* no lado esquerdo, e a *Substantial Universal* no outro lado. Idem para as entidades nomeadas como *Moment*.

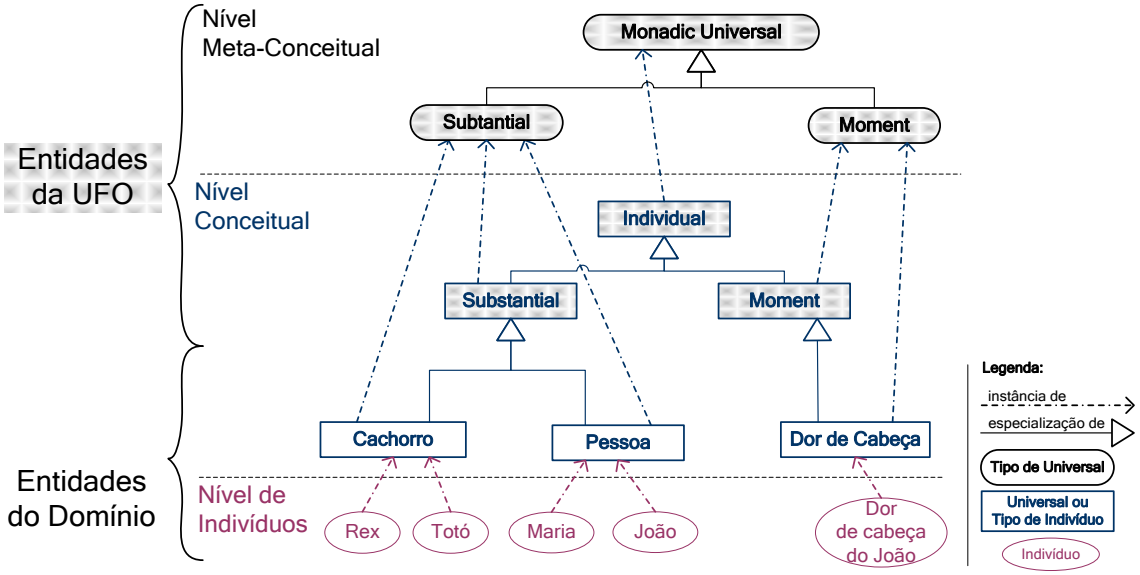


Figura 5. Fragmento da UFO em esquema de níveis – *Substantials* e *Moments*.

ii - Substantials

Dentre os indivíduos classificados como *Substantial*, interessam neste trabalho aqueles que têm uma identidade bem definida, classificados como **Objeto**[5] (*Object*). Este tipo ainda se divide entre Complexo Funcional, Coleção e Quantidade. O tipo **Complexo Funcional** (*Functional Complex*) é tal que suas instâncias podem ser compostas por outros complexos funcionais, desde que as partes exerçam papéis diferentes no todo. Por exemplo, o **corpo humano** é um complexo funcional composto por partes com diferentes papéis, como **coração** e **cérebro**.

Em contrapartida, as instâncias tipo **Coleção** (*Collective*) possuem partes que exercem o mesmo papel funcional no todo (GUIZZARDI, 2010b), por exemplo, uma **floresta** (como conjunto de **árvores**) ou uma **pilha de livros**. Finalmente, o tipo **Quantidade** (*Quantity*) (GUIZZARDI, 2010c) agrupa indivíduos que são porções maximais de uma quantidade de matéria, por exemplo, **a quantidade (maximal) de água dentro de um copo**.

O fragmento de diagrama da UFO apresentado na Figura 6 mostra a especialização do tipo de indivíduos *Substantial*. Outro padrão adotado nos diagramas da UFO apresentados nesta seção é omitir as entidades anteriores àquela principal que está sendo especializada, mostrando-a em cor de fundo cinza escuro.

[5] A UFO define um outro tipo de *Substantial*, chamado Porção de Matéria (*Ammount of Matter*), que porém está fora do escopo deste trabalho.

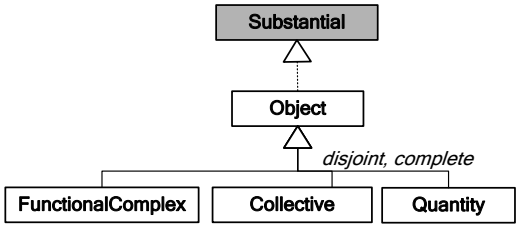


Figura 6. Fragmento da UFO – especialização do tipo de indivíduos *Substantial*.

Com relação ao tipo de universal *Substantial* (GUIZZARDI ET AL., 2004), ele é especializado segundo a aplicação das noções ontológicas básicas de identidade e rigidez aos universais deste tipo (ver Quadro 2 da seção 2.1.1). A distinção básica sub-classifica universais deste tipo como *Sortal* ou *Mixin*, conforme apresentado no diagrama da Figura 7. Os universais do tipo ***Sortal*** são tais que agregam indivíduos com o mesmo princípio de identidade. Por exemplo, supondo que a **impressão digital** defina a identidade de uma **pessoa**, são universais do tipo *Sortal* os conceitos **Pessoa**, **Cliente Pessoa Física** e **Adulto**, uma vez que todos agregam indivíduos que possuem o mesmo princípio de identidade, como **João**, **Maria** e **José**. Em contraste, universais do tipo ***Mixin*** são tais que agregam indivíduos com princípios de identidade diferentes. Seguindo o mesmo exemplo, supondo também que o **CNPJ** defina a identidade de uma **empresa**, então o conceito **Item Assegurável**, que agrega **pessoas** e **empresas**, é um universal do tipo *Mixin*.

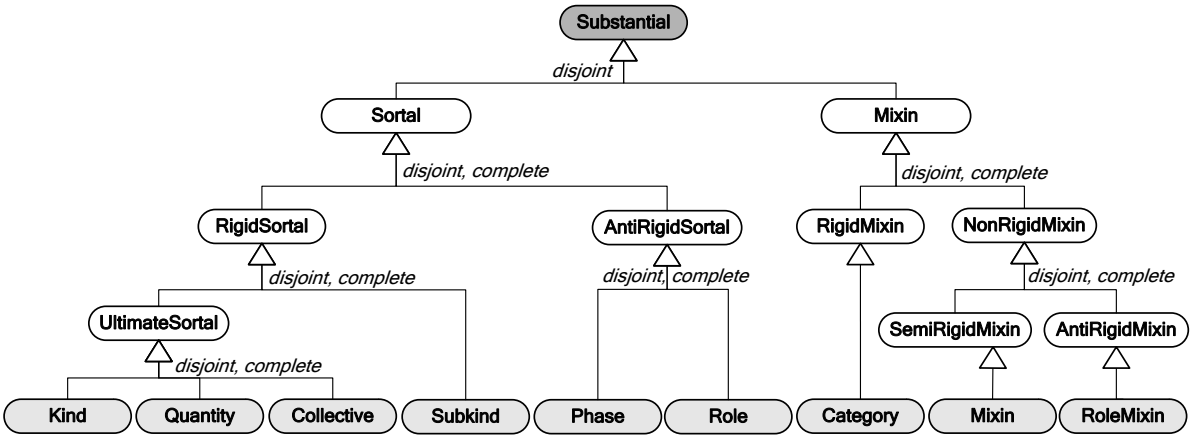


Figura 7. Fragmento da UFO – especialização do tipo de universais *Substantial*.

Na sequência do diagrama, os universais do tipo *Sortal* podem ainda ser classificados como *Rigid Sortal* ou *Anti Rigid Sortal*. Universais do tipo ***Rigid Sortal*** são conceitos rígidos como **Pessoa** e **Empresa**, cujos indivíduos devem instanciá-los enquanto existirem, por exemplo, **João** é necessariamente instância de **Pessoa** enquanto existir. Por outro lado, universais do tipo ***Anti Rigid Sortal*** são conceitos anti-rígidos como **Cliente** e **Adulto**, cujos indivíduos podem eventualmente

instanciá-los enquanto existirem, por exemplo, **João** pode ocasionalmente ser instância desses conceitos.

Os universais do tipo *Rigid Sortal* são ainda classificados como ***Ultimate Sortal*** quando provêem o princípio de identidade aos seus indivíduos, ou como ***Subkind*** quando apenas herdam este princípio. Por exemplo, considerando-se a **impressão digital** como o princípio de identidade provido a toda instância do conceito **Pessoa**, este é do tipo *Ultimate Sortal*, enquanto os conceitos **Homem** e **Mulher** são do tipo *Subkind* pois, além de serem rígidos, especializam o conceito **Pessoa** e, portanto, herdam dele o princípio de identidade. O tipo *Ultimate Sortal*, por sua vez, é sub-divido em **Espécie** (*Kind*), Universal de **Quantidade** (*Quantity Universal*) e Universal de **Coleção** (*Collective Universal*), que categorizam conceitos cujas instâncias são respectivamente complexos funcionais, quantidade e coleção.

Os universais do tipo *Anti-Rigid Sortal*, por sua vez, são classificados como Fase ou Papel. Conceitos do tipo **Fase** (*Phase*) são tais que sua instanciação é determinada por uma propriedade intrínseca do indivíduo. Por exemplo, **João** instancia a fase **Adulto** se sua **idade** (propriedade intrínseca) é maior que 18 anos. Já os conceitos do tipo **Papel** (*Role***)**, como **Cliente Pessoa Física** ou **Esposo**, são relacionalmente dependentes, ou seja, sua instanciação é determinada por uma propriedade relacional do indivíduo. Por exemplo, **João** instancia o papel de **Esposo** se **está casado com** (propriedade relacional) **Maria**.

Por fim, os universais do tipo *Mixin* são classificados como ***Rigid Mixin*** se forem rígidos, ou como ***Non-Rigid Mixin*** caso contrário. O tipo *Rigid Mixin* é também chamado de **Categoria** (*Category*), e um exemplo de instância deste é o conceito **Item Assegurável**, que generaliza conceitos rígidos com diferentes princípios de identidade, como **Pessoa** e **Empresa**. Por sua vez, o tipo *Non-Rigid Mixin* é sub-dividido em ***Semi-Rigid Mixin*** e ***Anti-Rigid Mixin***. O tipo *Semi-Rigid Mixin* é também chamado apenas de ***Mixin***, e um exemplo de instância deste é o conceito **Sentável**, que é ao mesmo tempo rígido para alguns indivíduos, como **cadeira**, e anti-rígido para outros, como **caixote**. Já o tipo *Anti-Rigid Mixin* é também chamado ***Role Mixin***, e um exemplo de instância deste é o conceito **Cliente** que generaliza conceitos anti-rígidos com diferentes princípio de identidade, como **Cliente Pessoa Física** e **Cliente Pessoa Jurídica.**

Os tipos de universais apresentados em cor de fundo cinza claro na Figura 7 são aqueles efetivamente instanciados pelos conceitos de domínio. A Figura 8 apresenta um esquema que

estende aquele da Figura 5 com as especializações de *Substantial* nos níveis meta-conceitual e conceitual (são omitidos alguns elementos explicativos para facilitar a visualização do esquema).

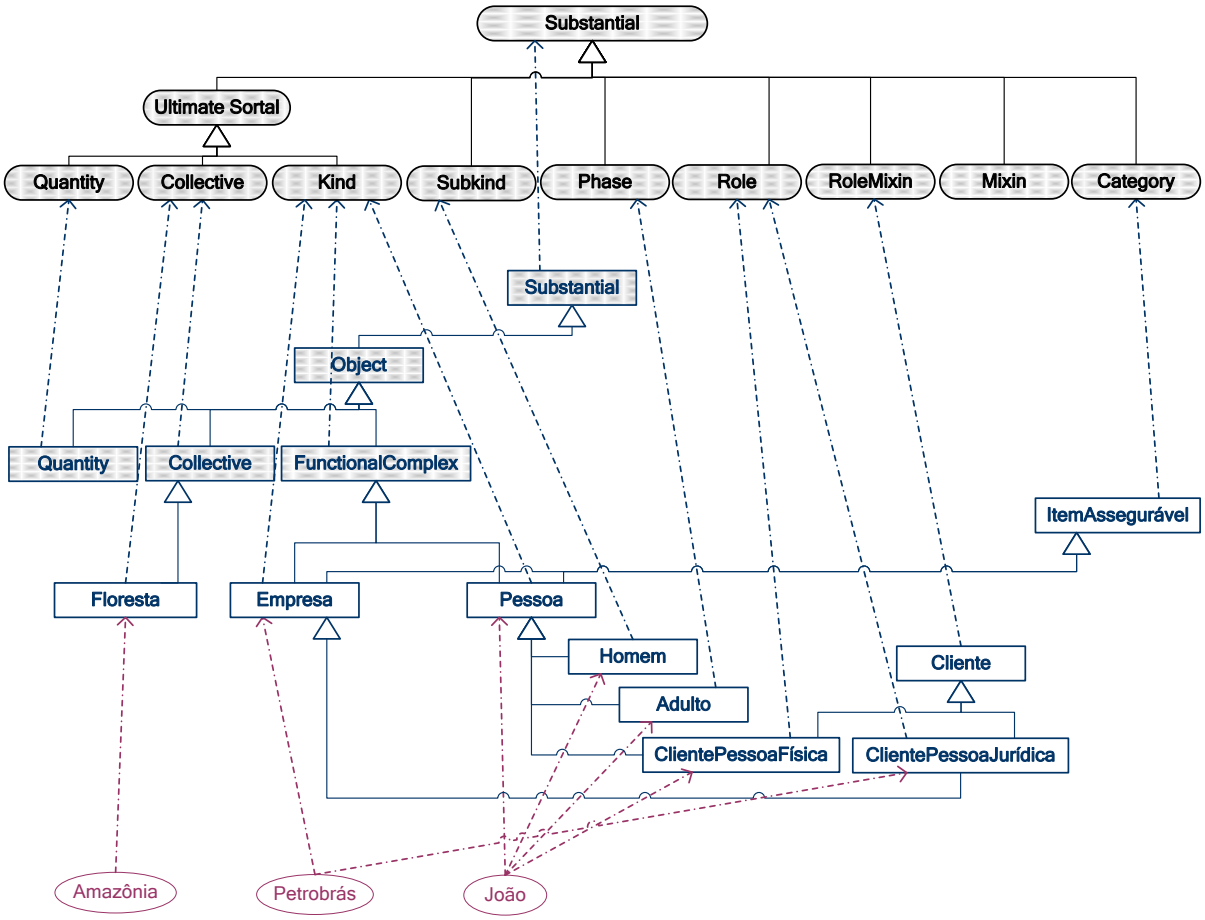


Figura 8. Fragmento da UFO em esquema de níveis – *Substantials*.

Neste exemplo, os conceitos **Pessoa** e **Empresa** instanciam o meta-conceito *Kind*, provendo o princípio de identidade a seus indivíduos – **João** e **Petrobrás**, respectivamente – e especializam o conceito *FunctionalComplex*. Analogamente, o conceito **Floresta** instancia o meta-conceito *Collective Universal*, provendo o princípio de identidade a seus indivíduos – **Amazônia** – e especializa o conceito *Collective*. Por sua vez, o conceito **Homem** é um tipo rígido que especializa **Pessoa** e, portanto, instancia o meta-conceito *SubKind*. Vale observar que os conceitos de domínio que instanciam o meta-conceito *Ultimate Sortal* (**Pessoa**, **Empresa** e **Floresta**) são aqueles que especializam diretamente os conceitos da UFO (*FunctionalComplex* e *Collective*). Os outros sempre especializam ou generalizam estes conceitos do domínio.

Já os conceitos **Adulto** e **Cliente Pessoa Física** são tipos anti-rígidos que especializam **Pessoa**. O primeiro instancia o meta-conceito *Phase*, enquanto o segundo instancia o meta-conceito *Role*. Também o conceito **Cliente Pessoa Jurídica** é um tipo anti-rígido que especializa **Empresa** e instancia o meta-

conceito *Role*. Por sua vez, o conceito **Cliente** é um tipo anti-rígido, relacionalmente dependente e que agrega indivíduos com princípios de identidade diferentes, e, portanto, instancia o meta-conceito *RoleMixin*. Finalmente, o conceito **Item Assegurável** é um tipo rígido que agrega indivíduos com princípios de identidade diferentes, e portanto instancia o meta-conceito *Category*.

i ii - Moments

Os *moments*, por sua vez, são indivíduos essencialmente inerentes a outro indivíduo, como mencionado anteriormente, e que podem ainda ser classificados como intrínsecos (*Instrinsic Moment*) ou relacionais (*Relator*), conforme apresentado no diagrama da Figura 9.

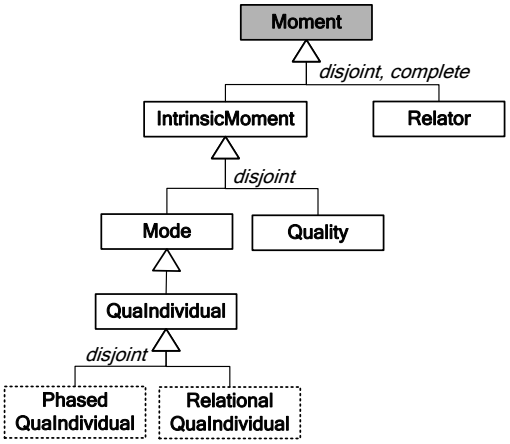


Figura 9. Fragmento da UFO – especialização do tipo de indivíduos *Substantial*.

Os indivíduos do tipo ***Instrinsic Moment*** (GUIZZARDI ET AL., 2006) denotam propriedades intrínsecas do indivíduo portador. Se essa propriedade é mensurável, ou seja, se tem um valor em uma ou mais dimensões de qualidade, ela é chamada de **Qualidade** (*Quality*). Por exemplo, propriedades como **peso** ou **idade** têm um valor numa estrutura unidimensional de números reais não-negativos (a **idade de João** é um *moment* inerente a **João** e possui o valor **20 anos**) enquanto outras, como cor ou sabor têm seu valor numa estrutura multidimensional. Por outro lado, se a propriedade não tem representação num sistema de medida, ela é chamada de **Modo** (*Mode*), por exemplo, a **dor de cabeça de João**.

Um tipo especial de *Mode*, chamado de **Indivíduo-Qua** (*Qua-Individual*) (MASOLO, CLAUDIO ET AL., 2005; GUIZZARDI, 2006), refere-se a um *moment* que representa o conjunto de características que um indivíduo adquire num certo contexto. Esse nome vem justamente da idéia de se considerar um indivíduo apenas a respeito de determinados aspectos. Se o contexto é dado pela instanciação de uma relação, o *moment* é dito do tipo **Indivíduo-Qua Relacional** (*Relational*

Qua-Individual). São exemplos o conjunto de características que **João** adquire quando da instanciação da relação de **estar casado com Maria**, a saber, **JoãoQuaEsposoDeMaria**, bem como sua contraparte mutuamente dependente, o modo análogo de **Maria**, a saber, **MariaQuaEsposaDeJoão**.

Neste ponto, faz-se uma adaptação à definição original da UFO, em que um *qua-individual* é tido apenas como relacional, acrescentando-se um tipo chamado **Indivíduo-Qua Fasal** (*Phased Qua-Individual*). Este tipo refere-se ao o conjunto de características que um indivíduo possui quando da instanciação de uma fase. Por exemplo, **JoãoQuaAdulto** é um *moment* inerente a **João** quando este instancia a fase **Adulto**. Os tipos *PhasedQuaIndividual* e *RelationalQuaIndividual* estão pontilhados no diagrama da Figura 9 para denotar que estes conceitos não existem originalmente na UFO.

Com relação aos *moments* relacionais, classificados pelo tipo ***Relator*** (GUIZZARDI; WAGNER, 2008), estes são indivíduos interventores ou mediadores, ou seja, indivíduos que mediam outros tornando verdadeira uma certa relação entre eles. Por exemplo, o *moment* **casamento de João e Maria** media os indivíduos **João** e **Maria**, tornando verdadeira a relação de **estar casado com** entre eles. Além disso, um indivíduo *relator* é composto essencialmente dos indivíduos-qua relacionais que são inerentes aos indivíduos mediados quando da instanciação da relação que ele representa, e é também inerente à soma mereológica desses indivíduos-qua. No exemplo supracitado, o *moment* **casamento de João e Maria** é composto dos indivíduos-qua **JoãoQuaEsposoDeMaria** e **MariaQuaEsposaDeJoão**, e é inerente à soma mereológica destes.

As mesmas distinções se refletem na especialização do tipo de universais ***Moment***, exceto pelos indivíduos-qua que, geralmente, não são explicitamente representados dentre os universais específicos de domínio, uma vez que indivíduos-qua são *moments* surgem inerentes a seus portadores quando estes instanciam certos conceitos e/ou relações. Assim, o tipo ***Moment Universal*** é sub-dividido em *Intrinsic Moment Universal*, *Mode Universal*, *Quality Universal* e *Relator Universal*, que classificam os conceitos que agrupam os *moments* do tipo correspondente. Por exemplo, o conceito **Casamento** é um universal do tipo *Relator Universal* cujas instâncias são do tipo *Relator*, como **o casamento de João e Maria**. O fragmento de diagrama da UFO apresentado na Figura 10 mostra a especialização do tipo *Moment Universal*. Os tipos de universais com cor de fundo cinza claro são aqueles diretamente instanciados pelos conceitos de domínio.

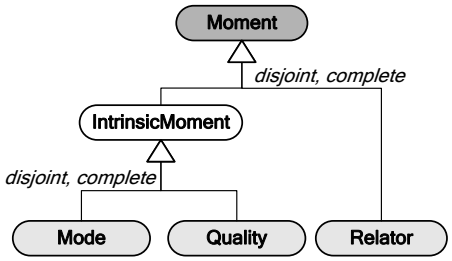


Figura 10. Fragmento da UFO – especialização do tipo de universais *Moment*.

A Figura 11 apresenta um esquema que estende aquele da Figura 5 com as especializações de *Moment* nos níveis meta-conceitual e conceitual. Para simplificar o esquema, alguns elementos são omitidos. Os conceitos **Dor de Cabeça** e **Idade** referem-se a indivíduos que denotam propriedades intrínsecas do seu portador – **dor de cabeça de João** e **idade de João**. Enquanto o conceito **Idade**, cujas instâncias possuem um valor numa dimensão de qualidade, instancia o meta-conceito *Quality Universal* e especializa o conceito *Quality*, o conceito **Dor de Cabeça** instancia o meta-conceito *Mode Universal* e especializa o conceito *Mode*. Já o conceito **Casamento** denota uma propriedade relacional dos indivíduos por ele mediados, instanciando, portanto, o meta-conceito *Relator Universal* e especializa o conceito *Relator*.

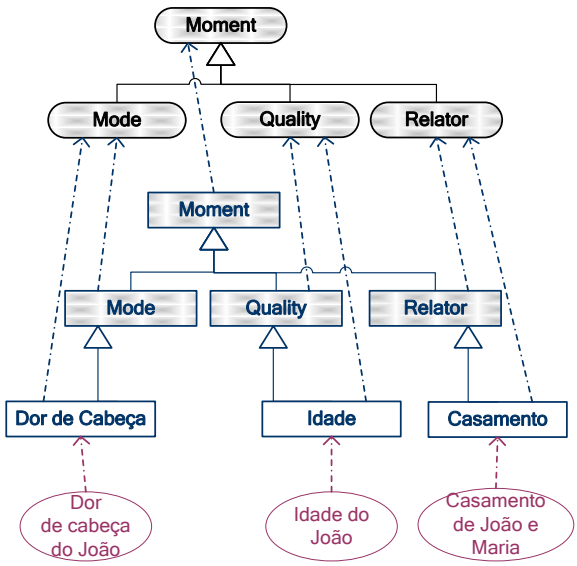


Figura 11. Fragmento da UFO em esquema de níveis – *Moments*.

II - As Categorias de Universais de Relação e Categorias de Instâncias de Relações

Retomando a distinção inicial dos tipos de universais (mónadicos e de relação), o tipo ***Relation Universal*** é a categoria geral que se aplica aos universais de relação, também chamados apenas de relações (assim como universais monádicos são chamados apenas de conceitos). Esta categoria é sub-dividida segundo a aplicação da distinção ontológica básica entre relações (ver Quadro 3 da

seção 2.1.1), a saber, Formal e Material (GUIZZARDI; WAGNER, 2008), como pode ser visualizado na Figura 12. Os tipos de universais representados com cor de fundo cinza claro são aqueles diretamente instanciados pelas relações no nível conceitual.

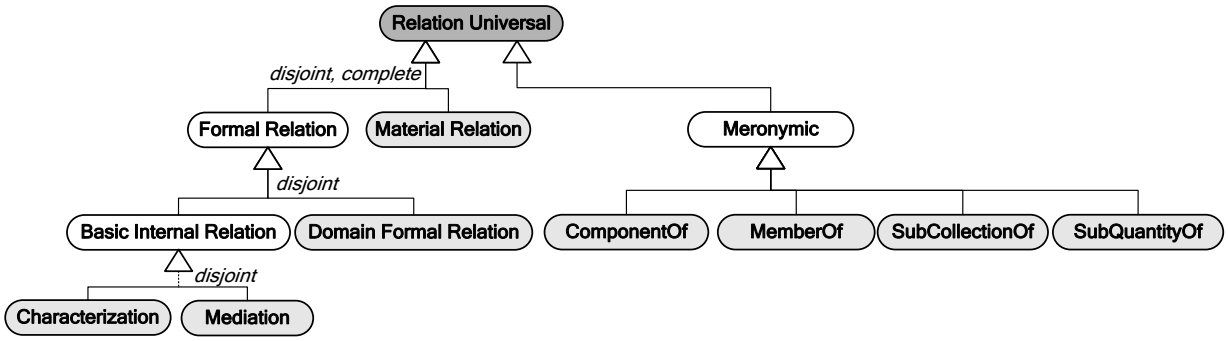


Figura 12. Fragmento da UFO – tipos de Relação.

O tipo ***Material Relation*** se aplica às relações que dependem de algum interventor para valer, a saber, um indivíduo do tipo *Relator*. Por exemplo, a relação do tipo material **casado com** entre **João** e **Maria** vale enquanto existir o *relator* **Casamento de João e Maria**. Contrariamente, as relações do tipo ***Formal Relation*** valem pela simples existência dos indivíduos relacionados. Por exemplo, a relação **parte de** entre **João** e **seu Cérebro** vale sempre que ambos existirem.

As relações do tipo **Formal** podem ser ainda classificadas como **Relação Básica Interna** (*Basic Internal Relation*) ou **Relação Formal de Domínio** (*Domain Formal Relation*). O tipo *Basic Internal Relation* aplica-se a relações formais internas ditas de dependência existencial que têm representação entre as categorias de Indivíduos da UFO. Este tipo se sub-divide em **Caracterização** (*Characterization*), que se aplica à relação de inerência (*inheresIn*) que define os indivíduos do tipo *Moment*, e **Mediação** (*Mediation*), que se aplica à relação de mediação (*mediates*) que define os indivíduos do tipo *Relator*. Por sua vez, o tipo *Domain Formal Relation* se aplica às relações formais que são específicas de domínio e que, por isso, não são representadas entre os tipos de indivíduos da UFO, mas entre os conceitos específicos de domínio, assim como as do tipo *Material Relation*.

Por fim, o tipo de relação **Meronímica** (*Meronymic*) refere-se às relações todo-parte (*partOf*) que ocorrem entre os subtipos de *Object*. Em particular, há duas meta-propriedades associadas a este tipo que interessam neste trabalho: é-essencial (*isEssential*) e é-inseparável (*isInseparable*). O primeiro implica que o todo é existencialmente dependente da parte, e o segundo implica no caso inverso. Este tipo é ainda subdividido em Componente-de (*ComponentOf*), Membro-de (*MemberOf*), Sub-Coleção-de(*SubcollectionOf*) e Sub-Quantidade-de (*SubQuantityOf*), cujas relações correspondentes são explicadas mais adiante no texto.

Enquanto o fragmento de diagrama da UFO apresentado na Figura 13a mostra a representação da relações do tipo *Characterization* e *Mediation* entre os tipos de indivíduos, o da Figura 13b mostra a representação da relações do tipo *Meronymic*.

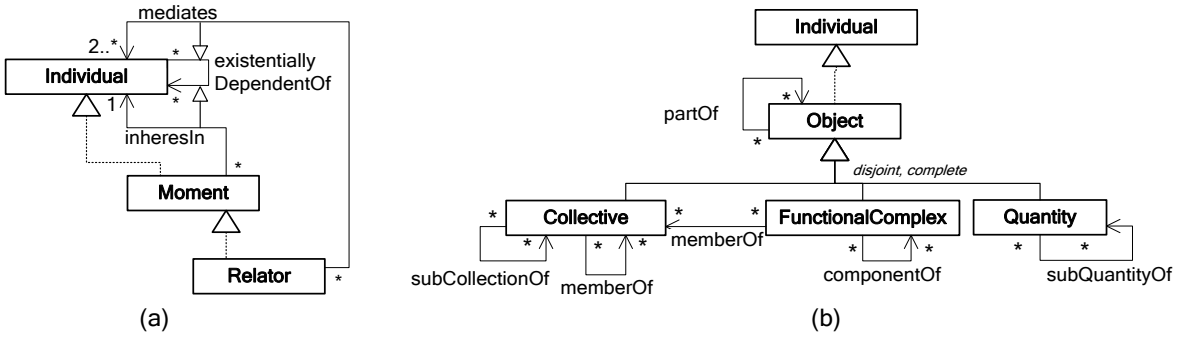


Figura 13. Fragmentos da UFO – relações entre tipos de indivíduos.

A relação de dependência existencial (*existentiallyDependentOf*) entre indivíduos é uma relação irreflexiva tal que sempre que o indivíduo dependente existir, então o outro indivíduo do qual aquele depende também existe. A relação de inerência (*inheresIn*) entre *moments* e indivíduos é um tipo especial de dependência existencial que define *Moment* como o tipo dos indivíduos que são inerentes a exatamente um outro indivíduo. Também a relação de mediação (*mediates*) entre *relators* e indivíduos é um tipo especial de dependência existencial que define *Relator* como o tipo dos indivíduos que mediam dois ou mais indivíduos.

Por sua vez, a relação parte-de (*partOf*), que instancia o tipo *Meronymic*, ocorre entre indivíduos do tipo *Object*. Ela é especializada[6] conforme os sub-tipos da classe *Object* que são ligados pela relação, como segue: (i) **componente-de** (*componentOf*) vale entre complexos funcionais; (ii) **membro-de** (*memberOf*) vale entre complexos funcionais e coleções, ou coleções entre si; (iii) **sub-coleção-de** (*subColectionOf*) vale entre coleções; e (iv) **sub-quantidade-de** (*subQuantityOf*) vale entre quantidades.

Enquanto a Figura 14 apresenta um esquema, semelhante ao da Figura 3, que exemplifica as relações do tipo *Material*, *Characterization* e *Mediation*, o esquema da 15 exemplifica as relações do tipo *ComponentOf*. A relação **inerente-a** entre **Pessoa** e **Dor de Cabeça**, que instancia o tipo *Characterization* e especializa a relação *inheresIn*, é instanciada no nível de indivíduos entre **João** e **Dor de Cabeça de João**. Analogamente, a **media** entre **Pessoa** e **Casamento**, que instancia o tipo *Mediation* e especializa a *mediates*, instanciada no nível de indivíduos entre **João** e **Casamento**, e. Por sua vez, a relação **casado com** entre

[6] a relação de especialização entre as relações é omitida para facilitar a visualização do diagrama.

indivíduos do tipo **Pessoa** instancia o tipo *Material* pois depende de um interventor para valer entre indivíduos. Esta relação é instanciada no nível de indivíduos entre **João** e **Maria**, uma vez que existe um interventor, o *relator* **Casamento de João e Maria**, que os media.

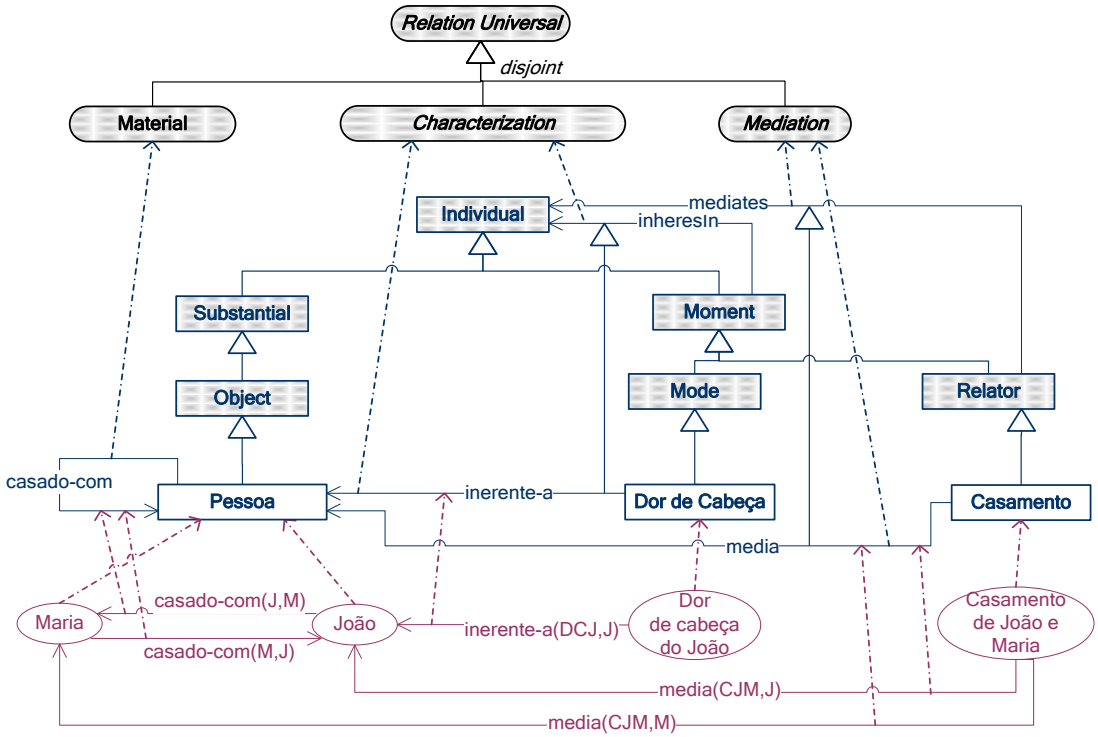


Figura 14. Fragmento da UFO em esquema de níveis – Relações.

A relação **componente-de** entre **Pessoa** e **Coração**, que instancia o tipo *ComponentOf Universal* e especializa a relação *componentOf*, é instanciada no nível de indivíduos entre os complexos funcionais **João** e **Coração de João**

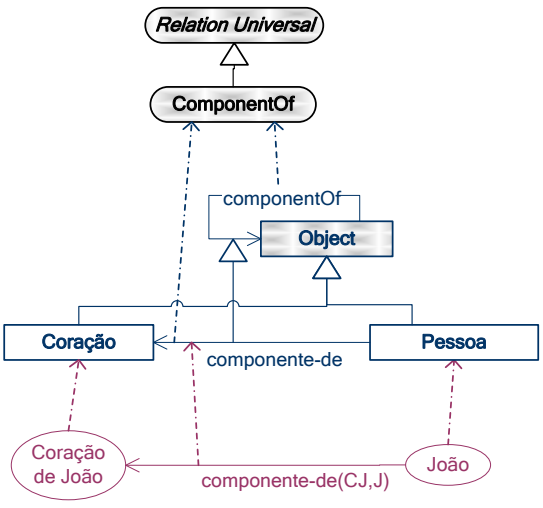


Figura 15. Fragmento da UFO em esquema de níveis – Relações Meronímicas.

2.2. NÍVEIS DE LINGUAGEM

Esta seção apresenta de forma sucinta alguns níveis de linguagem relevantes a este trabalho e suas principais características, conforme resumido na Tabela 1. Esta classificação foi proposta por Guarino em 1994 (GUARINO, NICOLA, 1994), revisitando a classificação proposta em 1979 por Brachman (BRACHMAN, 1979 apud GUARINO, 1994).

Tabela 1. Principais características dos níveis de linguagem Lógico, Epistemlógico e Ontológico.

Níveis	Construtos Primitivos	Principal Característica	Interpretação
Lógico	Predicados	Formalização	Arbitrária
Epistemológico	Relações Estruturais (Conceitos e Papéis)	Estrutura	Arbitrária
Ontológico	Relações Estruturais (satisfazem postulados de significado)	Significado	Restrita

Brachman defende que o conhecimento consiste de proposições, cujas estruturas formais dão origem a novos conhecimentos. Segundo o autor, se por um lado as linguagens de nível lógico provêem construtos para formalização do conhecimento que permitem o raciocínio formal, por outro lado lhes falta fundamentação cognitiva que favoreça a representação do conhecimento de forma estruturada. Assim, Brachman propõe o nível epistemológico, cujas linguagens são tais que permitam não só formalizar o conhecimento, mas estruturá-lo permitindo favorecer o raciocínio formal e a derivação de novos conhecimentos (GUARINO, NICOLA, 2009; GUIZZARDI, 2007).

Por sua vez, Guarino argumenta que em ambos os níveis lógico e epistemológico a interpretação de mundo real da teoria lógica é completamente arbitrária. Em particular, o nível epistemológico, apesar de acrescentar um significado estrutural ao conhecimento, não está focado na representação formal. Assim, o autor propõe a existência de nível ontológico em que "as primitivas de conhecimento satisfaçam postulados formais de significado, que restringem a interpretação de uma teoria lógica com base em distinções ontológicas formais" (GUARINO, NICOLA, 2009, p. 6).

Enquanto exemplos de linguagens de nível epistemológico são diversos, como UML e DL/OWL, entre outros, exemplos de linguagens de nível ontológico são raros. Segundo Guarino (2009), por não haver um consenso geral sobre ter as distinções ontológicas embutidas em linguagens de representação, "importantes questões a respeito de diferentes suposições ontológicas subjacentes ao nosso uso de termos têm sido simplesmente deixadas de lado enquanto busca-se simplificação

lógica e tratabilidade computacional" (GUARINO, NICOLA, 2009, p. 2). Assim, decisões de modelagem são deixadas por conta do engenheiro do conhecimento e permanecem implícitas em sua mente, prejudicando a precisão da comunicação e compartilhamento da informação. Ainda segundo o autor, uma proposta concreta nesta direção é feita por Guizzardi (2005), em que o metamodelo do diagrama de classes da UML é estendido para incorporar as distinções ontológicas definidas na ontologia de fundamentação UFO.

Se por um lado linguagens de nível ontológico são então mais apropriadas para representação clara, precisa e não ambígua do conhecimento acerca de domínio, por outro lado elas tendem a não ser computacionalmente tratáveis. Assim, uma vez que se deseja realizar raciocínio automatizado sobre uma ontologia, torna-se então necessário utilizar linguagens de nível lógico/epistemológico que tenham tal objetivo. Diante deste conflito entre focar na qualidade da representação ou em questões computacionais, Guizzardi (2007, 2010a) defende que é necessária uma abordagem de Engenharia de Ontologia em que essas duas classes de linguagem têm seu papel. Esta abordagem é explicada na seção a seguir.

2.3. ENGENHARIA DE ONTOLOGIA

A abordagem de Engenharia de Ontologia proposta por Guizzardi (2007), alinhada com a posição defendida por Masolo et al. (2003 apud GUIZZARDI, 2007) , aponta a necessidade de duas classes de linguagens que permitam produzir tanto uma **ontologia de referência de domínio** (*domain reference ontology*), visando capturar de forma clara, concisa e não ambígua os conceitos do domínio, quanto uma **ontologia leve** (*lightweight ontology*), para atender a requisitos computacionais.

Esta abordagem segue métodos análogos aos de engenharia nas disciplinas de Engenharia de Software e de Sistemas de Informação, em que existe uma clara distinção entre Modelagem Conceitual, Projeto e Implementação. Assim, uma abordagem teoricamente consistente de Engenharia de Ontologia consiste de três fases: Análise, Projeto e Implementação. Na fase de **Análise** utiliza-se uma linguagem de nível ontológico para criar a **ontologia de referência de domínio**, focada na adequação da representação, que pode ser usada por humanos em tarefas como comunicação, análise de domínio, negociação de significados, estabelecimento de consensos e solução de problemas.

Uma vez que uma conceituação comum é estabelecida como referência, na fase de **Implementação** utiliza-se linguagens de nível lógico/epistemológico para criar **ontologias leves**, ou seja, versões da ontologia de referência com foco em garantir propriedades computacionais desejáveis. Contudo, é necessária ainda uma fase de **Projeto** para ligar a análise à implementação de ontologias. Nesta fase, leva-se em consideração, por exemplo, a diferença de expressividade entre as linguagens usadas nas outras fases, ou ainda, a questão de como produzir ontologias leves que maximizem requisitos não-funcionais. A Figura 16 ilustra esta abordagem.

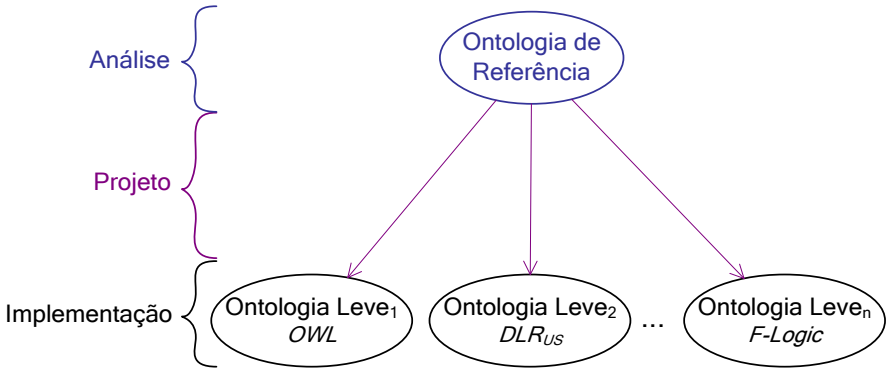


Figura 16. Abordagem de Engenharia de Ontologia.

2.4. LINGUAGENS PARA REPRESENTAÇÃO DE ONTOLOGIAS

As linguagens para representação de ontologias adotadas neste trabalho são a de nível ontológico OntoUML, apresentada na seção 2.4.1, e a de nível epistemológico OWL, apresentada na seção 2.4.2.

2.4.1. OntoUML – *Ontological Unified Modeling Language*

OntoUML é o nome dado à versão ontologicamente bem-fundamentada (do diagrama de classes) da UML 2.0 (*Unified Modeling Language*), proposta por Guizzardi (2005). O autor mostra que falta uma definição precisa da semântica formal da linguagem UML, e estende o metamodelo desta para ser isomórfico à UFO-A, tornando-o ontologicamente consistente. A UML tem mecanismos de extensão que permitem modificar os elementos da linguagem de tal forma que um conjunto coerente de tais extensões constitui um perfil UML. A OntoUML é, portanto, um perfil UML composto por um conjunto de estereótipos que representam as categorias ontológicas dos tipos de universais propostos na UFO-A, bem como por restrições formais que refletem a axiomatização da UFO de tal modo que se restringe o conjunto de modelos gramaticalmente válidos em OntoUML àqueles que representam situações admissíveis segundo a

teoria da UFO (GUIZZARDI, 2007). Assim, a explicação da aplicação dos estereótipos é idêntica à aplicação dos tipos de universais correspondentes da UFO.

A Figura 17 e a Figura 18 mostram fragmentos do metamodelo dessa linguagem, cujos elementos-folha são os estereótipos da linguagem relevantes no contexto deste trabalho. A primeira apresenta fragmentos do metamodelo da OntoUML em que o elemento de representação *Class* da UML é especializado conforme os tipos de Universais Monádicos da UFO.

Também o elemento de representação *Relationship* da UML é especializado conforme os tipos de Universais de Relação da UFO, vide Figura 18. As relações do tipo formais básicas e meronímicas especializam um tipo especial de *Relationship*, a saber, *Directed Binary Relationship* (Relação Binária e Direcionada), enquanto os outros tipos especializam *Association*, que refere-se a relações enárias. Em particular, o estereótipo *Formal Association* refere-se ao tipo de universal *Domain Formal Relation* da UFO.

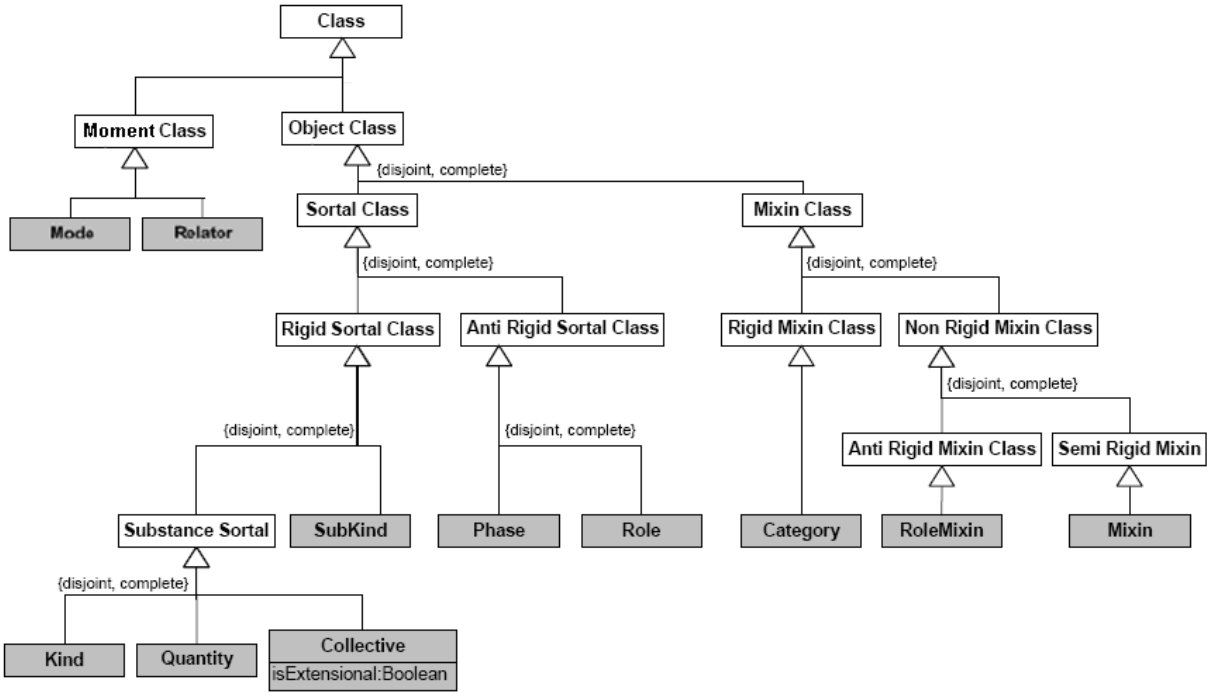


Figura 17. Fragmento do metamodelo da linguagem OntoUML para o tipo *Class*.
Fonte: Guizzardi (2005, p. 316) adaptado

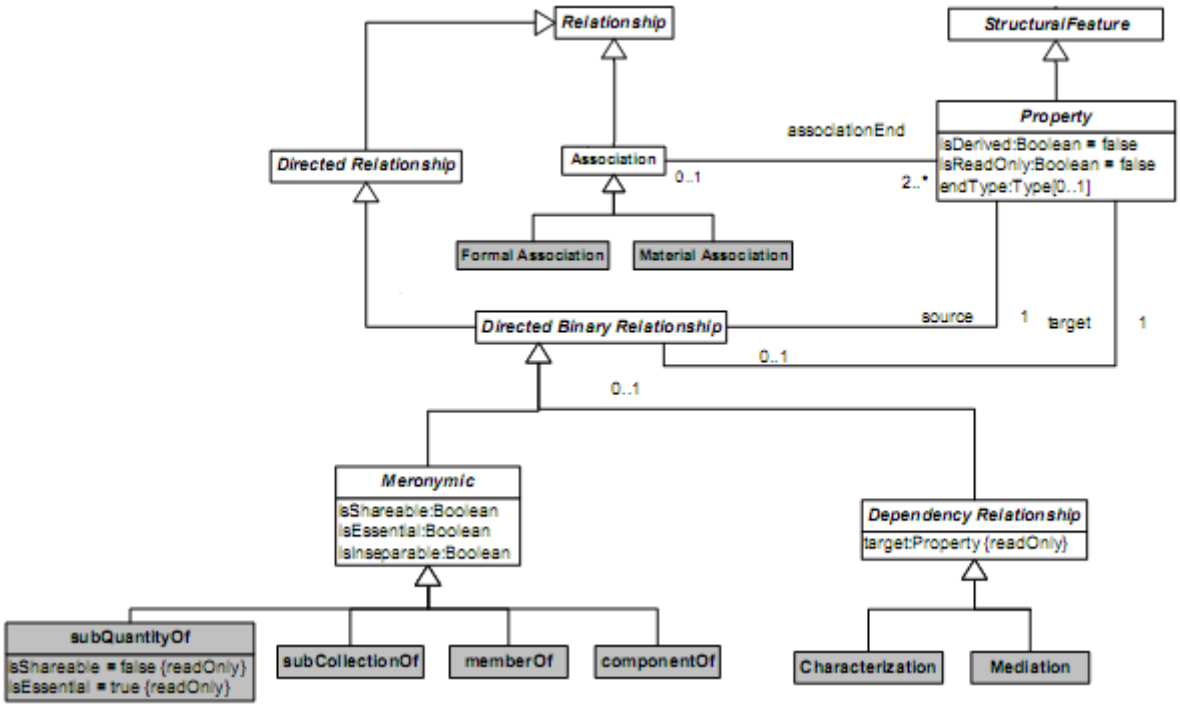


Figura 18. Fragmento do metamodelo da linguagem OntoUML para o tipo *Relationship*.
Fonte: Guizzardi (2005, p. 334) adaptado

2.4.2. OWL – Web Ontology Language

A linguagem OWL (*Web Ontology Language*) é um padrão de fato recomendado pela W3C (*World Wide Web Consortium*) para representação de ontologias na *Web* Semântica. Ela foi projetada para representar categorias de objetos e as relações entre eles, além de informações sobre os próprios objetos. (HORROCKS ET AL., 2003).

A OWL foi proposta com o intuito de atender às seguintes restrições: (i) manter a compatibilidade com padrão RDF (*Resource Description Framework*) para representação de informação na *Web*, estendendo, porém, a capacidade de expressar o conhecimento dito "ontológico"; (ii) ter sintaxe e semântica bem definidas, bem como um poder de expressividade que mantenha propriedades computacionais desejáveis. (ANTONIOU; HARMELEN, 2003).

Esta linguagem tem sido amplamente usada para representação de ontologias em diversos domínios como medicina, biologia, geografia, astronomia. Entretanto, sua ampla utilização apontou várias limitações, que se buscou resolver com a proposta de uma extensão da linguagem original, chamada de OWL 1, para uma nova versão chamada OWL 2 (GRAU ET AL., 2008). Neste trabalho é utilizada a linguagem OWL 2 (referida apenas como OWL no restante do texto).

Sintaxe e Semântica

A Figura 19 apresenta um diagrama estilo UML, que é usado no decorrer deste trabalho para representar modelos OWL, visando facilitar o entendimento destes. Tal estrutura cobre apenas parcialmente a capacidade de representação de OWL, mas é suficiente para as discussões conduzidas aqui. Este exemplo é representado a seguir na sintaxe funcional de OWL (que é definida em seguida), e sua explicação visa ilustrar tanto a utilização do diagrama quanto a interpretação dos construtos nele contemplados.

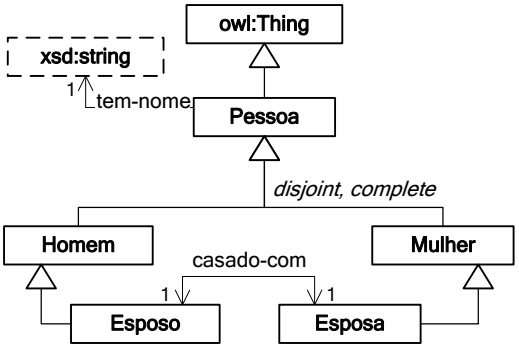


Figura 19. Exemplo de esquema estilo UML para representação de OWL.

Classes: Pessoa, Homem, Mulher, Esposo, Esposa
Propriedades de Objeto: casado-com
Propriedades de Dados: tem-nome
(1) SubClassOf(Homem Pessoa)
(2) SubClassOf(Mulher Pessoa)
(3) DisjointClasses(Mulher Homem)
(4) EquivalentClasses(Pessoa
ObjectUnionOf(Mulher Homem))
(5) SubClassOf(Pessoa
DataExactCardinality(1 tem-nome xsd:string))
(6) SubClassOf(Esposo Homem)
(7) EquivalentClasses(Esposo
ObjectSomeValuesFrom(casado-com Esposa))
(8) SubClassOf(Esposa Mulher)
(9) EquivalentClasses(Esposa
ObjectSomeValuesFrom(casado-com Esposo))
(10) FunctionalObjectProperty(casado-com)
(11) SymmetricObjectProperty(casado-com)
(12) ObjectPropertyDomain(casado-com
ObjectUnionOf(Esposo Esposa))
(13) ObjectPropertyRange(casado-com
ObjectUnionOf(Esposo Esposa))
(14) DataPropertyDomain(tem-nome Pessoa)
(15) DataPropertyRange(tem-nome xsd:string)

A principal distinção neste domínio de exemplo é que os indivíduos classificados como pessoa são homens ou são mulheres. Esta distinção é representada no diagrama pelo conjunto de generalização (*generalization set*) que contém as classes **Homem** e **Mulher**, que especializam a classe **Pessoa**, rotulado com os termos *disjoint* e *complete*. As especializações, representadas nas linhas 1 e 2 do modelo OWL, significam que a interpretação da classe **Homem**, bem como de **Mulher**, está contida na interpretação da classe **Pessoa**, ou seja, todo indivíduo que seja homem ou mulher é pessoa.

O termo *disjoint*, representado na linha 3, declara que as classes **Homem** e **Mulher** são disjuntas, e significa que a interseção da interpretação destes é vazia, ou seja, um homem não pode ser mulher e vice-versa. O termo *complete*, representado na linha 4, declara que a classe **Pessoa** é particionada de forma completa pelas classes **Homem** e **Mulher**, e significa que a interpretação da classe **Pessoa** é equivalente à interpretação da união das classes **Homem** e **Mulher**, ou seja, toda pessoa é um homem ou uma mulher. Além disso, um Homem pode ser Esposo, e uma Mulher pode ser Esposa. A explicação dessas especializações é semelhante à outra já realizada neste parágrafo.

Também é verdade que toda pessoa tem exatamente um nome, que é do tipo *string*. Isto é representado no diagrama pela relação nomeada **tem-nome** entre a classe **Pessoa** e o tipo de dado **xsd:String** com uma restrição de cardinalidade igual a 1. No modelo OWL, isto se refere à linha 5, e significa que a interpretação da classe **Pessoa** é equivalente à interpretação da expressão de classe que corresponde a todos os indivíduos que instanciam a propriedade de dados **tem-nome** com exatamente um elemento do domínio de dados do tipo *string*.

Ainda as linhas 14 e 15 restringem respectivamente o domínio e a imagem desta propriedade. A primeira significa que, para cada instância da propriedade **tem-nome**, o elemento do domínio da propriedade pertence à interpretação da classe **Pessoa**, por exemplo, se **tem-nome(João, "João da Silva")** então a interpretação de **João** pertence à união da interpretação da classe **Pessoa**. Analogamente, a segunda significa que, para cada instância da propriedade **tem-nome**, o elemento da imagem pertence à interpretação do tipo de dados **String**, por exemplo, se **tem-nome(João, "João da Silva")** então **"João da Silva"** tem que ser uma *string*.

Um indivíduo do tipo **Esposo** é necessariamente **casado com** exatamente um do tipo **Esposa**, e vice-versa. Essa situação é representada no diagrama pela relação bidirecional **casado-com** entre as classes **Esposo** e **Esposa**. No modelo OWL, isto se refere às linhas 7, 9 e 10 a 13. As duas primeiras significam que a interpretação da classe **Esposo** (**Esposa**) é equivalente à interpretação da expressão de classe que corresponde a todos os indivíduos que instanciam a propriedade de objetos **casado-com** com algum elemento do domínio que pertença à interpretação da classe **Esposa** (**Esposo**). Também a linha 10 significa que se a propriedade **casado-com** for instanciada duas vezes para um mesmo indivíduo, por exemplo, **casado-com(João, Maria)** e **casado-com(João, Ana)**, então **Maria** e **Ana** são a mesma pessoa, ou seja, têm a mesma interpretação.

Ainda a linha 11 significa que sempre a propriedade **casado-com** for instanciada para um par ordenado, ela vale também para o seu par simétrico, por exemplo, se **casado-com(João, Maria)** então **casado-com(Maria, João)**. Finalmente, as linhas 12 e 13 restringem respectivamente domínio e imagem da propriedade **casado-com**, e, sendo semelhantes, significam que, para cada instância da propriedade **casado-com**, o elemento do domínio (ou imagem) da propriedade pertence à interpretação da expressão de classe que a todos os indivíduos que pertencem à união da interpretação das classes **Esposo** e **Esposa**, por exemplo, se **casado-com(João, Maria)** então a interpretação de **João** pertence à união da interpretação das classes **Esposo** e **Esposa**, bem como a de **Maria**.

A sintaxe e a semântica formal dos construtos de OWL utilizados neste trabalho, são apresentadas na Tabela 2. Adota-se a sintaxe funcional conforme definido por Motik et al. (2009). A teoria semântica, correspondente à da lógica de descrição $\mathcal{SROIQ}$(D) (HORROCKS ET AL., 2006), é apresentada conforme definido por Motik, Patel-Schneider e Grau (2009), porém de forma simplificada.

A semântica de OWL é definida usando uma semântica de modelos (*Model-Theoretic Semantics*) ou a chamada Semântica Tarskiana (*Tarskian-Semantics*), em homenagem ao logicista polonês Alfred Tarski. Nesta teoria, um chamado *modelo* inclui um conjunto (chamado de domínio ou universo) $\Delta.^{\mathcal{I}}$ e de uma função de interpretação $.^{\mathcal{I}}$. O domínio $\Delta.^{\mathcal{I}}$ é o conjunto de elementos para uma certa interpretação, e a função de interpretação $.^{\mathcal{I}}$ é um mapeamento de nomes de indivíduos em elementos do domínio, nomes de classes em subconjuntos do domínio, e nomes de propriedades de indivíduos em conjuntos de pares ordenados de elementos do domínio. A função de interpretação pode ser estendida para expressões de classes ou propriedades Utiliza-se um domínio à parte $\Delta.^{\mathcal{D}}$ para os valores de tipos de dados, bem como uma função de interpretação $.^{\mathcal{D}}$ que mapeia um nome de tipo de dados em elementos deste domínio. Enfim, uma interpretação satisfaz uma ontologia se e somente se ela satisfaz cada axioma e fato da ontologia; e uma ontologia é consistente se e somente se ela é satisfeita por ao menos uma interpretação.

Tabela 2. Sintaxe Funcional e Semântica de alguns contrutos OWL.

Sintaxe	Semântica
DECLARAÇÕES	
Declaration(Class(C **))**	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Declaration(ObjectProperty(OP **))**	$OP^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Declaration(Datatype(DT **))**	$DT^{\mathcal{D}} \subseteq \Delta^{\mathcal{D}}$

Declaration(DataProperty(DP) **)**	$DP^I \subseteq \Delta^I \times \Delta^D$
EXPRESSÕES DE CLASSE	
ObjectComplementOf (C) ,	$\{x \mid x \in \Delta^I \setminus C^I\}$
ObjectIntersectionOf (C_1 .. C_n) ,	$\{x \mid x \in C_1^I \cap .. \cap C_n^I\}$
ObjectUnionOf (C_1 .. C_n) ,	$\{x \mid x \in C_1^I \cup .. \cup C_n^I\}$
ObjectSomeValuesFrom (OP C)	$\{x \mid \exists y : (x,y) \in OP^I \wedge y \in C^I\}$
ObjectAllValuesFrom (OP C)	$\{x \mid \forall y : (x,y) \in OP^I \rightarrow y \in C^I\}$
ObjectMinCardinality (n OP C)	$\{x \mid \#\{y \mid (x,y) \in OP^I \wedge y \in C^I\} \geq n\}$
ObjectMaxCardinality (n OP C)	$\{x \mid \#\{y \mid (x,y) \in OP^I \wedge y \in C^I\} \leq n\}$
ObjectExactCardinality (n OP C)	$\{x \mid \#\{y \mid (x,y) \in OP^I \wedge y \in C^I\} = n\}$
DataSomeValuesFrom (DP DT)	$\{x \mid \exists y : (x,y) \in DP^I \wedge y \in DT^D\}$
DataMinCardinality (n DP DT)	$\{x \mid \#\{y \mid (x,y) \in DP^I \wedge y \in DT^D\} \geq n\}$
DataMaxCardinality (n DP DT)	$\{x \mid \#\{y \mid (x,y) \in DP^I \wedge y \in DT^D\} \leq n\}$
EXPRESSÕES DE PROPRIEDADES DE OBJETO	
InverseObjectProperty (OP)	$\{(x,y) \mid (y,x) \in OP^I\}$
AXIOMAS DE EXPRESSÕES DE CLASSE	
SubClassOf (SC C)	$SC^I \subseteq C^I$
EquivalentClasses (C_1 C_2)	$C_1^I = C_2^I$
DisjointClasses (C_1 .. C_n)	$C_j^I \cap C_k^I = \varnothing, 1 \leq j < k \leq n$
ObjectPropertyDomain (OP C)	$\forall x,y\ (x,y) \in OP^I \rightarrow x \in C^I$
ObjectPropertyRange (OP C)	$\forall x,y\ (x,y) \in OP^I \rightarrow y \in C^I$
AXIOMAS DE EXPRESSÕES DE PROPRIEDADES DE OBJETO	
SubObjectPropertyOf (SOP OP)	$SOP^I \subseteq OP^I$
SubObjectPropertyOf(**ObjectPropertyChain** (OP_1 .. OP_n) OP)	$\forall y_0 .. y_n\ (y_0, y_1) \in OP_1^I \wedge .. \wedge (y_{n-1}, y_n) \in OP_n^I \rightarrow (y_0, y_n) \in OP^I$
FunctionalObjectProperty (OP)	$\forall x,y_1,y_2\ (x,y_1) \in OP^I \wedge (x,y_2) \in OP^I \rightarrow y_1 = y_2$
IrreflexiveObjectProperty (OP)	$\forall x,y\ (x,y) \in OP^I \rightarrow x \neq y$
SymmetricObjectProperty (OP)	$\forall x,y\ (x,y) \in OP^I \rightarrow (y,x) \in OP^I$
AXIOMAS DE EXPRESSÕES DE PROPRIEDADES DE DADOS	
SubDataPropertyOf (SDP DP)	$SDP^I \subseteq DP^I$
DataPropertyDomain (DP C)	$\forall x,y\ (x,y) \in DP^I \rightarrow x \in C^I$
DataPropertyRange (DP DT)	$\forall x,y\ (x,y) \in DP^I \rightarrow y \in DT^D$
FunctionalDataProperty (DP)	$\forall x,y_1,y_2\ (x,y_1) \in DP^I \wedge (x,y_2) \in DP^I \rightarrow y_1 = y_2$

Ademais, por ter a teoria semântica baseada em DL, algumas importantes características desta linguagem lógica, incorporadas à semântica de OWL, devem ainda ser observadas. Primeiramente assume-se a **hipótese de mundo aberto** (HMA), ou seja, não se pode concluir a falsidade de uma informação unicamente pela ausência de uma afirmação sobre sua veracidade. A veracidade da informação deve ser definida explicitamente num axioma ou ser deduzida a partir de outros axiomas existentes, caso contrário, nada se pode afirmar sobre ela.

Outra **hipótese** assumida é a **de nomes não-únicos** (HNNU), que significa que dois indivíduos com nomes diferentes podem representar o mesmo objeto do domínio. Em conjunto com a HMA, só se pode afirmar que dois nomes de indivíduos referem-se (ou não) ao mesmo objeto se eles forem explicitamente declarados iguais (ou diferentes), caso contrário, nada se pode afirmar. Finalmente, a **monotonicidade** significa que a adição de nova informação, supostamente desconhecida, não deve interferir na informação anteriormente derivada. Em outras palavras, o que é verdade em uma situação, deve continuar sendo independentemente da adição de outra informação no modelo. Assim, tais linguagens são tipicamente projetadas para representar cenários estáticos, em que a informação pode ser completada, mas não pode, de fato, mudar. (HOEKSTRA, 2009)

3. Abordagens para Representação de Informação Temporal em OWL

3. ABORDAGENS PARA REPRESENTAÇÃO DE INFORMAÇÃO TEMPORAL EM OWL

Em vista da conhecida limitação da linguagem OWL, bem como de outras linguagens baseadas em DL, em permitir representar a informação temporal, duas linhas de pesquisa têm sido investigadas visando: (i) aumentar o poder de expressividade da linguagem (LUTZ ET AL., 2008) ou (ii) criar uma estrutura que permita representar tal informação mas preserve a semântica original da linguagem (GANGEMI, ALDO, 2005; KRIEGER, 2008; O'CONNOR; DAS, 2011; WELTY, CHRIS; FIKES, 2006). A primeira tem como consequências indesejáveis, além de frequentemente ocasionar o aumento da complexidade computacional, a indisponibilidade de ferramentas de edição e máquinas de inferência. Em contrapartida, a segunda linha, por preservar a semântica original da linguagem, mantém-se compatível com as ferramentas existentes, além de não provocar aumento da complexidade computacional.

Nesta segunda linha, este capítulo aborda duas estratégias conhecidas na literatura para representação de informação temporal, particularmente no contexto da linguagem OWL, baseadas na visão filosófica **Perdurantista 4D** e na chamada visão de **Reificação Temporal**. Ambas as estratégias são revisitadas sob a ótica de resultados da disciplina filosófica de Ontologia Formal, e então são propostas duas abordagens para representação de informação temporal em OWL que consistem de uma estrutura base ontologicamente fundamentada e de diretrizes gerais para sua aplicação. A seção 3.1 apresenta a abordagem baseada na visão Perdurantista 4D, enquanto a seção 3.2 apresenta a abordagem baseada na visão de Reificação Temporal. As abordagens são comparadas na seção 3.3, e então, em vista do entendimento da relação entre elas, são apresentadas na seção 3.4 possíveis alternativas híbridas de representação de informação temporal. Finalmente, a seção 3.5 apresenta as conclusões do capítulo.

O modelo de exemplo apresentado na Figura 20 é usado ao longo deste capítulo para ilustrar a aplicação das abordagens. Este modelo, descrito em UML, representa uma situação em que uma **pessoa**, que pode ser **homem** ou **mulher**, tem um único e imutável **nome**, pode ter um **CPF** que não pode mudar, uma **idade** que muda anualmente, e também pode ter um ou mais **apelidos** ao longo de sua vida. Ela ainda deve tem como **partes** um **cérebro** que nunca muda, e um **coração** que pode mudar. Por outro lado, tanto o **cérebro** quanto o **coração** devem ser **parte de** uma única **pessoa** durante sua existência. Finalmente, um **homem** pode estar **casado com** apenas uma **mulher** por vez (e vice-versa), tornando-se, então, **esposo** e **esposa** respectivamente.

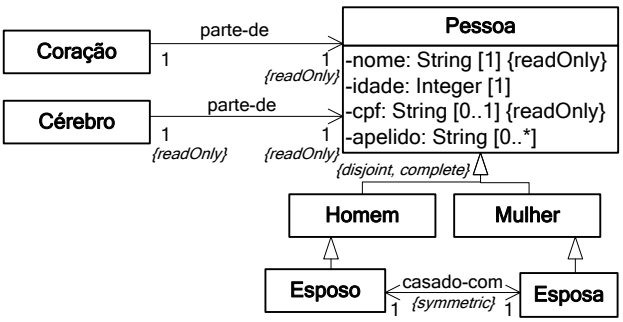


Figura 20. Modelo de domínio de exemplo.

São identificadas aqui três fontes de mudanças: atributos, instanciações de relações e instanciações de classes. Atributos e relações (também chamados propriedades) podem ser classificados em duas dimensões ortogonais: **obrigatoriedade** versus **contingência**; e **mutabilidade**. A primeira dimensão diz respeito à necessidade de um objeto de possuir tal propriedade a despeito do seu valor. Ela é representada no modelo pela restrição de cardinalidade, de forma que o caso de obrigatoriedade implica em cardinalidade mínima maior ou igual a 1. Por exemplo, os atributos **nome** e **idade**, bem como as relações **parte-de,** são obrigatórios para uma **pessoa**, enquanto os atributos **cpf** e **apelido** são contingentes. Por outro lado, a segunda dimensão diz respeito à mutabilidade do valor da propriedade, uma vez que ele seja atribuído (obrigatoriamente ou não). O caso de imutabilidade é representado no modelo por uma anotação *readOnly* ao lado do atributo ou do final de associação da relação. Por exemplo, os atributos **nome** e **cpf**, bem como a relação **parte-de** com o **cérebro** são imutáveis para uma **pessoa**, enquanto os atributos **idade** e **apelido**, bem como a relação **parte-de** com um **coração** são mutáveis para uma **pessoa**.

Por sua vez, a respeito da instanciação de classes, pode-se classificar quanto à **obrigatoriedade** versus **contingência** (apesar de não haver tal distinção na representação em UML). Algumas classes, ditas *rígidas*, são de instanciação obrigatória, ou seja, seus indivíduos devem instanciá-la enquanto existirem. Nesse exemplo, considera-se que os indivíduos das classes **Pessoa**, **Homem**, **Mulher**, **Coração** e **Cérebro** instanciam estas classes obrigatoriamente, ou seja, não podem deixar de instanciá-la sem deixar de existir. Por outro lado, outras classes, ditas *anti-rígidas*, são de instanciação contingente, ou seja, seus indivíduos podem deixar de instanciá-la sem deixar de existir. Por exemplo, considera-se que os indivíduos das classes **Esposo** e **Esposa** instanciam estas classes contingentemente, ou seja, podem deixar de instanciá-la sem deixar de existir.

Em resumo, estes são os **aspectos temporais do modelo estrutural** que se pretende representar: **classes obrigatórias** ou **contingentes**; e **propriedades obrigatórias** ou **contingentes**, **mutáveis** ou **imutáveis**.

Finalmente, combinando-se tais aspectos, identificam-se os casos de **dependência genérica, específica, ou ainda existencial**. A **dependência genérica** ocorre quanto indivíduos de uma classe A instanciam uma relação com indivíduos de uma classe B de forma obrigatória, porém mutável. Nesse exemplo, instâncias da classe **Esposo** instanciam obrigatoriamente, porém mutavelmente, a relação casado-com com alguma instância da classe **Esposa** (e vice-versa), portanto há uma dependência genérica entre as instâncias das classes. A **dependência específica**, por sua vez, requer que a relação seja imutável, além de obrigatória. Por exemplo, se se pretendesse representar nesse modelo que uma **pessoa**, uma vez casada, não pudesse casar-se com outro **esposo** ou **esposa**, tal relação seria imutável e ter-se-ia então o caso de dependência específica.

Por fim, o último e mais restritivo tipo de dependência, a **dependência existencial**, requer que a classe A, dos indivíduos dependentes, seja obrigatória, ou seja, ocorre quanto indivíduos de uma classe A, obrigatória, instanciam uma relação com indivíduos de uma classe B de forma obrigatória e imutável. Assim, nesse exemplo, as instâncias da classe **Coração** são especifica e obrigatoriamente dependentes de uma instância da classe **Pessoa** em todas as situações em que aquelas instâncias de **Coração** existirem, ou seja, um coração existe somente se uma pessoa específica que o possui existir, e não pode existir sem ela. Por outro lado, as instâncias de **Pessoa** são genericamente dependentes de alguma instância de **Coração**, ou seja, existem somente se alguma instância de coração pertencente a ele existir.

3.1. ABORDAGEM PERDURANTISTA 4D

Esta seção apresenta uma abordagem para representação de informação temporal em OWL baseada numa visão específica da teoria filosófica Perdurantista chamada de *worm view* (visão de minhoca). Primeiramente apresenta-se, na subseção 3.1.1, a teoria em questão introduzindo-se a noção ontológica de *conceito individual*. Em seguida, na subseção 3.1.2, define-se em função desta uma abordagem para representação de informação temporal em OWL. Esta consiste de uma estrutura base OWL e diretrizes de aplicação da abordagem, resultando em 3 alternativas de representação. Por fim, a subseção 3.1.3 apresenta dois trabalhos relacionados, que são comparados com as alternativas proposta neste trabalho na subseção 3.1.4.

3.1.1. Perdurantismo

Na literatura filosófica, existem duas visões clássicas que refletem diferentes modos de classificar entidades com respeito à sua relação com o tempo. Elas são chamadas de Endurantismo e

Perdurantismo (MASOLO, C. ET AL., 2003 apud GUIZZARDI, 2005). De acordo com Varzi, a primeira teoria afirma que "coisas como pessoas, pedras e mesas são continuantes tri-dimensionais que literalmente persistem através do tempo a despeito das muitas mudanças qualitativas que eles podem sofrer" (VARZI, 2003, p. 4). Em tal teoria, um continuante como pessoa é fundamentalmente diferente do que é chamado de perdurante (ou processo). O primeiro é dito totalmente presente sempre que ele existir, enquanto o segundo tem partes temporais que se desdobram no tempo. Exemplos de perdurantes incluem uma festa de aniversário, um processo de negócio, um jogo de futebol e a Segunda Guerra Mundial. Além disso, nesta visão, perdurantes não podem mudar genuinamente. Em outras palavras, se um perdurante **p** tem a propriedade **x** no tempo $\mathbf{t_1}$ e uma outra propriedade **y** (possivelmente contraditória) no tempo $\mathbf{t_2}$, então existem $\mathbf{tp_1}$ e $\mathbf{tp_2}$ partes temporais de **p** tais que: (i) $\mathbf{tp_1}$ acontece em $\mathbf{t_1}$ e tem a propriedade **x** e (ii) $\mathbf{tp_2}$ acontece em $\mathbf{t_2}$ e tem a propriedade **y**.

Ao contrário da posição endurantista, a visão perdurantista elimina a distinção entre continuantes e perdurantes argumentando que

> objetos ordinários tais como pessoas, pedras ou mesas não são continuantes, eles são perdurantes. Eles têm tanto partes espaciais quanto temporais, ou estágios, e dizer que tais objetos persistem no tempo é dizer que eles tem diferentes partes que existem em diferentes tempos. Assim, nesta visão, a pessoa na minha frente agora não é John Doe em sua totalidade. Ele é apenas uma parte de John, assim com eu não estou exposto a toda sua vida, mas apenas ao seu estágio atual. (Neste sentido, objetos ordinários não são diferentes de eventos, que também se estendem no tempo) (VARZI, 2003, p. 4).

Assim, entidades que outrora seriam vistas como continuantes 3D, são tidas numa posição perdurantista como objetos 4D, uma vez que nesta teoria todas as entidades são vistas como minhocas espaço-temporais, cujas partes temporais são fatias da minhoca. Embora a teoria endurantista esteja mais próxima do senso comum, a visão perdurantista ganha notoriedade quando se precisa observar (e representar) um dado fenômeno no decorrer do tempo. Esta teoria é referida apenas como visão 4D no restante deste trabalho.

Outra forma de observar um fenômeno numa visão 4D, é aplicando-se a noção ontológica de conceito individual (*individual concept*). Segundo Guizzardi (2005), esta noção foi introduzida por Leibniz e se refere a uma característica (ou conjunto de características) única, obrigatória e imutável, que vale apenas para um indivíduo e permite mapeá-lo às suas partes temporais. Por exemplo, supondo que o nome próprio[7] de João representa seu conceito individual, então este permite referir-se a João em sua infância ou adolescência. Assim, propõe-se observar o indivíduo

[7] O sentido de "nome próprio" usado aqui é o sentido filosófico tal qual empregado por S. Kripke (1982 apud GUIZZARDI, 2005)

em duas visões: uma estática e outra dinâmica. Enquanto a primeira diz respeito ao conceito individual, a segunda visão abrange justamente as outras características do indivíduo, que representam diferentes momentos em sua história.

A Figura 21a ilustra a visão clássica 4D, em que um continuante 3D possui uma quarta dimensão temporal. Neste exemplo, um indivíduo é representado como sendo estendido em uma quarta dimensão temporal, compondo um objeto 4D, também chamado de minhoca espaço-temporal, cujas fatias são visões instantâneas durante sua vida (também ditas suas partes ou fatias temporais). Por outro lado, a Figura 21b mostra o mesmo exemplo dividido nas visões estática e dinâmica, de forma que o conceito individual, representado pelo nome do indivíduo, João, permite mapeá-lo às suas partes temporais, como sua infância ou adolescência.

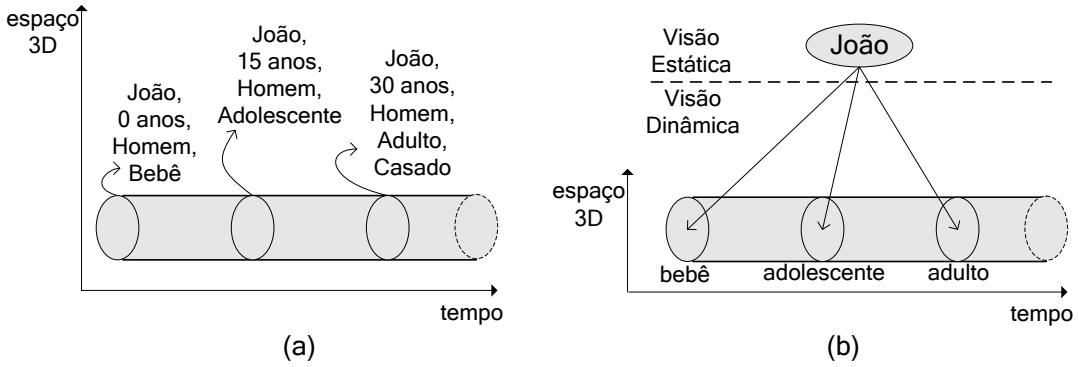


Figura 21. Esquemas ilustrativos de um objeto segundo a visão clássica 4D (a) e dividido em visões estática e dinâmica (b).

3.1.2. Abordagem 4D

A abordagem para representação de informação temporal apresentada nesta seção é baseada na estratégia 4D divida em visões estática e dinâmica apresentada na seção anterior. Assim, os indivíduos são compostos de partes temporais e um conceito individual que as mapeia.

A estrutura base OWL para representação desta abordagem é dividida em 2 níveis que refletem as visões estática e dinâmica, conforme ilustrado na Figura 22 com um esquema estilo UML. No primeiro nível, chamado nível IC (*IC Level*) ou nível estático, fica a classe *IndividualConcept* cujas instâncias são conceitos individuais que podem estar relacionados entre si através da propriedade *objPropertyIC*. Por outro lado, no segundo nível, chamado de nível TS (*TS Level*) ou nível dinâmico, ficam a classe *TimeSlice*, cujas instâncias são partes temporais dos indivíduos que podem estar relacionadas entre si através da propriedade *objPropertyTS*, bem como a classe *TemporalExtent*, cujas instâncias representam a extensão temporal dessas partes. As instâncias deste modelo devem pertencer a exatamente uma dessas classes, ou seja, a classe raiz do modelo OWL, *owl:Thing*, é particionada disjunta e completamente (*disjoint*, *complete*) por tais classes.

Ademais, uma instância de *TimeSlice* representa uma parte temporal de uma instância de *IndividualConcept* instanciando a propriedade *timeSliceOf*.

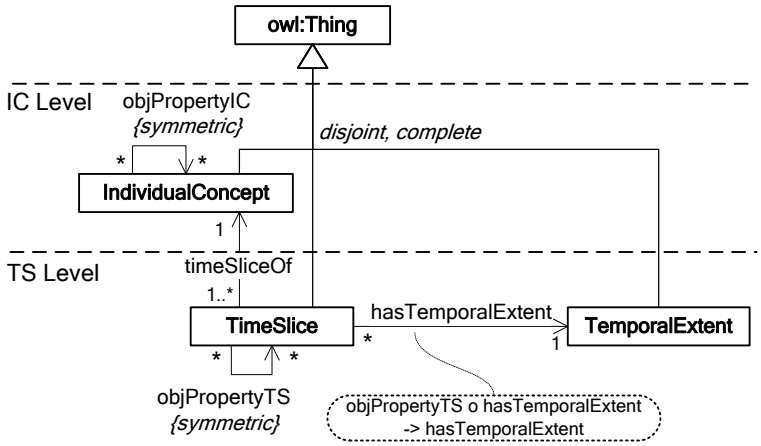


Figura 22. Esquema estilo UML da estrutura base OWL para a abordagem 4D.

As seguintes diretrizes gerais são definidas para guiar o uso adequado desta estrutura, e, consequentemente, da abordagem:

(d1)	cada conceito individual deve ter pelo menos uma parte temporal, bem como cada parte temporal deve pertencer a um único conceito individual, e também deve ter uma única extensão temporal. A extensão temporal total de um conceito individual é dada pela soma da extensão de todas as suas fatias[8]. Esta diretriz é garantida pelas restrições de cardinalidade sobre as classes *IndividualConcept* e *TimeSlice* em relação às propriedades *timeSliceOf* e *hasTemporalExtent*;
(d1.1)	todo conceito **C** do domínio representado no nível IC deve especializar (direta ou indiretamente) a classe *IndividualConcept* e também deve ter uma contraparte temporal específica **CTS** no nível TS que especializa (direta ou indiretamente) a classe *TimeSlice*. Ambos devem ser restringidos em relação à instanciação da relação *timeSliceOf*, de forma que as instâncias de **C** tenham como partes temporais apenas instâncias de sua contraparte **CTS**, e que cada instância de **CTS** seja parte temporal apenas de alguma instância de **C**;
(d1.2)	todo conceito representado no nível TS deve ser acrescido do sufixo 'TS', e também deve ter uma contraparte deste conceito no nível IC, ou, caso contrário, deve especializar (direta ou indiretamente) um outro conceito **CTS** que tenha tal contraparte;
(d2)	as instâncias de cada nível podem se relacionar (instanciar propriedades de objeto) apenas com indivíduos do mesmo nível (excetuando-se a propriedade *timeSliceOf*). Assim, as relações específicas do domínio representadas no nível IC especializam a propriedade *objPropertyIC*, enquanto aquelas representadas no nível TS especializam *objPropertyTS*;
(d3)	as instâncias inter-relacionadas do nível TS devem ter a mesma extensão temporal. Esta diretriz é garantida pela cadeia de propriedade [*objPropertyTS* o *hasTemporalExtent* → *hasTemporalExtent*].

[8] Essas fatias não devem se sobrepor temporalmente, entretanto, restrições sobre o tempo não são contempladas desde trabalho.

A seguir são detalhadas três alternativas, nomeadas **Alternativa A0**, **A1** e **A2**, que consistem em diretrizes que distribuem diferentemente as informações do domínio nos níveis IC e TS conforme sua natureza ontológica.

Alternativa A0

Uma primeira alternativa consiste simplesmente em compor conceitos individuais com suas fatias temporais completas, ou seja, toda a informação do domínio está nas fatias temporais. Assim, as diretrizes desta alternativa são:

(a0.1)	apenas os conceitos rígidos do domínio são representados no nível IC;
(a0.2)	todos os conceitos, relações e atributos do domínio são representados no nível TS.

Aplicando-se esta alternativa ao modelo de exemplo da Figura 20 tem-se como resultado o modelo OWL ilustrado na Figura 23. Os elementos em cinza não fazem parte do domínio, são tipos de dados ou os elementos especializados da estrutura base.

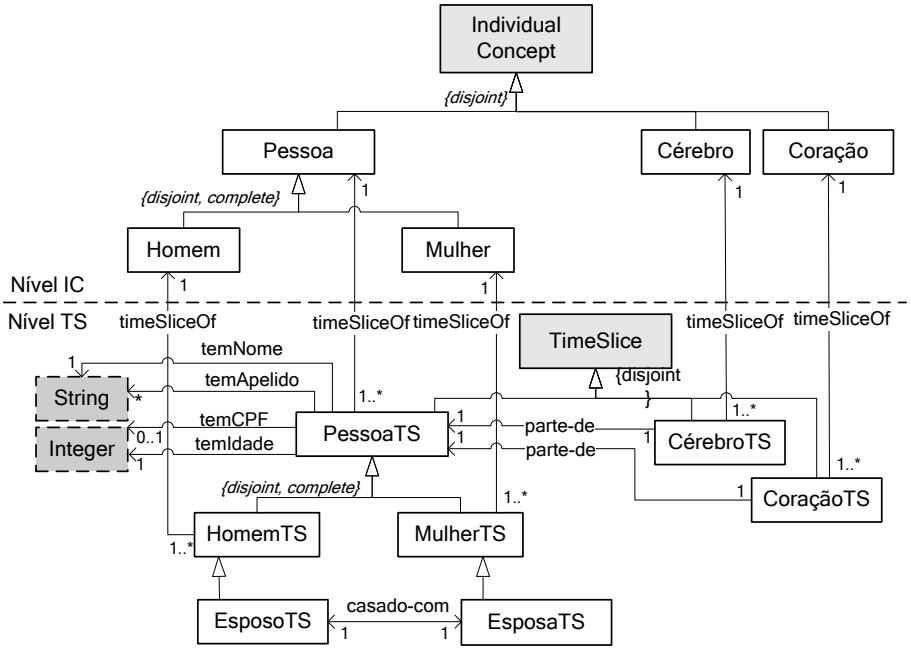


Figura 23. Esquema estilo UML da aplicação da alternativa A0 ao exemplo da Figura 20.

Um exemplo de instanciação deste modelo é ilustrado na Figura 24. Ele ilustra um cenário em que um **homem** dos **27** aos **28 anos** é chamado **João** e possui um **cpf "098..."** e um **cérebro**, aos **28 anos casou-se** com uma **mulher** de **26 anos** chamada Maria, e esteve **casado** até os **28 anos** e também aos **28 anos** (possivelmente[9]) trocou o **coração** que possuía aos **27**. Na parte superior, o nível IC, estão representados como elipses os conceitos individuais, enquanto na parte inferior, o nível TS, são

[9] Segundo a semântica de OWL/DL, adotando-se a chamada hipótese de nomes não-únicos, os indivíduos **h1** e **h2** podem ser idênticos.

representadas como figuras cilíndricas as minhocas espaço-temporais, que se estendem no tempo verticalmente para baixo. Os rótulos precedidos do símbolo **::** representam as classes que os indivíduos instanciam. Cada divisão no cilindro representa uma fatia temporal que se estende até o início da próxima (ou até o final do cilindro no caso da última fatia). As divisões mais escuras representam fonte de mudança, em que uma classe ou relação passou a ser ou deixou de ser instanciada, ou atributo mudou de valor.

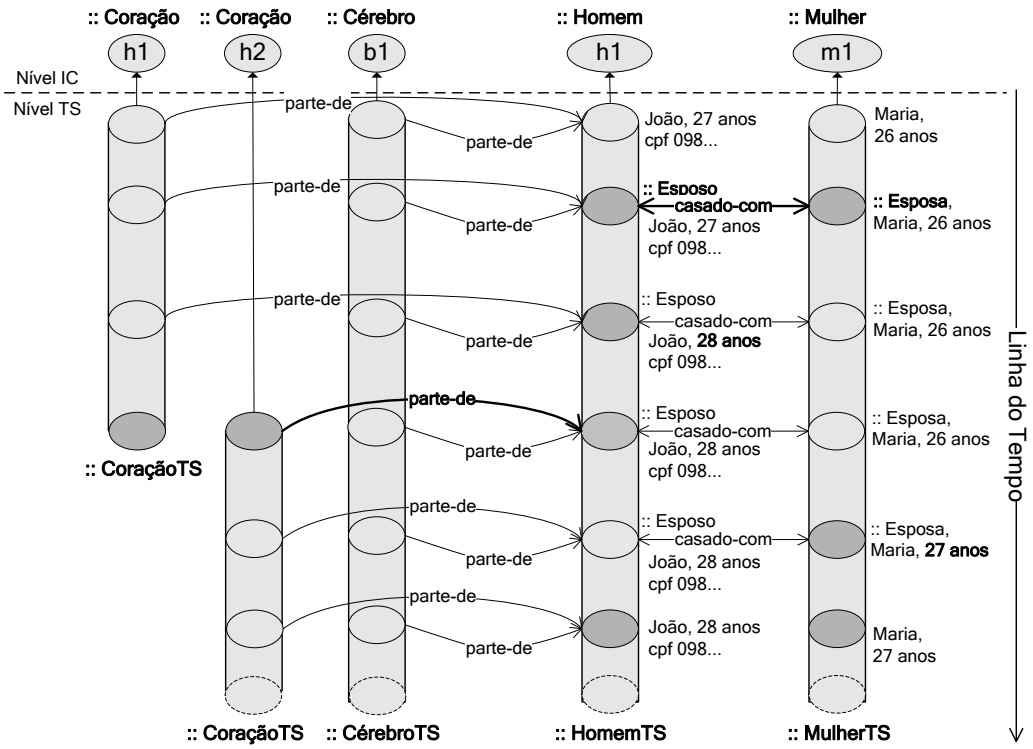


Figura 24. Esquema ilustrativo de instanciação do modelo da Figura 23.

Observa-se neste exemplo que o resultado da aplicação desta alternativa permite representar a informação temporal de modo bem próximo à idéia original da visão 4D, em que as fatias temporais to indivíduo contém toda informação ele num dado momento. Contudo, identificam-se alguns problemas:

(i) **proliferação de partes temporais**: devido à diretriz *d3*, qualquer mudança numa parte temporal *p* requer uma nova parte *p'*, e requer também que cada parte temporal q_i na cadeia de partes inter-relacionadas (que inclui *p*) tenham também novas partes temporais q_i' que possuam a mesma extensão temporal de *p'*;

(ii) **interpretação ontológica estranha de conceitos contingentes**: instâncias de conceitos rígidos são representadas tanto na visão estática como conceitos individuais, quanto na visão dinâmica como partes temporais. Este não é o caso, porém, para instâncias de conceitos não-rígidos, que existem apenas na visão dinâmica;

(iii) **repetição tediosa das propriedades imutáveis no nível TS**: o **nome de João** e o **CPF de João** são tediosamente repetidos para cada parte temporal de **João**;

(iv) **não é possível garantir a imutabilidade das propriedades imutáveis**: não há como garantir que as propriedades imutáveis repetidas para cada parte temporal tenham os valores idênticos, por exemplo, que o valor do **cpf de João** no intervalo de tempo t_1 é idêntico ao do **seu cpf** no intervalo t_2.

Alternativa A1

Outra alternativa consiste em representar no nível IC as propriedades necessárias e imutáveis dos indivíduos, ou seja, aquelas que valem enquanto ele existir. Neste caso, deve-se considerar apenas as relações que implicam em dependência existencial mútua, pois aquelas unilaterais são imutáveis apenas para um dos indivíduos envolvidos. Assim, as diretrizes desta alternativa são:

(a1.1)	os conceitos rígidos, os atributos necessários e imutáveis, bem como as relações que implicam em dependência existencial mútua são representados no nível IC;
(a1.2)	os conceitos, relações que não implicam em dependência existencial mútua e atributos não simultaneamente necessários e imutáveis são representados no nível TS.

Aplicando-se esta alternativa ao modelo de exemplo da Figura 20 tem-se como resultado o modelo OWL ilustrado na Figura 25 (os elementos em cinza não fazem parte do domínio).

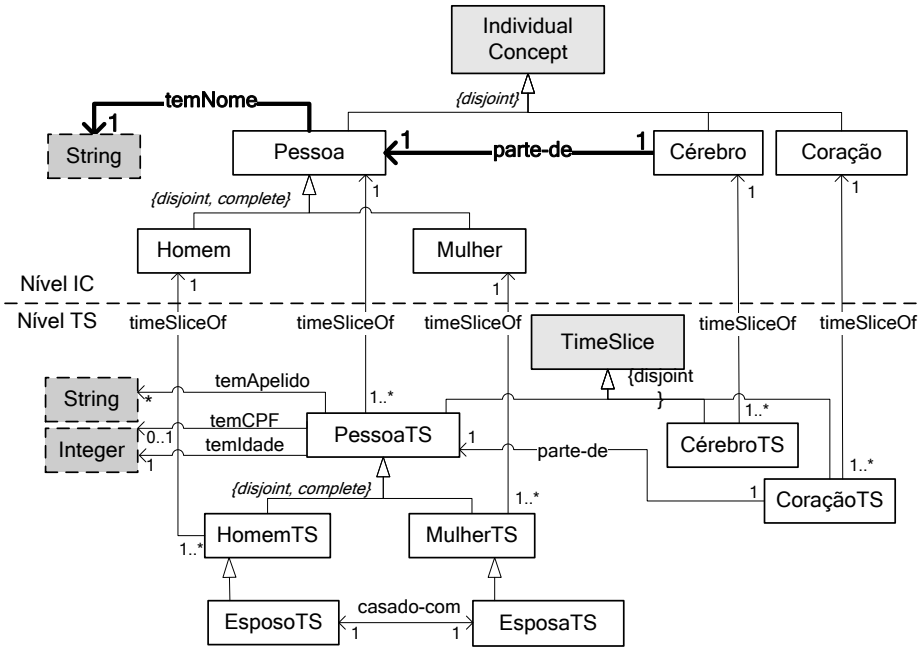


Figura 25. Esquema estilo UML da aplicação da alternativa A1 ao exemplo da Figura 20.

Destaca-se que a propriedade **parte-de** entre as classes **Pessoa** e **Cérebro**, que implica em dependência existencial mútua, bem como a propriedade **temNome** entre **Pessoa** e o tipo de dado *String*, que é necessária e imutável, são representadas agora no nível IC.

Um exemplo de instanciação deste modelo é ilustrado na **Erro! Fonte de referência não encontrada.** (explicação análoga à da Figura 24). Ele ilustra um cenário em que um **homem** é chamado **João** e possui um **cérebro**, e dos **27** aos **28 anos** possui um **cpf "098..."**, aos **28 anos casou-se** com uma **mulher** chamada Maria de **26 anos**, e esteve casado até os **28 anos** e também aos **28 anos** trocou o **coração** que possuía aos **27**. Destaca-se que (i) o nome das instâncias de **Homem** e **Mulher**, bem como a relação **parte-de** entre a instância **b1** de **Cérebro** e a instância **João** de **Homem** estão representados no nível IC; (ii) as mudanças ocorridas nas partes temporais de **João** não implicam em novas partes temporais para a instância **b1**.

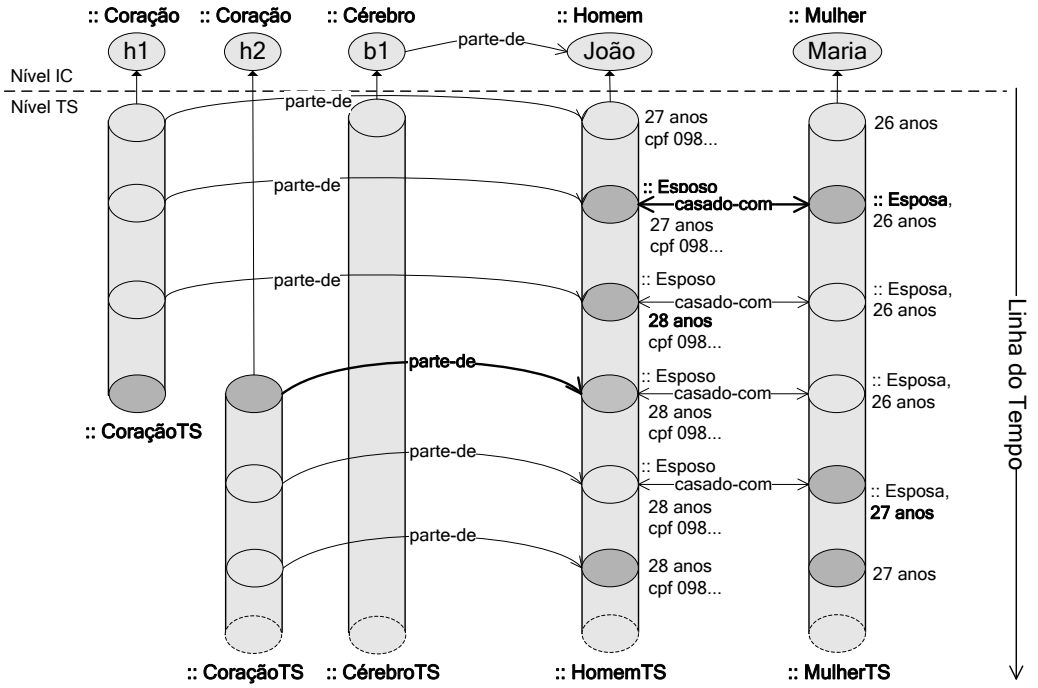


Figura 26. Esquema ilustrativo de instanciação do modelo da Figura 25.

Assim, observa-se que esta alternativa traz as seguintes vantagens em comparação com a alternativa A0: (i) diminui-se a inter-relação entre as partes temporais, uma vez que as relações de dependência existencial mútua são representadas no nível IC, permitindo-se uma redução na proliferação de partes temporais; (ii) as informações necessárias e imutáveis representadas no nível IC já não são repetidas nem podem ser modificadas. Em contrapartida, impede-se inferências no nível TS que dependam das relações representadas no nível de IC, pois as partes temporais não estão mais ligadas diretamente, mas indiretamente. Por fim, mesmo em escala menor, ainda são observados os mesmos inconvenientes, por exemplo, a repetição das propriedades imutáveis ocorre apenas para aquelas mantidas no nível TS, como o **cpf de João**.

Alternativa A2

A última alternativa consiste em representar no nível IC as propriedades necessárias e imutáveis dos indivíduos, ou seja, aquelas que valem enquanto ele existir. Neste caso, as relações que implicam em dependência existencial unilateral são interpretadas como válidas apenas durante o tempo de existência do dependente. Assim, as diretrizes desta alternativa são:

(a2.1)	os conceitos rígidos, os atributos necessários e imutáveis e as relações que implicam em dependência existencial são representados no nível IC;
(a2.2)	os conceitos, relações que não implicam em dependência existencial e atributos não simultaneamente necessários e imutáveis são representados no nível TS;
(a2.3)	as relações que implicam em dependência existencial unilateral, são representadas no nível IC e interpretadas como válidas durante o tempo de existência do dependente. Assim, se a tal relação for mutável para o indivíduo independente, ou seja, se este indivíduo pode instanciar tal relação várias vezes durante sua existência, então a cardinalidade máxima no sentido inverso da dependência deve ser flexibilizada. Por exemplo, uma instância de **Pessoa** pode instanciar a relação (inversa de) **parte-de** com diferentes instâncias de **Coração** durante sua existência, portanto, a restrição de cardinalidade máxima 1 deve ser flexibilizada para permitir vários relacionamentos deste tipo para cada instância de **Pessoa**.

Aplicando-se esta alternativa ao modelo de exemplo da Figura 20, tem-se como resultado o modelo OWL ilustrado na Figura 27 (os elementos em cinza não fazem parte do domínio). Destaca-se que a propriedade **parte-de** entre as classes **Pessoa** e **Coração**, que implica em dependência existencial unilateral, é representada agora no nível IC.

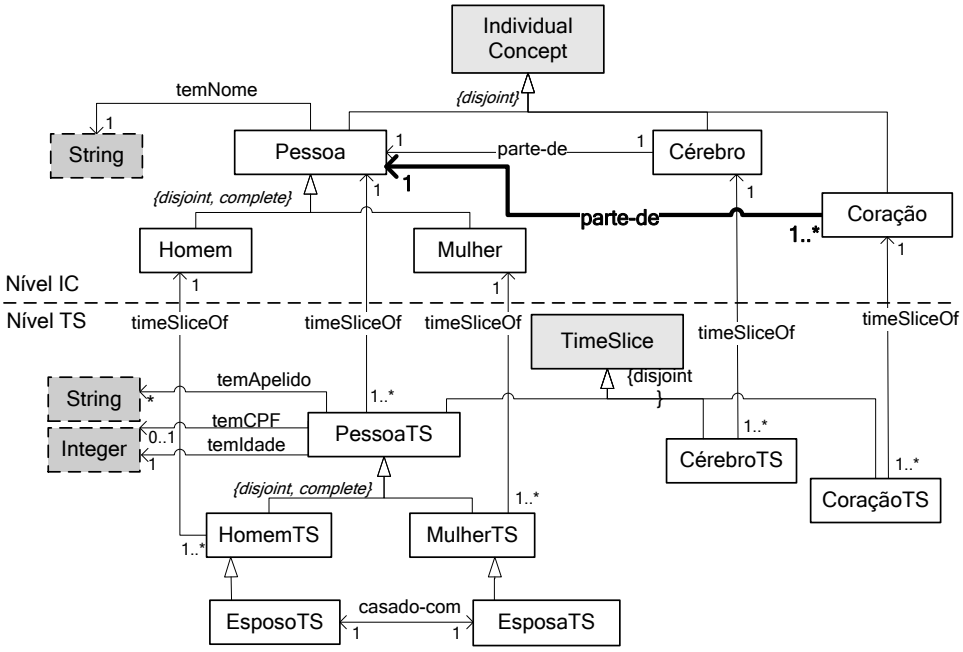


Figura 27. Esquema estilo UML da aplicação da alternativa A2 ao exemplo da Figura 20.

Um exemplo de instanciação deste modelo é ilustrado na Figura 28 (a explicação é análoga à da Figura 24). Ele ilustra um cenário em que um **homem** é chamado **João** e possui um **cérebro**, e dos **27**

aos **28 anos** possui um **cpf "098..."**, aos **28 anos casou-se** com uma **mulher** chamada Maria de **26 anos**, e esteve **casado** até os **28 anos** e possui ainda duas instâncias de **coração** com projeções temporais disjuntas, sendo que o momento da troca corresponde ao período em que **João** tinha **28 anos**. Destaca-se que (i) as relações **parte-de** entre a instância **h1** e **h2** de **Coração** e a instância **João** de **Homem** estão representados no nível IC; (ii) as mudanças ocorridas nas partes temporais da instância **João** não implicam em novas partes temporais para a instância **h1** e **h2**, e vice-versa.

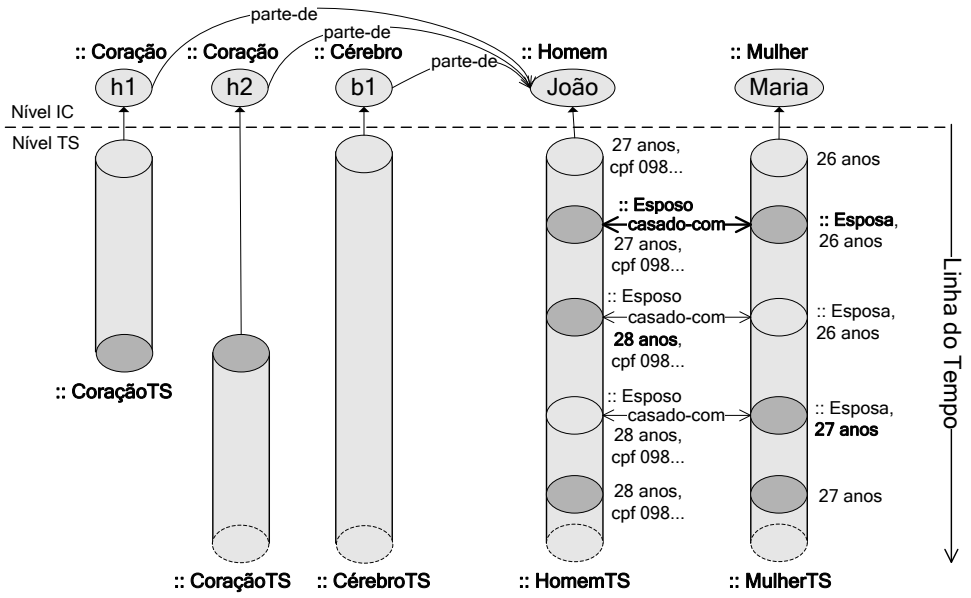


Figura 28. Esquema ilustrativo de instanciação do modelo da Figura 27.

Assim, observa-se que esta alternativa traz as como vantagem em comparação com a alternativa A1, diminuir ainda mais a inter-relação entre as partes temporais, uma vez que todas as relações de dependência existencial são representadas no nível IC, permitindo-se uma maior redução na proliferação de partes temporais. Em contrapartida, impede-se inferências no nível TS que dependam das relações representadas no nível de IC, pois as partes temporais não estão mais ligadas diretamente, mas indiretamente. Também perde-se a capacidade de restringir a cardinalidade das relações de dependência existencial unilateral. Por fim, mesmo em escala menor, ainda são observados os mesmos inconvenientes.

3.1.3. Trabalhos Relacionados

Abordagem de Welty & Fikes – *Ontology for fluents*

Welty e Fikes (2006) introduzem o conceito de fluentes na modelagem OWL, considerando-se fluentes como relações que valem durante um certo intervalo de tempo. A estrutura de fundamentação desta abordagem é apresentada na Figura 29 (em fundo branco). Cada indivíduo do modelo pode ser relacionado a uma ou mais partes temporais através da relação *temporalPartOf*.

O domínio desta relação é a classe *TemporalPart* enquanto o domínio é definido como o complemento da classe *TimeInterval* (denotada como ~*TimeInterval*). Uma ontologia de domínio deve especializar tal estrutura da seguinte forma: (i) todas as classes são representadas como especializações da classe ~*TimeInterval* e (ii) as propriedades que são fluentes são representadas entre partes temporais, especializando-se a relação *fluentPropertyOf*, e seu domínio e imagem são definidos como classes anônimas que restringem a classe *TemporalPart*.

Seguindo-se estritamente essas diretrizes, não fica óbvio como o exemplo da Figura 20 pode ser adequadamente representado, uma vez que as classes contingentes *Esposo* e *Esposa* seriam representadas, juntamente com as outras, como especialização de ~*TimeInterval*, isto é, não seriam tratadas como contingentes. Uma representação aproximadamente equivalente à alternativa A1 é apresentada na Figura 29, excluindo-se os conceitos contingentes supracitados. De fato, esta abordagem foi definida com um alto nível de generalidade visando reusabilidade.

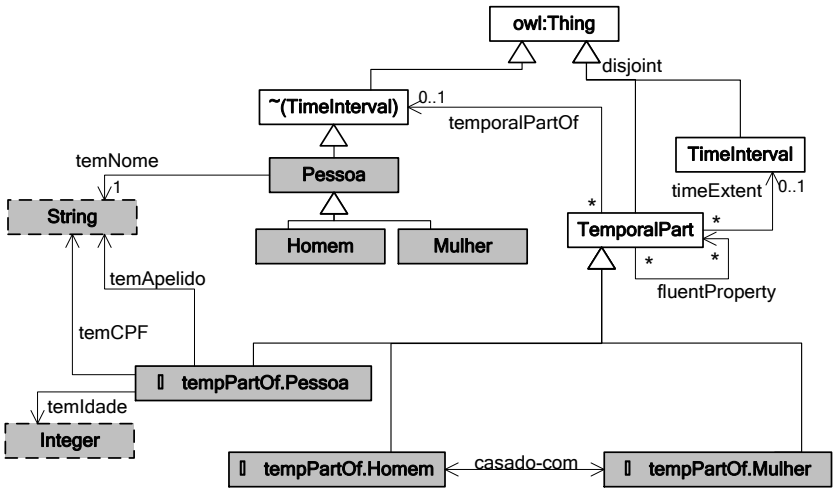


Figura 29. Esquema estilo UML da abordagem proposta por Welty e Fikes.

Um exemplo de instanciação deste modelo é ilustrado na Figura 30 (a explicação é análoga à da Figura 24). O resultado obtido é similar ao da alternativa A1, exceto pelo fato de que os conceitos rígidos e não rígidos não são explicitamente representados dentre as partes temporais (na verdade eles são representados pelas classes anônimas mostradas na Figura 29).

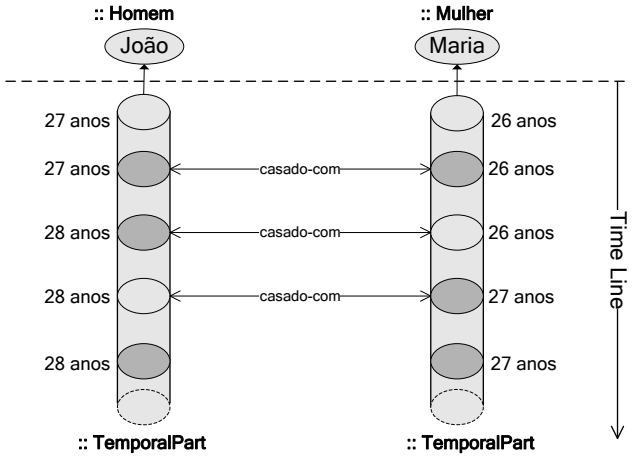


Figura 30. Esquema ilustrativo de instanciação do modelo da Figura 29.

Abordagem de Krieger – *4D Re-interpretation*

Krieger (2008) também propõe uma abordagem perdurantista, baseada na abordagem de Welty e Fikes supracitada, com o objetivo de favorecer o reuso de ontologias existentes. Ele sugere que "o que foi uma entidade agora se torna uma parte temporal", isto é, as classes de uma ontologia existente devem especializar a classe *TimeSlice*, e suas instâncias podem compor um perdurante, instância da classe *Perdurant*. Portanto, todas as propriedades desta mesma ontologia são, a princípio, fluentes. Entretanto, o autor sugere que as relações imutáveis sejam representadas entre perdurantes (ao invés de ser entre as classes da ontologia original). Além de representar a mutabilidade das relações, esta proposta permite também representar conceitos contingentes. Outras contribuições estão especificamente relacionadas a representação e raciocínio sobre o tempo. A estrutura de fundamentação desta abordagem é apresentada na Figura 31 (em fundo branco).

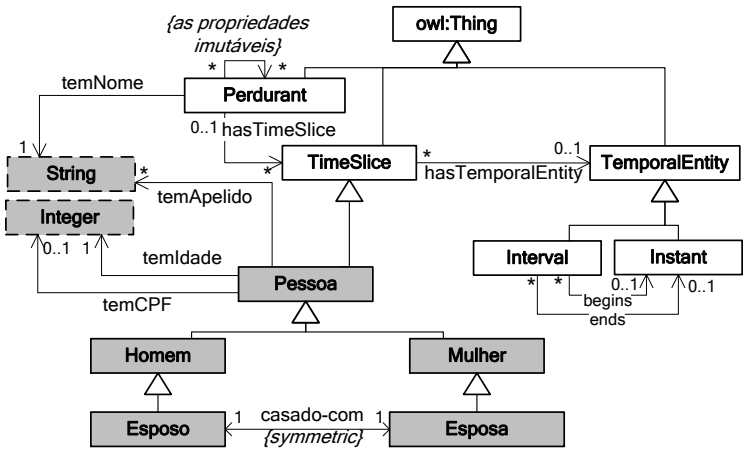


Figura 31. Esquema estilo UML da estrutura OWL proposta por Krieger.

Em contraste com a abordagem de Welty e Fikes, uma representação mais direta do modelo de exemplo pode ser obtida seguindo estritamente as diretrizes, como pode ser observado na Figura

31 (em fundo cinza). Neste caso, todo o modelo é posto como especialização de *TimeSlice*. Apesar de ser esta a razão pela qual os conceitos contingentes podem agora ser representados, outra consequência disto é a possibilidade de uma combinação arbitrária de partes temporais formando um perdurante. Por exemplo, um perdurante pode ser composto por partes temporais de uma mesa, um cachorro, um esposo, etc., permitindo-se criar indivíduos absurdos do ponto de vista ontológico (GUIZZARDI, 2005). Esta situação indesejável é evitada em nossa abordagem por causa das diretrizes **d1**, **d1.1** e **d1.2**.

Um exemplo de instanciação deste modelo é ilustrado na Figura 32 (a explicação é análoga à da Figura 24). O resultado obtido é similar ao da alternativa A1, exceto pelo fato de que os conceitos rígidos não são representados no nível IC.

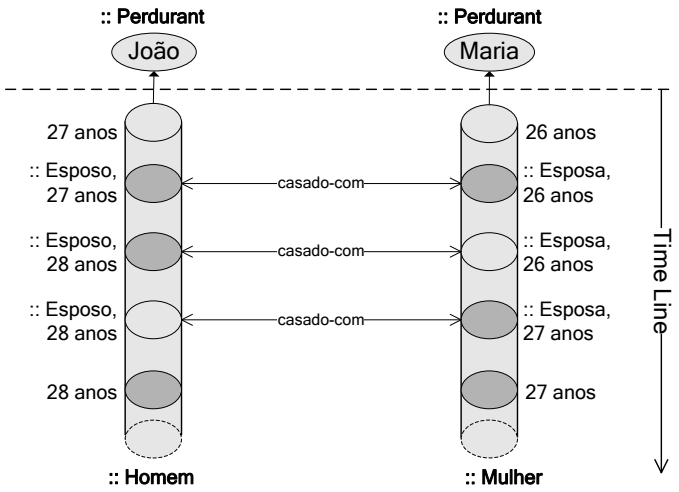


Figura 32. Esquema ilustrativo de instanciação do modelo da Figura 31.

3.1.4. Considerações e Comparação entre as Abordagens 4D

As abordagens 4D aqui discutidas têm por objetivo a representação de informação temporal em OWL. A comparação entra as abordagens é feita em duas dimensões: (i) em relação aos aspectos temporais contemplados; e (ii) em relação aos problemas detectados. Considera-se a existência dos níveis, mesmo não sendo explicitamente definida pelos autores dos trabalhos relacionados, de forma que o nível, dinâmico (TS) é aquele cujas entidades possuem extensão temporal, ao contrário do nível estático (IC), cujas entidades possuem partes temporais representadas pelas entidades do outro nível.

Com relação à primeira dimensão, os aspectos temporais do modelo estrutural são contemplados da seguinte forma na abordagem 4D:

- **Classes**
  - **obrigatórias**: se são representadas no nível estático;

- **contingentes**: se são representadas apenas no nível dinâmico;

- **Propriedades**
  - **obrigatórias**: se a cardinalidade mínima para a classe domínio da relação é restringida para exatamente ou no mínimo 1 em qualquer um dos níveis, por exemplo, a cardinalidade da relação **temNome** é exatamente 1 para a classe **Pessoa**, logo, esta relação é uma propriedade obrigatória para toda instância desta classe. Entretanto, na alternativa A2, para as relações de dependência unilateral, representadas no nível estático, não se pode garantir a obrigatoriedade para o indivíduo independente. Garante-se apenas que há uma instância desta relação durante a existência do indivíduo, mas não se garante que durante toda sua existência esta relação sempre é instanciada. Por exemplo, garante-se que toda instância de **Pessoa** instância pelo menos uma vez a relação **parte-de** com alguma instância de coração durante sua existência, mas não que esta relação esteja sempre instanciada durante toda sua existência;
  - **contingentes**: se são representadas no nível dinâmico com cardinalidade mínima zero, por exemplo, uma **pessoa** tem contingentemente um **CPF**. Na alternativa A2, as relações de dependência existencial unilateral, representadas no nível estático, são contingentes para classe de indivíduos independentes da relação se são representadas também com cardinalidade mínima igual a zero;
  - **mutáveis**: se são representadas no nível dinâmico. Na alternativa A2, as relações de dependência existencial unilateral, representadas no nível estático, são mutáveis para classe de indivíduos independentes da relação se são representadas com cardinalidade máxima flexibilizada, por exemplo, a relação **parte-de** entre **Coração** e **Pessoa**;
  - **imutáveis**: sendo obrigatórias, se são representadas no nível estático, por exemplo a relação **parte-de** entre **Cérebro** e **Pessoa**;

- **Dependência**
  - **genérica unilateral**: se é possível representar propriedades obrigatórias, no nível estático ou no dinâmico;
  - **genérica mútua**: se é possível representar propriedades mutuamente obrigatórias no nível dinâmico;
  - **existencial unilateral**: se é possível representar obrigatoriedade e imutabilidade, juntamente com contingência ou mutabilidade das propriedades no nível estático; é possível apenas na alternativa A2, em que uma relação é interpretada como válida durante o tempo de vida do dependente, podendo ser contingente e/ou mutável para o indivíduo independente da relação.

- **existencial mútua**: se é possível representar obrigatoriedade e imutabilidade das propriedades no nível estático;

Assim, com relação aos aspectos temporais do modelo estrutural contemplados pelas abordagens 4D tem-se que:

(i) a **abordagem de Welty & Fikes** não prevê a representação de classes no nível dinâmico. Assim, não é contemplado o aspecto temporal de **contingência para classes**. Consequentemente, não é possível restringir a cardinalidade dessas classes, e, portanto, a **obrigatoriedade das propriedades** não é contemplada no nível dinâmico. Também a **dependência genérica**, uma vez que depende da obrigatoriedade das propriedades neste nível, não pode ser contemplada. Apesar de garantir **obrigatoriedade e imutabilidade das propriedades** no nível estático, o que também garante a **dependência existencial mútua**, não é previsto propriedades mutáveis ou contingentes neste nível, o que impede a **dependência existencial unilateral**. As **propriedades imutáveis** são apenas aquelas representadas no nível estático.

(ii) a **abordagem de Krieger** não prevê a representação de classes no nível estático. Assim, não é contemplado o aspecto temporal de **obrigatoriedade para classes**. Consequentemente, não é possível restringir a cardinalidade dessas classes, e, portanto, a **obrigatoriedade das propriedades** não é contemplada no nível estático. Também a **dependência existencial mútua ou unilateral**, uma vez que dependem da obrigatoriedade das propriedades no nível estático, não podem ser contempladas. As **propriedades imutáveis** são apenas aquelas representadas no nível estático.

(iii) a **alternativa A0** prevê todas as propriedades do indivíduo representadas no nível dinâmico (exceto pelos conceitos necessários, que também são representados no nível estático). Assim, não é possível garantir a **imutabilidade** de nenhuma **das propriedades**, e, consequentemente, o aspecto temporal de **dependência existencial** não é contemplado;

(iv) a **alternativa A1**, apesar de garantir **obrigatoriedade e imutabilidade das propriedades** no nível estático, o que também garante a **dependência existencial mútua**, não prevê propriedades mutáveis ou contingentes neste nível, o que impede a **dependência existencial unilateral**. As **propriedades imutáveis** são apenas aquelas representadas no nível estático.

(v) a **alternativa A2** não garante a **imutabilidade** apenas das **propriedades contingentes** representadas no nível dinâmico. Por exemplo, o atributo CPF é imutável porém contingente, representado no nível dinâmico, e sua imutabilidade não pode ser garantida. Além disso, não permite restringir a cardinalidade temporal das relações de dependência existencial unilateral, uma vez sua cardinalidade máxima precisa ser flexibilizada para os

indivíduos independentes da relação. Não garante **obrigatoriedade das propriedades** no nível estático, quando a relação é de dependência existencial para um lado, e genérica para o outro. Assim, não permite representar a **dependência genérica unilateral** no nível estático.

Os aspectos temporais contemplados por cada alternativa são comparados na Tabela 3. A seguinte notação é adotada: (+) quando o aspecto é contemplado; (-) caso contrário; (-/+) ou (+/-) quando o aspecto é parcialmente contemplado, sendo que a segunda notação representa vantagem em relação à primeira.

Tabela 3. Comparação das abordagens 4D em relação aos aspectos temporais contemplados.

Abordagem	Classe		Propriedades				Dependência			
							genérica		existencial	
	obrig.	cont.	obrig.	cont.	mut.	imut.	uni-lat	mútua	uni-lat	mútua
Welty & Fikes	+	-	-/+	+	+	-/+	-	-	-	+
Krieger	-	+	-/+	+	+	-/+	+	+	-	-
A0	+	+	+	+	+	-	+	+	-	-
A1	+	+	+	+	+	-/+	+	+	-	+
A2	+	+	+/-	+	+	+/-	+/-	+	+	+

Por outro lado, os aspectos negativos identificados em cada abordagem são comparados a seguir:

(i) a **proliferação de partes temporais** é amenizada na abordagem de Krieger e nas alternativas A1 e A2, uma vez que as relações imutáveis são representadas no nível estático, ao invés do dinâmico, diminuindo assim a inter-relação entre as partes temporais, e, consequentemente, a proliferação destas em decorrência das mudanças. A alternativa A2 se destaca pois prevê também a representação de relações de dependência unilateral no nível estático;

(ii) a **combinação arbitrária de partes temporais** não ocorre nas alternativas A0, A1 e A2 devido à diretriz **d1.1**, que garante que os indivíduos de uma classe do nível TS são partes temporais apenas de instâncias de sua contraparte no nível IC. Este quesito não se aplica à abordagem de Welty & Fikes uma vez que as partes temporais não são classificadas;

(iii) a **interpretação ontológica de conceitos contingentes** ocorre nas alternativas A0, A1 e A2 uma vez que os conceitos contingentes têm como instâncias apenas partes temporais dos indivíduos, enquanto os conceitos necessários possuem instâncias em ambos os níveis estático e dinâmico. Na abordagem de Welty & Fikes, os conceitos contingentes não são representados diretamente, mas através das classes anônimas no nível TS. Assim, este

problema ocorre igualmente nesta abordagem. Com relação à abordagem de Krieger, todos os conceitos têm como instâncias apenas partes temporais, que por sua vez compõem indivíduos de um único tipo, *Perdurant*. Isto significa, em principio, que todos os objetos 3D da nossa experiência cotidiana pertencem a único tipo, o que seria absurdo do ponto de vista ontológico;

(iv) a **repetição da informação imutável**, bem como a **não-garantia da imutabilidade da informação imutável** são amenizadas na abordagem de Krieger e nas alternativas A1 e A2, uma vez que algumas das relações imutáveis são representadas no nível estático, ao invés do dinâmico. A alternativa A2 se destaca, pois prevê a representação de relações de dependência unilateral no nível estático.

Esses aspectos são resumidos na Tabela 4. A seguinte notação é adotada: (-) quando o aspecto negativo não é tratado, ou seja, o problema existe na abordagem; (+) caso contrário; (-/+) ou (+/-) quando o aspecto negativo é parcialmente tratado, ou seja, o problema existe mas é amenizado, sendo que a segunda notação representa o problema mais amenizado do que a primeira.

Tabela 4. Comparação das abordagens 4D com relação aos aspectos negativos tratados.

Abordagem	Proliferação de partes temporais	Combinação arbitrária de partes temporais	Interpretação ontológica estranha de conceitos contingentes	Repetição da informação imutável	Não garantia da imutabilidade da informação imutável
Welty & Fikes	-	-	-	-	-
Krieger	-/+	-	-	-/+	-/+
A0	-	+	-	-	-
A1	-/+	+	-	-/+	-/+
A2	+/-	+	-	+/-	+/-

Conclui-se que a abordagem proposta neste trabalho apresenta-se mais apropriadas do que as abordagens de Welty & Fikes e Krieger para representação de informação temporal em OWL por (i) prover estrutura base e diretrizes ontologicamente fundamentadas para orientar sua utilização e (ii) prover alternativas de modelagem explicitando suas consequências.

Com relação especificamente às alternativas aqui propostas, a alternativa A2 apresenta-se superior à alternativa A1 nos quesitos avaliados, que por sua vez apresenta-se superior à alternativa A0. Entretanto, como mencionado anteriormente, por ter todas as informações representadas no

nível dinâmico, a alternativa A0 é mais apropriada para realizar inferências neste nível do que a alternativa A1, que é mais apropriada que a alternativa A2, que de todas é a que mais informação representa no nível estático.

Finalmente, cada uma das abordagens/alternativas é aplicável uma vez compreendidas suas características e limitações, e a opção por uma delas depende da adequação de suas características aos requisitos da aplicação.

3.2. ABORDAGEM DE REIFICAÇÃO TEMPORAL

Esta seção apresenta uma abordagem para representação de informação temporal em OWL baseada na estratégia de Reificação Temporal. Primeiramente introduz-se, na subseção 3.2.1, uma breve teoria sobre reificação, juntamente com algumas noções ontológicas que fundamentam sua aplicação no contexto deste trabalho. Em seguida, na subseção 3.2.2, define-se em função desta uma abordagem para representação de informação temporal em OWL, que consiste de uma estrutura base OWL e diretrizes gerais para aplicação da abordagem. Por fim, a subseção 3.2.3 apresenta dois trabalhos relacionados, que são comparados com a abordagem proposta neste trabalho na subseção 3.2.4.

3.2.1. Reificação

De acordo com o dicionário *on-line Merriam Webster*, reificar significa "tratar algo abstrato como sendo algo material ou concreto" ("Reifying,"... 2010). Assim, reificar pode ser entendido como objetificar algo de forma que ele possa ser referenciado, qualificado e quantificado. Quine (1985) apresenta reificação como uma estratégia para forjar ligações entre sentenças ou cláusulas representadas em uma linguagem lógica de primeira ordem (LPO). Por exemplo, a sentença 'Sebastian caminhou vagarosamente e sem rumo em Bolonha no tempo t' pode ser reificada como **∃x (x é uma caminhada e x é vagaroso e x é sem rumo e x é em Bolonha e x é no tempo t e x é de Sebastian)**, onde **x** é a referência objetiva que conecta todas as cláusulas.

Galton (2006) afirma que a técnica de reificação tem sido usada na comunidade de Inteligência Artificial, particularmente na área de Representação do Conhecimento desde os anos 80. O autor revisa algumas estratégias de reificação, e aponta várias alternativas para representar conhecimento temporal reificado usando LPO. Entretanto, o autor aponta a dificuldade de se definir um critério claro para decidir o que e como reificar. De fato, não existe na literatura de representação do conhecimento (até onde sabemos) uma alternativa de representação que proveja uma interpretação ontológica para reificação de propriedades, relações e estados..

No contexto deste trabalho, a distinção ontológica entre **objetos** (*objects*) e ***moments*** (ver seção 2.1.2) é vista como o significado ontológico para a reificação das propriedades dos indivíduos, uma vez que ***moments*** denotam justamente a materialização das propriedades que caracterizam outros indivíduos, enquanto **objetos** representam indivíduos caracterizados por aqueles. Particularmente, interessa neste trabalhado reificar as informações (possivelmente) contingentes e/ou mutáveis do indivíduo, como os atributos, os conceitos contingentes/anti-rígidos e as relações materiais. Assim, os tipos de *moments* que contemplam essas propriedades são respectivamente *quality*, *qua-individual* e *relator*. Para mais detalhes sobre o assunto e uma defesa ontológica para a existência de propriedades reificadas (*moments*) recomenda-se as referências (GUIZZARDI ET AL., 2006; GUIZZARDI; WAGNER, 2008).

A Figura 33 ilustra a representação de instanciação das propriedades supracitadas da forma tradicional e da forma reificada. Os rótulos precedidos do símbolo **::** próximos aos elementos indicam as classes que eles instanciam, e as setas rotuladas indicam a instanciação de atributo ou relação. A Figura 33a mostra o indivíduo **João** que instancia as classes **Homem** e **Adulto** e o atributo **idade** da forma tradicional, enquanto na Figura 33b a mesma situação é representada da forma reificada utilizando-se (i) um *quality* que reifica a **idade de João** e ao qual então é atribuído o valor da sua idade, e que é inerente a ele; e (ii) um *qua-individual* que reifica a fase **Adulto** de **João**, e que, portanto, é também inerente a ele.

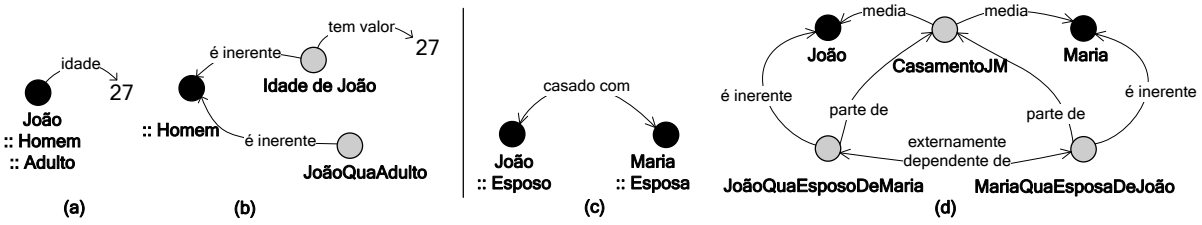


Figura 33. Esquema ilustrativo comparativo entre a representação de atributos, papéis e relações materiais da forma tradicional (a, c) e a forma reificada (b, d).

Analogamente, a Figura 33c mostra o indivíduo **João** que instancia a relação material **casado-com** e a classe/papel **Esposo**, e o indivíduo **Maria** que instancia a relação material **casado-com** e a classe/papel **Esposa** da forma tradicional. Por outro lado, na Figura 33d, a mesma situação é representada da forma reificada utilizando-se (i) o *relator* **CasamentoJM** que reifica a relação material entre **João** e **Maria** e que, portanto, os media; (ii) bem como os *qua-individuals* **JoãoQuaEsposoDeMaria** e **MariaQuaEsposaDeJoão** que reificam os papéis de **João** como **Esposo** e **Maria** como **Esposa** respectivamente, e que, portanto, são inerentes a eles.

Uma vez que papéis são tipos de conceitos anti-rígidos relacionalmente dependentes, eles são reificados como *qua-individuals* também relacionalmente dependentes, que compõem (como partes essenciais e inseparáveis) o *relator* **CasamentoJM** que os contextualiza, e também são externamente dependentes entre si. Neste trabalho, a relação de dependência externa é um tipo especial de dependência existencial que ocorre entre *qua-individuals* mediados pelo mesmo relator. Em contrapartida, fases, que são tipo de conceitos anti-rígidos dependentes apenas de propriedades intrínsecas do indivíduo, são representadas como *qua-individuals* não necessariamente relacionalmente dependentes.

Assim, as propriedades dos indivíduos, uma vez reificadas, podem ser referenciadas, qualificadas e quantificadas. Particularmente, interessa qualificar os indivíduos reificados, bem como os indivíduos não-reificados, com o intervalo de tempo correspondente à sua existência, uma vez que este trabalho tem como foco a representação de informação temporal. Assim, a Figura 37 mostra um esquema ilustrativo desta estratégia, com um exemplo similar ao da Figura 33 visto, porém, numa perspectiva temporal.

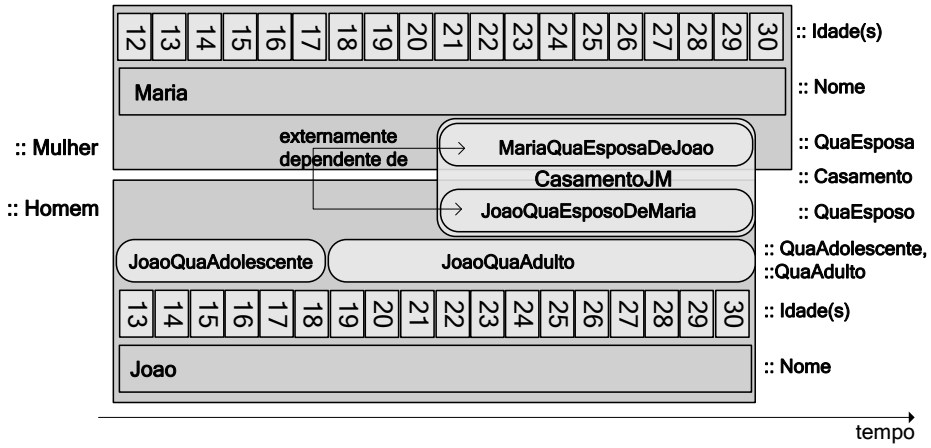


Figura 34. Esquema ilustrativo da abordagem de reificação temporal.

Neste esquema ilustrativo, objetos e *moments* são representados por elementos cuja extensão horizontal corresponde à extensão temporal ou tempo de existência do indivíduo. A inclusão espacial dos elementos representa as relações de dependência existencial entre os indivíduos, ou seja, os elementos espacialmente incluídos são (*moments*) dependentes existencialmente do elemento contenedor, o que também reflete a inclusão temporal imposta pela relação de dependência existencial. Os rótulos precedidos do símbolo **::** próximos aos elementos indicam as classes que eles instanciam. Assim, os elementos mais externos representam as instâncias do tipo objeto, a saber, uma instância de **Mulher** e uma instância de **Homem**. Além disso, enquanto as propriedades necessárias são representadas por retângulos, as contingentes são representadas por elipses, e as mutáveis são representadas com cor de fundo cinza mais claro.

O atributo **nome** é representado como *qualities,* instâncias de **Nome**, cujo valor é **Maria** para a instância de **Mulher** e **João** para a instância de **Homem**. Sua extensão temporal é igual à dos objetos qualificados, uma vez que se trata de um atributo necessário e imutável. A **idade** também é representada como *qualities*, instâncias de **Idade**, que, porém, é instanciado diversas vezes para cada objeto, uma para cada mudança em seu valor. Neste caso, a "soma" das extensões temporais destes *qualities* é igual à dos objetos qualificados, uma vez que este é um atributo necessário. Se o atributo é contingente, deve apenas estar contido (temporalmente) no objeto qualificado.

A relação material **casado-com** entre os indivíduos **João** e **Maria** é representada pelo *relator* **CasamentoJM**, instância de **Casamento**, que reifica a relação entre os indivíduos, e, portanto, os media. Assim, ele é representado por um elemento que conecta os dois indivíduos, e está horizontalmente incluído em ambos. Os respectivos papéis de **Esposo** e **Esposa** são reificados como os *qua-individuals* relacionais **JoaoQuaEsposoDeMaria** e **MariaQuaEsposaDeJoao**, que são instâncias de **QuaEsposo** e **QuaEsposa** respectivamente, e são externamente dependentes entre si, inerentes a **João** e **Maria** respectivamente, e também partes (essenciais e inseparáveis) do *relator* **CasamentoJM**. Por isso, estão contido (temporalmente) nos indivíduos **João** e **Maria** e no *relator* **CasamentoJM**.

Finalmente, as fases **Adolescente** e **Adulto** são reificadas como os *qua-individuals* fasais **JoaoQuaAdolescente** e **JoaoQuaAdulto** que são instâncias de **QuaAdolescente** e **QuaAdulto** respectivamente. Ontologicamente, fases são representadas como um conjunto de classes disjuntas e completas que um indivíduo instancia necessariamente conforme variações em suas propriedades intrínsecas, por exemplo, **Morto** e **Vivo**. Assim, por um lado, os *qua-individuals* referentes a um conjunto de fases não podem ser sobrepostos temporalmente, e, por outro lado, deve sempre haver um destes *qua-individuals* inerente ao indivíduo. Seguindo o exemplo citado, o indivíduo **João** não pode ter ambos *qua-individuals* **JoãoQuaVivo** e **JoãoQuaMorto** inerentes a ele simultaneamente, mas sempre deve ter exatamente um deles em cada momento de sua existência.

Assim, tem-se a estratégia de reificação ontologicamente fundamentada para a representação de informação temporal.

3.2.2. Abordagem de Reificação Temporal

A abordagem para representação de informação temporal apresentada nesta seção é baseada na estratégia de reificação temporal apresentada na seção anterior, que consiste em atribuir uma extensão temporal aos indivíduos e suas propriedades objetificadas, denotando seu tempo de existência. Excetuando-se os conceitos rígidos/necessários, que se aplicam diretamente e

necessariamente aos indivíduos, e as relações formais internas, que neste trabalho implicam em dependência existencial (ver seção 2.1.1), as outras propriedades dos indivíduos do domínio são representadas como *moments* existencialmente dependentes daqueles. Nisto se fundamenta esta abordagem: em representar as propriedades dos indivíduos em função da relação de dependência existencial, interpretando-as como válidas durante a existência do dependente.

A estrutura base OWL para representação desta abordagem é apresentada na Figura 35 como um esquema estilo UML. As instâncias do modelo são do tipo *Individual* (Indivíduo) ou *TemporalExtent* (Extensão Temporal), de forma que cada indivíduo tem uma extensão temporal, através da relação *hasTemporalExtent*. Os **indivíduos** são instâncias de *Object* ou *Moment*. A relação de dependência existencial (*existentiallyDependentOf*) vale entre indivíduos, e as demais relações do modelo (exceto *hasTemporalExtenti* e *hasValue*) são especializações desta (esta informação é omitida do modelo por questão de legibilidade).

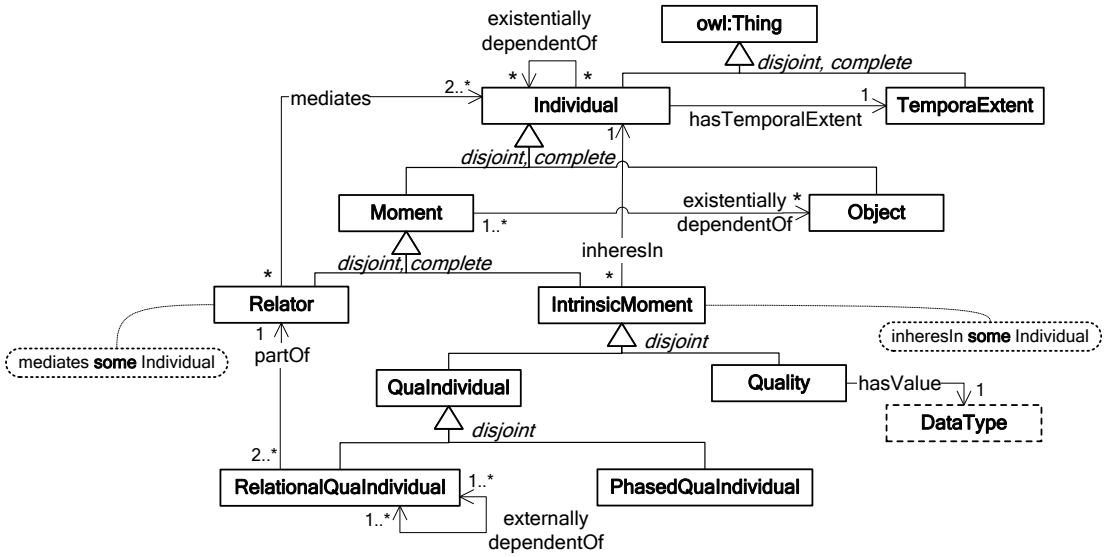


Figura 35. Esquema estilo UML da estrutura base OWL para a abordagem de reificação temporal.

Um ***moment***, que pode ser do tipo *IntrinsicMoment* ou *Relator*, é tal que sempre é existencialmente dependente de pelo menos um indivíduo. O primeiro tipo é inerente (*inheresIn*) a exatamente um indivíduo, e é definido como qualquer instância que é inerente a algum indivíduo (*inheresIn some Individual*), enquanto o segundo media (*mediates*) pelo menos dois indivíduos, e é definido como qualquer instância que media algum indivíduo (*mediates some Individual*). Por outro lado, seguindo-se a afirmação de Bunge de que não deve haver indivíduos do domínio sem propriedades (BUNGE, 1977 apud GUIZZARDI, 2005), um **objeto** deve ter pelo menos um *moment* existencialmente dependente dele.

Um *moment* intrínseco é classificado aqui como *Quality* ou *QuaIndividual*. O primeiro possui um único valor num domínio concreto, que é representado por uma propriedade funcional de tipo de dados (*hasValue*). O segundo, por sua vez, pode ainda ser classificados como fasais ou como relacionais, sendo que os últimos são parte de (*partOf*) exatamente um *relator* e são externamente dependentes dos outros *qua-individuals*.

As seguintes diretrizes gerais são definidas para guiar o uso adequado desta estrutura, e, consequentemente, da abordagem:

(d1)	os **conceitos necessários/rígidos** do domínio são representados como subclasses (diretas ou indiretas) da classe *Object*. Por exemplo, o conceito **Pessoa** é representado como subclasse direta de *Object,* enquanto os conceitos **Homem** e **Mulher** são representados como a subclasses de **Pessoa**, especializando indiretamente a classe *Object*;
(d2)	os **conceitos contingentes/anti-rígidos**, reificados, são representados como subclasses de *QuaIndividual*, que agrupam os *qua-individuals* resultantes de sua reificação;
(d2.1)	as **fases** são representadas como subclasses de *PhasedQuaIndividual*. Por exemplo, a fase **Adulto** deve ser representada como a classe **QuaAdulto**, que agrupa os *qua-individuals* resultantes de sua reificação;
(d2.2)	os **papéis** são representados como subclasses de *RelationalQuaIndividual*. Por exemplo, o papel **Esposo** deve ser representado como a classe **QuaEsposo**, que agrupa os *qua-individuals* resultantes de sua reificação;
(d2.3)	cada sub-classes de *QuaIndividual* é tal que suas instâncias são inerentes a exatamente um indivíduo do tipo *object*, instância da super-classe rígida mais próxima do conceito contingente em questão. Por exemplo, as instâncias da classe **QuaEsposo** são inerentes a exatamente uma instância da classe **Homem**;
(d2.4)	se o conceito contingente em questão for especialização de outro conceito contingente, então os indivíduos reificados daquele são existencialmente dependentes de exatamente um indivíduo reificado deste. Por exemplo, se o conceito **Esposo** fosse especialização de um outro conceito contingente **Cônjuge**, as instâncias da classe **QuaEsposo** seriam existencialmente dependentes de exatamente uma instância da classe **QuaConjuge**;
(d3)	as **relações formais internas**, que implicam em dependência existencial, são representadas diretamente:
(d3.1)	se domínio e imagem são conceitos rígidos, então a relação vale diretamente entre eles. Por exemplo, a relação formal **parte-de** entre **Pessoa** e **Coração**, que são ambos rígidos, vale diretamente entre eles;
(d3.2)	se domínio ou imagem são conceitos anti-rígidos (contingentes), então tornam-se domínio ou imagem da relação os super-conceitos rígidos mais próximos na hierarquia; Por exemplo, se uma instância de **Cérebro** fosse existencialmente dependente de uma instância do conceito contingente **PessoaViva**, que especializa **Pessoa**, então a relação deveria valer entre **Cérebro** e **Pessoa**;
(d3.3)	se a relação é de dependência existencial unilateral, e é mutável na direção inversa, então a cardinalidade máxima nesta direção é flexibilizada. Por exemplo, a relação **parte-de**

	entre **Pessoa** e **Coração** é de dependência existencial apenas na direção **Coração→Pessoa**. Entretanto, na direção contrária, uma instância de **Pessoa** pode se relacionar com mais de uma instância de **Coração** durante sua existência. Assim, como a restrição de cardinalidade máxima 1 não se aplica à toda a existência de uma instância de **Pessoa**, mas a instantes temporais, ela deve ser flexibilizada para permitir vários relacionamentos deste tipo.
(d4)	as **relações materiais,** reificadas, são representadas como subclasses de *Relator*, e agrupam os indivíduos resultantes de sua reificação. Por exemplo, a relação **casado-com** é representada como a classe **Casamento**, que agrupa todos os *relators* resultantes de sua reificação;
(d4.1)	cada *relator* media instâncias dos conceitos rígidos que os papéis envolvidos na relação material especializam. Por exemplo, uma instância de **Casamento** media uma instância da classe **Homem** e uma da classe **Mulher**, que são os conceitos rígidos que os papéis **Esposo** e **Esposa**, envolvidos na relação material **casado-com**, especializam.
(d4.2)	cada *relator* é composto pelos *qua-individuals* correspondentes aos papéis exercidos pelos indivíduos envolvidos na relação material. Por exemplo, cada instância **Casamento**, que é a classe correspondente à reificação da relação material **casado-com**, tem como parte uma instância da classe **QuaEsposo** e uma de **QuaEsposa**, correspondentes aos papéis **Esposo** e **Esposa** envolvidos na relação material;
(d4.3)	cada *qua-individual* correspondente a um papel exercido numa relação material é externamente dependente dos outros *qua-individuals* que compõem o mesmo *relator*. Por exemplo, uma instância da classe **QuaEsposo** é externamente dependente da instância de **QuaEsposa** que compõe com ele um dado *relator* **Casamento** (e vice-versa), uma vez que os papéis **Esposo** e **Esposa** envolvidos na relação material **casado-com**;
(d5)	os **atributos**, reificados, são representados como subclasses de *Quality*, e agrupam os indivíduos resultantes de sua reificação. Por exemplo, o atributo **nome** do conceito **Pessoa** é representado como a classe **Nome** que agrupa todos os *qualities* resultantes de sua reificação.
(d5.1)	se o atributo é mutável, então a cardinalidade máxima relativa à quantidade de *qualities* referentes ao atributo imutável podem ser inerentes ao indivíduo é flexibilizada. Por exemplo, embora uma **pessoa** possa ter apenas uma **idade** por vez, sendo mutável, a cardinalidade máxima é flexibilizada para que, uma vez que a **idade** mude, outros *qualities* com valores diferentes possam ser inerentes à **pessoa**.

Aplicando-se esta abordagem ao modelo de exemplo da Figura 20 tem-se como resultado o modelo OWL ilustrado na Figura 36 (os elementos em cinza não fazem parte do domínio). Enquanto os conceitos rígidos/necessários são representados como especialização da classe *Object*, os atributos são reificados como **Nome**, **Idade**, **Apelido** e **CPF** e representados como especialização da classe *Qualitiy*, os conceitos contingentes **Esposo** e **Esposa** são reificados como **QuaEsposo** e **QuaEsposa** e representados como especialização da classe *RelationalQuaIndividual*, e a

relação material **casado-com** é reificada como **Casamento** e representada como especialização da classe *Relator*.

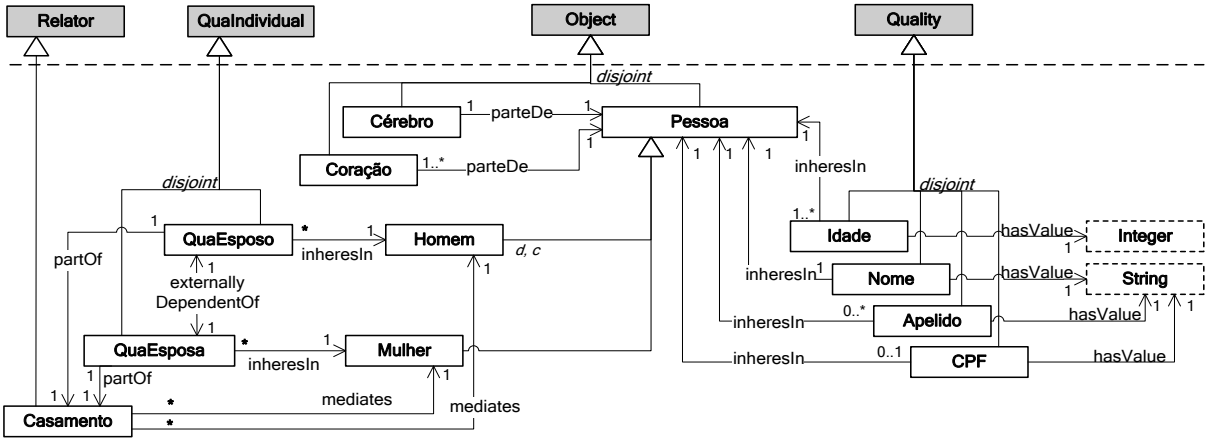


Figura 36. Esquema estilo UML da aplicação da abordagem de reificação ao exemplo da Figura 20.

Um exemplo de instanciação deste modelo é mostrado na Figura 37. Ele ilustra um cenário em que um **homem** é chamado **João**, possui **idades** que variam de **18** a **30 anos** e possui um **cérebro b1**, e a partir dos **19 anos** possui um **cpf "098..."**, aos **23 anos casou-se** com uma **mulher** chamada Maria de **22 anos**, e esteve **casado** até os **30 anos** quando morreu. Possui ainda duas instâncias de **coração h1** e **h2** com projeções temporais disjuntas, sendo que o momento da troca corresponde ao período em que **João** tinha **23 anos**. Destaca-se que todas as relações representadas são de dependência existencial.

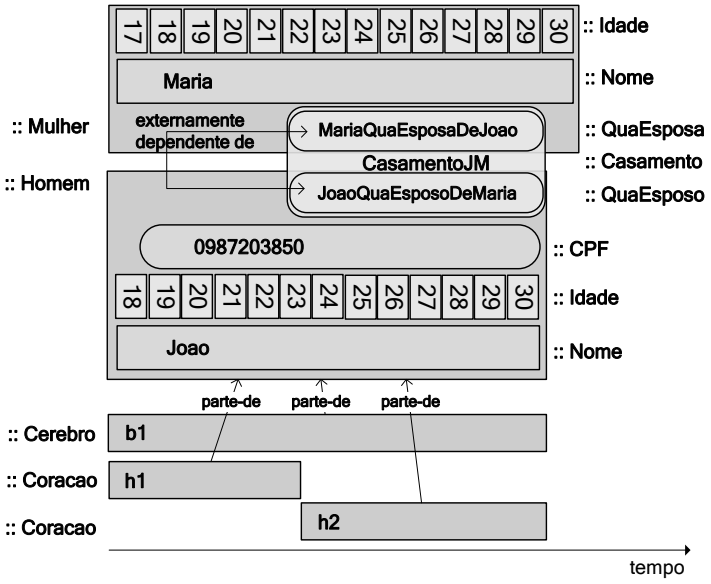


Figura 37. Esquema ilustrativo de instanciação do modelo da **Figura 34**.

Observa-se que as informações contingentes a respeito dos indivíduos não são representadas diretamente, mas indiretamente através dos *moments* que as reificam. Tais informações devem ser

recuperadas através de consultas ao modelo, por exemplo, a relação material **casado-com** entre **João** e **Maria** vale durante a existência do relator **CasamentoJM**. Entretanto, a elaboração destas consultas está fora do escopo deste trabalho.

O resultado da aplicação desta alternativa permite representar a informação temporal conforme a proposta teórica de Reificação Temporal da seção anterior. Contudo, identificam-se alguns problemas:

(i) não são representadas as restrições temporais previstas na teoria, nem permite-se impor restrição de cardinalidade temporal; uma vez que isto não é possível fazer utilizando-se apenas os recursos de OWL, e portanto está fora do escopo deste trabalho;

(ii) pressupõe-se conhecido o tempo de existência de cada indivíduo; ou seja, não está adequada à hipótese de mundo aberto em que novas informações sobre os indivíduos poderiam ser acrescidas ao modelo. Embora este trabalho não trate as restrições temporais previstas na teoria, esta condição sobre o tempo de existência foi imposta para viabilizá-las, ou seja, para permitir tratar a inclusão temporal;

(iii) não se permite inferir, mas apenas consultar as informações contingentes, uma vez que elas são representadas como indivíduos que devem ser qualificados temporalmente e eventualmente relacionados a outros. Por exemplo, não é possível inferir que se **João** é mediado por um *relator* do tipo **Casamento**, então ele tem um indivíduo do tipo **QuaEsposo** inerente a ele que tem a mesma extensão temporal deste *relator*, que é também parte deste, e que é externamente dependente do(s) outro(s) indivíduo(s) mediado(s) pelo mesmo *relator*.

3.2.3. Trabalhos Relacionados

Abordagem de Gangemi

Gangemi (2005) propõe vários padrões para a web semântica baseados na ontologia de fundamentação DOLCE. Um padrão particularmente interessante é o padrão para reificação de relações indexadas no tempo. Este padrão trata especificamente da reificação da relação formal de **participação** de **objetos** em **eventos**. A Figura 38 apresenta um esquema estilo UML ilustrando a utilização de tal padrão para representar a reificação da relação material **casado-com**. A relação é reificada em um **evento** chamado **Casamento**, e então as **participações** de ambos os **objetos Homem** e **Mulher** em tal **evento** são reificadas em duas **situações** do tipo participação indexada no tempo (*time indexed participation*).

Entretanto, esta abordagem não prevê a representação de aspectos temporais como atributos e conceitos contingentes ou mutáveis. Embora seja possível representar os *moments* intrínsecos

como situações indexadas no tempo, tem-se que: (i) esta abordagem não foi desenvolvida com este propósito, e portanto não possui a estrutura e diretrizes apropriadas; (ii) os indivíduos do domínio (**homem**, **mulher**) não são qualificados temporalmente, impossibilitando a imposição das restrições temporais previstas na teoria apresentada de Reificação Temporal.

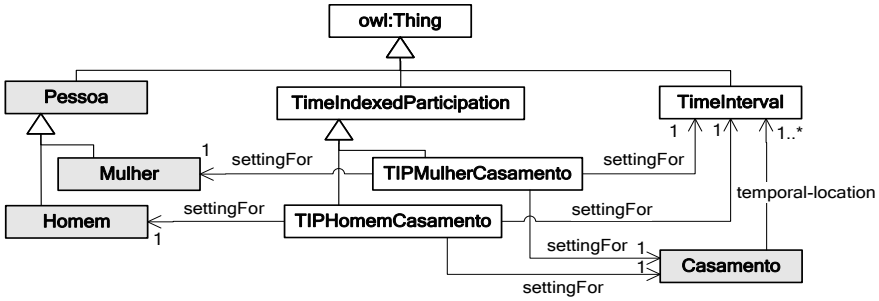


Figura 38. Esquema estilo UML da estrutura OWL para a abordagem proposta por Gangemi.

Abordagem de O'Connor & Das

O'Connor e Das (2011) propõem um método para representação de informação temporal em OWL que visa compatibilidade com ontologias OWL já existentes. A Figura 39 apresenta a estrutura proposta pelos autores. Ela inclui a classe *Fact* que pode ser estendida no tempo, e então compreende tanto fatos temporais quanto atemporais. A propriedade *hasValidTime* (tem tempo válido) vale para o(s) intervalos(s) de tempo durante o(s) qual(is) um certo fato é tido como verdade (os outros elementos não são relevantes a este trabalho). Assim, a classe *Fact* deve ser especializada pelas classes da ontologia.

Entretanto, este método não provê nenhum suporte para decidir o que o como reificar, enquanto simplesmente introduzir uma ontologia existente nesta estrutura não é suficiente, pois, especializando a classe *Fact*, apenas permitiria atribuir aos indivíduos um intervalo de tempo, mas suas propriedades permaneceriam representadas estaticamente. Por exemplo, de nada adiante ter um intervalo de tempo associado a **João** se ele instancia a classe **Esposo** de forma estática, ou seja, uma vez que a instanciou esta informação permanece inalterada.

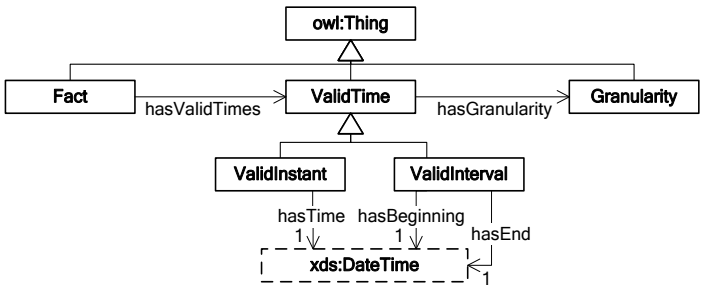


Figura 39. Esquema estilo UML da estrutura OWL para a abordagem de O'Connor e Das.

3.2.4. Considerações e Comparação entre as Abordagens de Reificação Temporal

As abordagens de Reificação Temporal discutidas na seção anterior têm por objetivo a representação de informação temporal em OWL. O método proposto por O'Connor e Das apresenta uma estrutura para atribuir extensão temporal a fatos gerais, mas não provê nenhum suporte para decidir o que e como reificar. De fato, o principal problema dos trabalhos que usam a estratégia de reificação é a falta de compromisso com uma estrutura de fundamentação de alto nível para guiar as decisões de modelagem. Isto tem o indesejável efeito colateral de colocar a responsabilidade das decisões de modelagem nas mãos do usuário. Por outro lado, embora a abordagem de Gangemi baseie-se nas noções dinâmicas de **situação** e **evento** da ontologia de fundamentação DOLCE, ela não está voltada para capturar os aspectos temporais do modelo estrutural, ou seja, não prevê a representação de aspectos como atributos, papéis e fases contingentes ou mutáveis.

Portanto, a abordagem de reificação proposta neste trabalho é considerada mais completa e abrangente uma vez que provê tanto uma estrutura ontologicamente fundamentada quanto diretrizes para guiar sua utilização como o intuito de representar a informação temporal segundo os aspectos temporais do modelo estrutural. Assim, com relação aos aspectos temporais contemplados pela **abordagem de Reificação Temporal**, resumidos na

Tabela 5, tem-se que:

- com relação às **classes**, as **obrigatórias** são representadas como especialização da classe *Object*, enquanto as **contingentes** são reificadas como subclasses de *QuaIndividual*;
- com relação às **propriedades**, tem-se que:

(a) **Relações formais** - são sempre de um dos tipos:

- **Dependência existencial mútua:** são **propriedades obrigatórias** e **imutáveis** para ambos os indivíduos envolvidos;
- **Dependência existencial unilateral:** são **propriedades obrigatórias** e **imutáveis** para o indivíduo dependente. No sentido inverso, ela é **contingente** e/ou **mutável**. Caso seja **obrigatória** no sentido inverso, tal obrigatoriedade não pode ser garantida.. Por exemplo, pode-se garantir que um **coração** é necessariamente **parte-de** uma **pessoa** uma vez que aquele é existencialmente dependente deste; entretanto, não se pode garantir que uma **pessoa** tenha necessariamente um **coração** em todos os momentos. Caso seja imutável no sentido inverso, tal imutabilidade não pode ser garantida;

(b) **Relações materiais**: assume-se que são **propriedades contingentes** com relação às classes obrigatórias envolvidas, mas obrigatórias para os respectivos papéis. Por exemplo, a relação de **casado-com** é contingente para **Pessoa**, mas obrigatória para **Esposo** e **Esposa**. São representadas como *relators*, *qua-individuals* e as relações de inerência, mediação, parte de entre *qua-individuals* e *relator* e dependência externa entre *qua-individuals*.

- Se forem **imutáveis**, restringe-se a cardinalidade máxima relativa à quantidade de *relators* referentes a esta relação material que podem mediar os indivíduos envolvidos. Por exemplo, supondo que uma **pessoa** pode casar-se apenas uma vez, restringe-se a cardinalidade com relação ao *relator* **casamento**, de forma que uma pessoa possa ser mediada por apenas um *relator* deste tipo durante sua existência.
- Se forem **mutáveis**, flexibiliza-se a cardinalidade máxima relativa à quantidade de *relators* referentes a esta relação material que podem mediar os indivíduos envolvidos. Por exemplo, supondo que uma **pessoa** pode ter apenas um casamento por vez, flexibiliza-se a cardinalidade com relação ao *relator* **casamento**, de forma que uma pessoa possa ser mediada por mais de *relator* deste tipo durante sua existência. Entretanto, não se pode garantir que seja apenas um por vez.

(c) **Atributos:** são representados como *qualities*

- **obrigatórios:** a obrigatoriedade não pode ser garantida, uma vez que requereria que a extensão temporal do *quality* fosse igual à do indivíduo ao qual ele é inerente.
- **contingentes:** são *qualities* cuja extensão temporal é menor que a do indivíduo ao qual ele é inerente;

- **imutáveis:** restringe-se quantos qualities podem ser inerentes ao indivíduo. Por exemplo, uma vez que os atributos **nome** e **CPF** são únicos e imutáveis, restringe-se para 1 a cardinalidade máxima relativa à quantidade de qualities referentes a estes atributos que podem ser inerentes a uma **pessoa**;
- **mutáveis:** exigem flexibilização da cardinalidade máxima relativa à quantidade de qualities referentes ao atributo imutável podem ser inerentes ao indivíduo. Por exemplo, embora uma **pessoa** possa ter apenas uma **idade** por vez, sendo mutável, a cardinalidade máxima é flexibilizada para que, uma vez que a **idade** mude, outros *qualities* com valores diferentes possam ser inerentes à **pessoa**.

(d) **Dependência genérica:**

- **unilateral**: podem ser as relações de dependência existencial unilateral obrigatórias e mutáveis no sentido inverso, ou relações materiais obrigatórias e mutáveis num só sentido;
- **mútua**: relações materiais obrigatórias e mutáveis em ambos os sentidos;

Tabela 5. Aspectos temporais contemplados pela abordagem de Reificação Temporal.

Abordagem	Classe		Propriedades				Dependência			
							genérica		existencial	
	obrig.	cont.	obrig.	cont.	mut.	imut.	uni-lat	mútua	uni-lat	mútua
Reificação Temporal	+	+	+/-	+	+	+	-	+	-	+

Segundo Welty e Fikes (2006), um problema da abordagem de reificação em OWL é que, ao reificar as relações, perde-se a habilidade de representar as meta-propriedades, como simetria, reflexividade, transitividade e funcionalidade. Entretanto, nesta abordagem, tal problema é reconsiderado em função da aplicabilidade de tais meta-propriedades às relações materiais, que são aquelas reificadas:

(i) **Reflexividade**: uma vez que um relator é um tipo especial de *moment* que deve mediar mais de uma entidade, não pode haver relações materiais reflexivas (GUIZZARDI, 2005). Em outras palavras, se uma relação material **MR** é reflexiva e **R** é o *relator* do qual **MR** deriva, então **MR(a,a)** vale e é derivado de **r** tal que **R(r)** media apenas uma entidade, a saber, **a**. Isso leva a uma contradição com a definição de relator. Portanto, não esta meta-propriedade não é necessária para relações materiais.

(ii) **Simetria**: uma relação material é dita intencionalmente simétrica sss a classe *relator* **R**, a partir da qual **MR** é derivada, é tal que cada instância de **R** media apenas entidades que exercem o mesmo papel (GUIZZARDI, 2005). Por exemplo, a relação material **casado-com** não pode ser dita simétrica a não ser que os indivíduos mediados exerçam o mesmo papel, neste caso, **cônjuges**. Assim, a simetria não é representada como uma meta-propriedade OWL, mas está implícita na estrutura de representação de forma que **MR(a, b)** e **MR(b, a)** são ambos reificados da mesma forma e, consequentemente, são derivados da mesma informação reificada. Por exemplo, considerando-se o papel comum *Cônjuge*, os fatos **casado-com(João, Maria)** e **casado-com(Maria, João)** são representados da mesma forma, a saber, **mediates(Casamento, João)** e **mediates(Casamento, Maria)**, e são igualmente derivados a partir da mesma informação reificada.

(iii) Transitividade: uma vez que a instanciação de uma relação material requer existência de um *relator* que o fundamente, o mesmo acontece para uma relação material derivada transitivamente (GUIZZARDI, 2009). Assim, a relação material transitiva não deve ser representada simplesmente como uma meta-propriedade OWL, mas deve impor a existência do *relator* e das relações de mediação. Por exemplo, se **MR(a,c)** é derivada de **MR(a,b)** e **MR(b,c)**, que são fundamentadas em seus respectivos *relators*, deve-se inferir a existência de um *relator*

AC que media **a** e **c** e que tenha a mesma extensão temporal dos outros *relators*. Isso não é possível derivar usando apenas a linguagem OWL. Portanto, neste caso, a representação da transitividade realmente se perde.

(iv) **Funcionalidade**: uma relação material funcional é tal que um indivíduo do domínio pode estar relacionado com apenas um indivíduo da imagem por vez. Isto é representado nesta abordagem por restrições de cardinalidade da seguinte forma: (i) se a relação é imutável, então o indivíduo do domínio deve ser mediado por no máximo uma instância do *relator* **R**, que deve mediar exatamente um indivíduo da imagem; (ii) se a relação é mutável, então um indivíduo do domínio deve ser mediado por mais de um relator **R**, o qual deve mediar exatamente um indivíduo da imagem. Neste último caso, porém, não é possível exigir que todos os *relators* sejam temporalmente disjuntos, logo, a funcionalidade também é perdida. Por exemplo, se um **esposo** pode **estar casado com** uma única **esposa** de forma imutável, tem-se que **João** pode ser mediado por um único relator **Casamento** que por sua vez pode media uma única **esposa**, **Maria**. Por outro lado, se a relação for mutável, então deve-se flexibilizar a restrição de cardinalidade de forma que João possa ser mediado por mais de um relator **Casamento**, que por sua vez pode media uma única **esposa**.

Além disso, Vieu et. al (2008) afirmam que uma abordagem baseada em reificação, especialmente no que diz respeito aos *qua-individuals*, é mais apropriada para lidar com questões como o paradoxo de propriedades conflitantes e o problema da contagem do que a abordagem perduranista 4D ou uma abordagem baseada em eventos. Os autores dão um exemplo que explica as duas questões: suponha que Berlusconi participe de uma reunião industrial como Primeiro Ministro Italiano e como Presidente da Mediaset. Por um lado, pode-se facilmente imaginar a possibilidade de ele ter diferentes propriedades (por exemplo, direitos e deveres) como primeiro ministro e como presidente de uma companhia, eventualmente contraditórias. Essa questão não pode ser resolvida nem por uma estratégia baseada em eventos, uma vez que as propriedades inconsistentes não se aplicam aos eventos, mas a pessoas, nem por uma estratégia perdurantista 4D, já que ambas propriedades inconsistentes se aplicariam à mesma parte temporal de Berlusconi. Por outro lado, se devessem ser contados os representantes presentes na reunião, novamente não pode-se usar nem a estratégia baseada em eventos, nem a perdurantista 4D, uma vez que ambas resultariam em contar o Berlusconi apenas uma vez[10].

Entretanto, deve-se considerar uma peculiaridade do problema da contagem no contexto de OWL. Na verdade, não se pode contar nada sem considerar a hipótese de nomes únicos (HNU),

[10] Para mais detalhes,sobre o assunto outras duas referências são (MASOLO ET AL., 2005; GUIZZARDI, 2006).

ou seja, se uma única entidade pode ser representada por mais de um indivíduo OWL, não faz sentindo contá-los, a não ser que seja possível garantir que eles são distintos. Apesar de muitos raciocinadores OWL permitirem forçar HNU, esta não deve ser tomada como uma solução final, uma vez que aumenta exponencialmente o número de asserções na ABox.

Ademais, para resolver o problema do paradoxo das propriedades conflitantes, deve-se primeiramente trazer as propriedades para o contexto dos *qua-individuals*. Masolo et. al reconhecem esta questão como a necessidade de um argumento adicional na propriedade referindo-se ao *qua-individual*, e a apontam como um problema de decisão de modelagem como segue: "como se pode saber a priori quais predicados devem permitir um argumento adicional? Este problema é uma questão para investigação futura" (MASOLO ET AL., 2005, p. 8). No contexto deste trabalho, esta questão torna-se um problema de decidir quais propriedades devem valer para os *qua-individuals* ao invés de valer para um indivíduo, ou ainda se devem valer para ambos. Enfim, apesar de a estratégia de reificação teoricamente resolver o problema da contagem e o paradoxo das propriedades conflitante, na prática deve-se levar em consideração outros aspectos, como as características da linguagem, a eficiência computacional ou ainda a complexidade estrutural do modelo.

3.3. COMPARAÇÃO ENTRE AS ABORDAGENS 4D E DE REIFICAÇÃO TEMPORAL

O Quadro 4 resume e compara as principais características das duas abordagens, apontando dentre elas qual é a mais indicada para cada caso. Em suma, se por um lado a abordagem 4D propõe uma representação da informação temporal de forma flexível e intuitiva, por outro a abordagem de reificação temporal mostra-se mais escalável e ontologicamente adequada.

Características		**Abordagem**
4D	**Reificação Temporal**	**Recomendada**
ESCALABILIDADE		
Proliferação de partes temporais em decorrência de cada mudança (no pior dos casos, em que todas as partes temporais são interligadas, uma mudança em uma única parte temporal provoca a duplicação de todas elas).	Aumento do número de indivíduos por causa da reificação (cada propriedade reificada requer um indivíduo a mais, e.g. a representação da idade de João).	**Reificação Temporal**
Repetição tediosa das informações imutáveis no nível TS.	Nenhuma informação é repetida. (contudo as propriedades mutáveis requerem uma nova instância reificada para cada novo valor a elas atribuído)	**Reificação Temporal**

ADEQUAÇÃO ONTOLÓGICA		
Interpretação ontológica estranha de conceitos contingentes (instâncias desses conceitos existem apenas como partes temporais).	Todas as informações representadas têm fundamentação ontológica.	**Reificação Temporal**
GARANTIA DE IMUTABILIDADE DA INFORMAÇÃO		
Não é possível garantir a imutabilidade das informações imutáveis representadas no nível TS.	As informações imutáveis são representadas univocamente.	**Reificação Temporal**
CARDINALIDADE TEMPORAL		
É possível restringir a cardinalidade das propriedades por parte temporal.	Não é possível restringir cardinalidade temporal.	**Perdurantista 4D**
INFERÊNCIA DE INFORMAÇÕES CONTINGENTES		
É possível inferir informações contingentes.	Não é possível inferir, apenas consultar, informações contingentes.	**Perdurantista 4D**
HIPÓTESE DE MUNDO ABERTO		
É possível estender o tempo de existência dos indivíduos, uma vez que ele é a soma das extensões temporais das partes temporais.	Pressupõe-se conhecer o tempo de existência do indivíduo a priori.	**Perdurantista 4D**

Quadro 4. Quadro comparativo de características das abordagens 4D e de Reificação Temporal.

Conclui-se novamente que cada uma das alternativas é aplicável uma vez compreendidas suas características e limitações. A escolha de uma delas deve levar em consideração os requisitos da aplicação. Todavia, uma característica importante, que não é abordada neste trabalho é a eficiência computacional. Este é um ponto bastante complexo e não foi abordado neste trabalho por uma questão de limitação de escopo (ARTALE; CALVANESE; ET AL., 2010).

Outra uma limitação destas abordagens é o tratamento da representação e raciocínio sobre o tempo, também por uma questão de limitação de escopo. Esta característica é necessária para impor restrições temporais, por exemplo, (i) restrição de cardinalidade temporal; (ii) inclusão temporal do indivíduo dependente da relação de dependência existencial na extensão temporal do indivíduo do qual ele depende; e (iii) coincidência temporal dos indivíduos com suas propriedades reificadas necessárias e imutáveis. Uma possível solução seria utilizar extensões de OWL que tratam da representação do tempo e suas relações (HOBBS; PAN, 2004).

Entretanto, para lidar com essas questões deve-se também levar em conta como a temporalidade se reflete nas hipóteses de mundo aberto e nomes não-únicos. Em outras palavras, deve-se considerar que um mesmo objeto do domínio pode ser representado tanto por indivíduos temporalmente sobrepostos quanto por indivíduos temporalmente não-subsequentes (devido a falta de informação num certo intervalo de tempo). Por exemplo, para restrição de cardinalidade temporal, tem-se que se uma relação funcional liga um indivíduo a outros temporalmente

sobrepostos, então se conclui que estes representam o mesmo objeto do domínio. Entretanto, se a relação não é funcional, nada se pode inferir (a não ser que os indivíduos temporalmente sobrepostos possam ser determinados como diferentes). Por exemplo, se um esposo pode ser casado com duas esposas por vez, e tem-se que João é casado com três indivíduos temporalmente sobrepostos, não se pode decidir se eles representam uma única esposa, ou ainda duas, mas uma delas sendo representada duas vezes.

Finalmente, combinando-se as duas abordagens, como mostrado na seção a seguir, obtêm-se alternativas híbridas interessantes que resolvem ou amenizam alguns dos problemas identificados.

3.4. ABORDAGENS HÍBRIDAS

Se por um lado a abordagem de reificação transforma as propriedades em *moments* com suas respectivas relações de dependência existencial, por outro lado a abordagem 4D-A2 é tal que promove todas as relações de dependência existencial para o nível "estático". Em ambos os casos, tais relações são interpretadas como válidas durante o tempo de existência do dependente. Assim, esta seção apresenta de forma sucinta alternativas híbridas, acrescentando-se à abordagem 4D-A2 a reificação de uma ou outra propriedade, e apontando qual a vantagem obtida. No caso extremo, em que todas as propriedades são reificadas, tem-se a estrutura da abordagem de reificação no nível estático, porém com a extensão temporal representada no nível dinâmico.

Alternativa H1

Na primeira alternativa híbrida, reificam-se as relações materiais, que ocorrem entre as partes temporais no nível dinâmico. Dessa forma, uma vez que as outras relações, ditas formais internas, já são representadas no nível estático segundo a alternativa 4D-A2, elimina-se o problema de proliferação de partes temporais, que é justamente decorrente da interligação das partes temporais. Um esquema ilustrativo desta alternativa é apresentado na Figura 40, seguindo o modelo de exemplo da Figura 20.

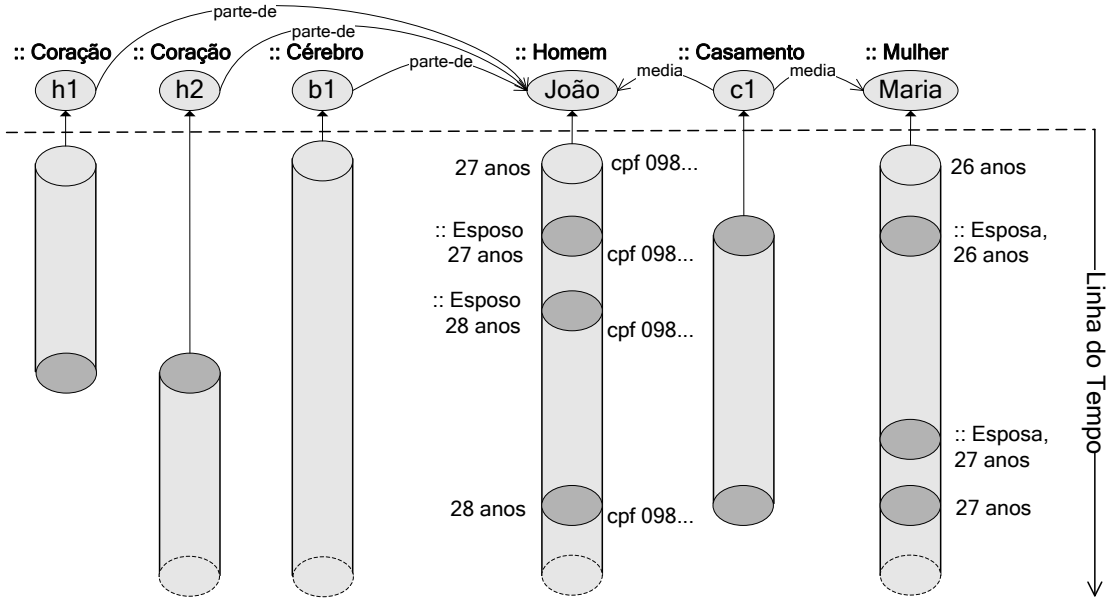


Figura 40. Esquema ilustrativo da alternativa H1.

Observa-se que a relação **casado-com** entre **João** e **Maria** não é representada diretamente, mas reificada pelo *relator* **Casamento**. A relação material pode então ser derivada da existência do relator da seguinte forma: **João** está **casado-com Maria** (e vice-versa) durante o tempo de existência do *relator* **Casamento** que os media. Assim, as mudanças ocorridas nos atributos de **João** e **Maria** não provocam mais a proliferação de partes temporais.

Alternativa H2

Na segunda alternativa híbrida, reificam-se os atributos contingentes e ou mutáveis dos indivíduos. Dessa forma, evita-se (i) a repetição tediosa daqueles imutáveis no nível dinâmico; (ii) a proliferação no nível dinâmico em decorrência de mudanças nos valores dos atributos. Um esquema ilustrativo desta alternativa é apresentado na Figura 41, seguindo o modelo de exemplo da Figura 20.

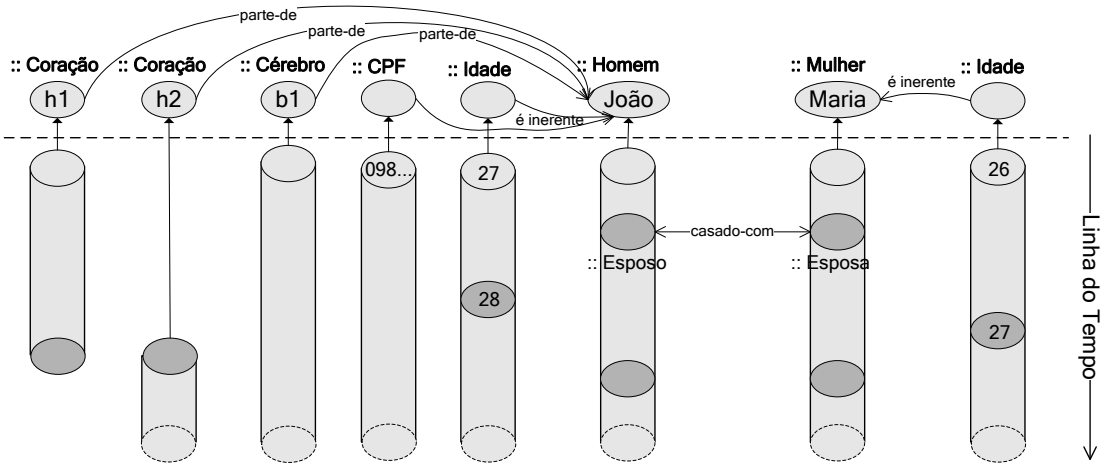


Figura 41. Esquema ilustrativo da alternativa H2.

Observa-se que os atributos **CPF** e **idade** não são mais representados diretamente como propriedades de dados, mas reificados como qualities inerentes a **João**, aos quais são atribuídos os devidos valores. Assim, tem-se que a mudança nos valores dos *qualities*, que estão relacionados apenas no nível estático, não provocam proliferação de partes temporais. Além disso, o atributo imutável **CPF** não precisa ser repetido a cada nova parte temporal de **João**.

Alternativa H3

Na terceira alternativa híbrida, reificam-se os conceitos contingentes, eliminando-se o problema da interpretação ontológica estranha de tais conceitos. De fato, a instanciação destes não ocorre mais apenas por partes temporais, mas é derivada da existência dos respectivos *qua-individual*. Entretanto, não se contribui para reduzir a proliferação, uma vez que a instanciação dos conceitos contingentes é decorrente da alteração de alguma propriedade, interna ou externa, que por si só já causa a proliferação de partes temporais. Um esquema ilustrativo desta alternativa é apresentado na Figura 42, seguindo o modelo de exemplo da Figura 20.

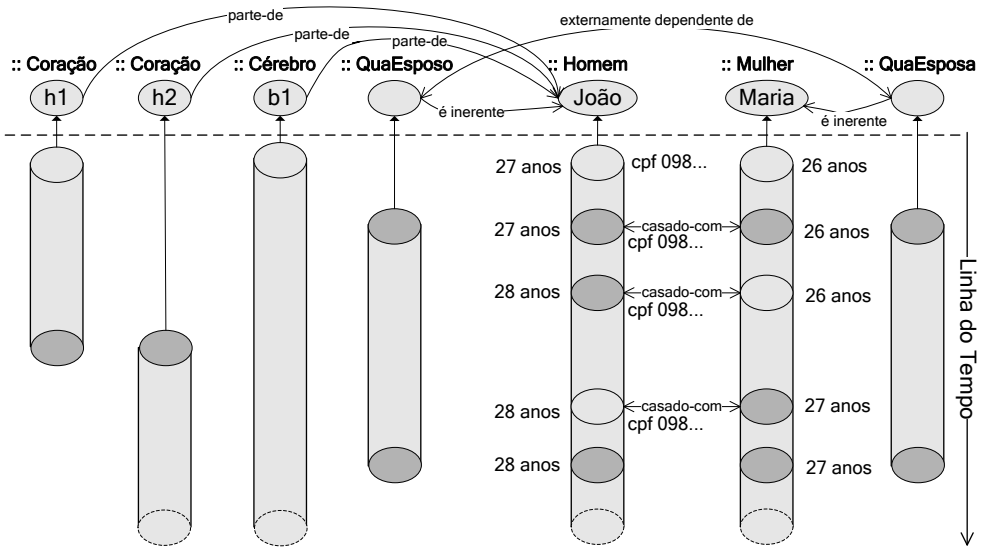


Figura 42. Esquema ilustrativo da alternativa H3.

Observa-se que os papéis **Esposo** e **Esposa** não são mais representados diretamente, mas reificados como *qua-individuals* relacionais inerentes a **João** e **Maria**. A instanciação dos papéis pode então ser derivada da existência dos *qua-individuals* da seguinte forma: **João** instancia a classe **Esposo** (e **Maria** a classe **Esposa**) sempre que existir algum *qua-individual* do tipo **QuaEsposo** (**QuaEsposa** para **Maria**) inerente a ele (ela).

3.5. CONCLUSÕES DO CAPÍTULO

Este capítulo apresenta duas abordagens para representação de informação temporal em OWL, baseadas respectivamente na visão Perdurantista 4D e na Reificação Temporal das propriedades do indivíduo. A abordagem 4D dá origem a três alternativas 4D. Além disso, verifica-se que a alternativa 4D-A2 pode ser mesclada com a abordagem de Reificação Temporal, dando origem a várias alternativas híbridas, das quais são destacadas três. Vale destacar que, embora tenham sido concebidas originalmente no contexto da linguagem OWL, tais abordagens podem ser diretamente estendidas a outras linguagens também baseadas em lógica de descrição.

As principais contribuições apresentadas neste capítulo são: (i) provimento de fundamentação ontológica para cada abordagem; (ii) proposta de alternativas de representação de informação temporal em OWL, cada uma consistindo de uma estrutura base ontologicamente fundamentada e diretrizes para guiar as decisões de modelagem; (iii) identificação das características das abordagens/alternativas e das suas conseqüências; (iv) comparação das abordagens; (v) identificação de alternativas híbridas.

Contudo, observa-se que a aplicação das abordagens não é trivial uma vez que se deve levar em consideração muitas características a respeito da informação do domínio que se pretende modelar. Assim, uma alternativa a isto é incorporar tais abordagens num processo de engenharia de ontologia, sistematizando-as em alternativas de mapeamento, partindo-se de uma linguagem expressiva que capture adequadamente os aspectos temporais do modelo estrutural. Desta forma, a aplicação destes mapeamentos sobre um modelo nesta linguagem resulta em modelos em uma linguagem alvo (por exemplo, OWL) aderente às abordagens. O capítulo seguinte discute a implementação da alternativas de mapeamento apresentadas neste capítulo, seguindo-se a estratégia mencionada, para o caso de mapeamentos da linguagem de modelagem conceitual OntoUML para a linguagem de codificação OWL. Contudo, as alternativas híbridas não são mapeadas por uma questão de restrição do escopo do trabalho.

4. Alternativas de Mapeamento de Modelos OntoUML em Modelos OWL

4. ALTERNATIVAS DE MAPEAMENTO DE MODELOS ONTOUML EM MODELOS OWL

Este capítulo apresenta alternativas de mapeamento de um modelo de referência descrito na linguagem de nível ontológico OntoUML, ou seja, sem preocupações tecnológicas, em modelos dependentes de tecnologia, codificados na linguagem OWL, conforme descrito na Figura 43. Cada especificação de mapeamento consiste de uma estrutura base em OWL e de diretrizes que orientam a especialização desta a partir das entidades do modelo de referência. Uma vez que a linguagem de origem é fundamentada na ontologia de fundamentação UFO-A, também as diretrizes e a estrutura base de cada alternativa são embasadas nesta teoria.

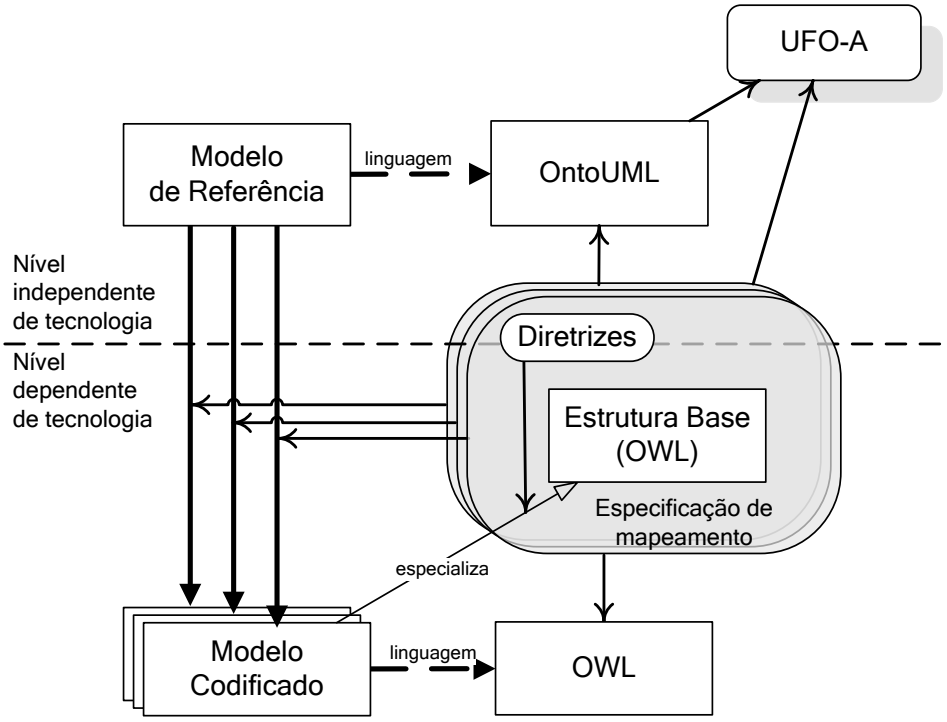


Figura 43. Esquema ilustrativo da sistematização dos mapeamentos.

Primeiramente são definidas duas alternativas de mapeamento, sem considerar temporalidade, que servem então de infra-estrutura para os mapeamentos que visam representar a informação temporal segundo as abordagens apresentadas no capítulo 3. Contudo, estas duas alternativas, definidas na seção 4.1, têm um propósito por si só de prover a sistematização de mapeamentos de modelos OntoUML para a linguagem OWL de duas formas: (i) **cenário estático simples**, em que as entidades do domínio são representadas diretamente, conforme os estereótipos da linguagem OntoUML e (ii) **cenário estático reificado,** em que se considera a representação reificada das entidades do domínio, também conforme mesmos os estereótipos. Os modelos resultantes destes mapeamentos, embora não representem temporalidade, preservam os

aspectos ontológicos do modelo de referência que não são modais, como a hierarquia de conceitos

Por sua vez, as seções 4.2 e 4.3 apresentam duas propostas de sistematização de mapeamento de modelos OntoUML para a linguagem OWL, permitindo-se representar a informação temporal, respectivamente da seguinte forma: (i) **cenário dinâmico simples**, que estende a proposta de mapeamento para cenário estático simples segundo a abordagem Perdurantista 4D e (ii) **cenário dinâmico reificado**, que estende a proposta de mapeamento para cenário estático reificado segundo a abordagem de Reificação Temporal.. Finalmente, a seção 4.4 apresenta o sistema que implementa as especificações de mapeamento definidas neste capítulo, e as conclusões deste estão na seção 4.5.

Para ilustrar os mapeamentos, usa-se o mesmo exemplo da Figura 20 do capítulo 3, porém modelado utilizando-se a linguagem OntoUML com seus os estereótipos e seguindo-se seus postulados, conforme apresentado na Figura 44. Além da representação explícita das categorias ontológicas, a principal diferença dá-se na modelagem da relação **casado-com**. Esta relação é representada como uma relação do tipo *material* uma vez que para valer ela depende de um interventor, o relator **Casamento**, ou seja, ela não vale pela simples existência dos indivíduos relacionados.

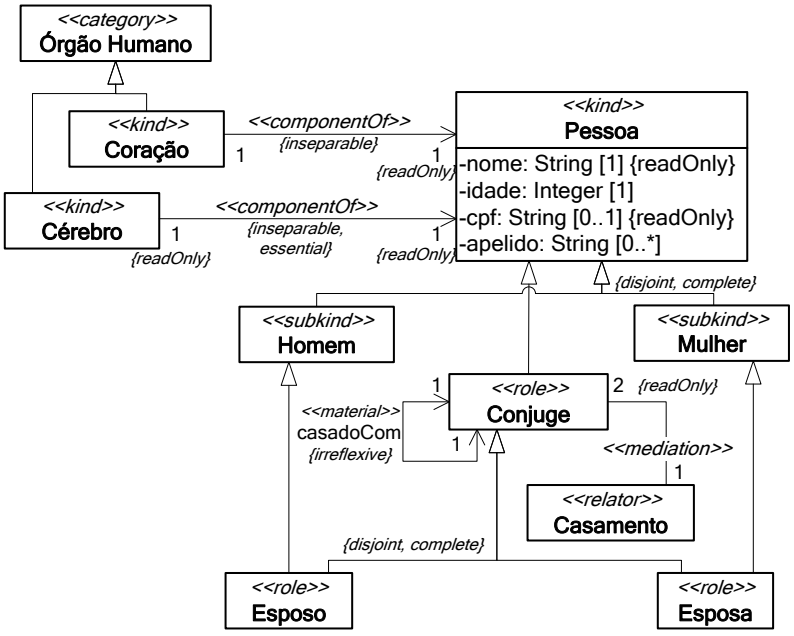


Figura 44. Modelo de exemplo na linguagem OntoUML.

Tais indivíduos desempenham um papel no contexto desta relação, que seriam a princípio os papéis de **Esposo** e **Esposa**, representados como classes do tipo *role*. Entretanto, uma vez que a relação é suposta simétrica, ela deve valer entre indivíduos que desempenham o mesmo papel. Neste caso, tal suposição requer que os papéis de **Esposo** e **Esposa** sejam vistos como idênticos, a menos do sexo da pessoa que o exerce. Estes papéis são então representados como especializações do papel comum de **Cônjuge**, sendo este o domínio e imagem da relação material simétrica.

Ademais, para fins de exemplificação do mapeamento, acrescenta-se também a classe **Órgão Humano**, que exemplifica as classes do tipo *mixin*, particularmente o tipo *category*. Este é o tipo que agrupa instâncias de classes rígidas com princípios de identidade distintos, neste caso, as instâncias das classes **Coração** e **Cérebro** do tipo *kind*. A explicação sobre os tipos de classes e relações, vista em mais detalhes na seção 2.1.2, é retomada de forma resumida durante a definição dos mapeamentos nas seções seguintes.

4.1. MAPEAMENTOS PARA CENÁRIO ESTÁTICO

Esta seção apresenta duas propostas de mapeamento para representação de *cenários estáticos* da realidade. O primeiro, chamado de mapeamento para cenário estático simples, considera a representação direta das entidades do domínio conforme os estereótipos da linguagem OntoUML, enquanto o segundo, chamado de mapeamento para cenário estático reificado, considera a representação reificada das entidades do domínio conforme os mesmos estereótipos. Primeiramente são definidas as estruturas base para cada cenário, de forma complementar, e, em seguida, são definidas as diretrizes e o mapeamento propriamente dito.

Estrutura base para cenário estático simples

A Figura 45 apresenta um esquema estilo UML da estrutura OWL que serve de base para o mapeamento para cenário estático simples, dividida em 2 partes. Esta estrutura é uma adaptação da ontologia de tipos de indivíduos da UFO-A (ver seção 2.1.2), que reúne as categorias diretamente contempladas pelos estereótipos linguagem OntoUML. Na primeira parte da figura, representados em azul claro, estão os conceitos ou universais monádicos que são diretamente especializados pelos conceitos de domínio. A segunda parte apenas representa separadamente algumas relações que ocorrem entre instâncias da classe *Individual,* por uma questão de

legibilidade. Além disso, a especialização das propriedades nesses modelos é representada por um seta parecida com a de generalização, porém tracejada para distingui-la das outras.

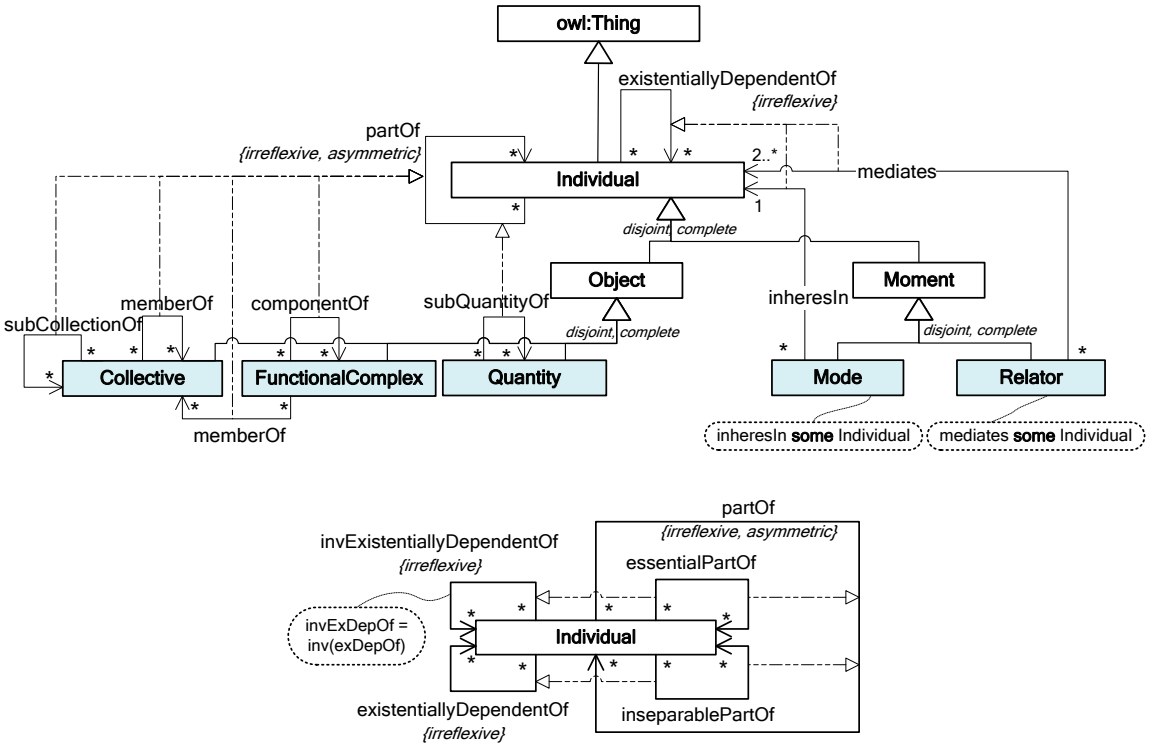


Figura 45. Esquema estilo UML da estrutura OWL para mapeamento para cenário estático simples.

Os indivíduos são divididos em dois tipos: Objeto (*Object*) ou *Moment*. O primeiro, por sua vez, se divide em três tipos: Complexo Funcional (*FunctionalComplex*), Coleção (*Collective*) e Quantidade (*Quantity*); enquanto o segundo se divide em dois tipos: Modo (*Mode*) e *Relator*. Assim, tem-se que todos os indivíduos do cenário estático devem ser instâncias de uma dessas classes.

Além das classes, o modelo representa também os tipos principais de relações que ocorrem entre as instâncias destas classes, a saber: (i) a relação de dependência existencial (*existentiallyDependentOf*) ocorre entre instâncias de *Individual* de forma irreflexiva, analogamente à (ii) relação de ser parte-de (*partOf*), sendo esta última também assimétrica. A relação parte-de é diferenciada entre as subclasses do tipo *Object* como (iii) sub-coleção-de (*subcollectionOf*), que ocorre entre instâncias de *Collective*, (iv) membro-de (*memeberOf*), que também ocorre entre instâncias de *Collective* ou ainda entre uma instância de *FunctionalComplex* e uma de *Collective*, (v) componente-de (*componentOf*), que ocorre entre instâncias de *FunctionalComplex*, e finalmente (vi) sub-quantidade-de (*subQuantityOf*), que ocorre entre instâncias de *Quantity*.

Além disso, a relação de inerência (*inheresIn*), nesta estrutura, ocorre entre instâncias de *Mode* ede *Individual*, é funcional (cardinalidade igual a 1) e define as instâncias de *Mode*, que instanciam de

forma necessária e suficiente esta relação (*inheresIn some Individual*). Por fim, a relação de mediação (*mediates*) ocorre entre uma instância de *Relator* e pelo menos duas de *Individual* (cardinalidade igual a 2 ou mais) e define as instâncias de *Relator,* que instanciam necessária e suficientemente esta relação (*mediates some Individual*).

Na segunda parte, as relações de dependência existencial e parte-de são detalhadas e relacionadas. São representadas a inversa da relação de dependência existencial (*invExistentiallyDepedentOf*), bem como duas sub-propriedades da relação parte-de, a saber, parte-essencial-de (*essentialPartOf*) e parte-inseparável-de (*inseparablePartOf*). A primeira é um tipo de dependência existencial inversa, enquanto a segunda é um tipo de dependência existencial.

Estrutura base para cenário estático reificado

A Figura 46 apresenta um esquema estilo UML da estrutura OWL que serve de base para o mapeamento para cenário estático reificado. Esta estrutura complementa aquela do mapeamento para cenário estático simples (elementos em cinza escuro), apresentado na Figura 45, como as entidades da UFO-A que não são diretamente contempladas nos estereótipos linguagem OntoUML.

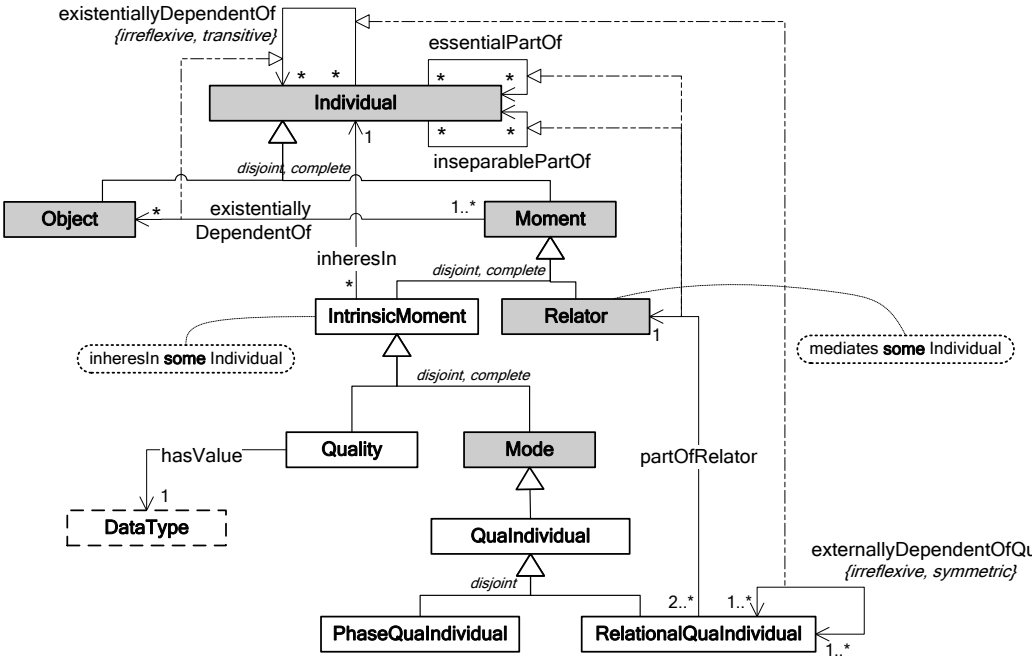


Figura 46. Esquema estilo UML estendendo a estrutura OWL da Figura 45 para o mapeamento para cenário estático reificado.

São incorporados os conceitos de *qualities* e *qua-individuals*, complementares aos *relators* e *modes*, que já estavam previstos na estrutura do cenário simples, para reificar as propriedades de um

indivíduo (ver seção 2.1.2). Também inclui-se o conceito de *moment* intrínseco (*intrinsic moment*) generaliza *qualities* e *modes*. A relação de inerência (*inheresIn*) ocorre agora entre instâncias de *IntrinsicMoment* e de *Individual* e define as instâncias do primeiro, que instanciam de forma necessária e suficiente esta relação (*inheresIn some Individual*). Um *quality* possui um único valor numa dimensão de qualidade, que é representada por uma propriedade funcional de tipo de dados (*hasValue*). Um tipo especial de *mode* é o *qua-individual*, que pode ainda ser classificado como fasal ou como relacional Este último é parte de (*partOf*) exatamente um *relator* e é externamente dependente (*externallyDependentOf*) dos outros *qua-individuals*.

A estrutura base da abordagem de Reificação Temporal (Figura 35) é similar à estrutura aqui proposta. Destacam-se as classes *TemporalExtent*, que não existe nesta pois trata-se da representação de cenário estático reificado, e *Mode*, que não existe naquela pois esta distinção ontológica estava fora do escopo da abordagem proposta. Contudo, algumas das diretrizes definidas para a abordagem de reificação temporal são seguidas para este mapeamento, excetuando-se aquelas específicas da representação temporal.

Diretrizes e Mapeamento

O Quadro 5 e o Quadro 6 a seguir tratam das diretrizes e a especificação do mapeamento das classes do tipo Objeto (*Object*) e *Moment* respectivamente, acompanhados de exemplos de aplicação. O texto explicativo contém a justificativa das diretrizes de mapeamento, identificadas por uma sigla numerada, por exemplo, **dsf1** significa "**D**iretriz **1** das classes tipo **S**ortal **F**inal". Tais diretrizes são atendidas pelo mapeamento, e, portanto, aparecem na especificação identificadas pela mesma sigla. Quando necessário, as diretrizes são diferenciadas para cada cenário, sendo colocadas lado a lado para facilitar a identificação das similaridades e diferenças.

CLASSES DO TIPO OBEJTO
CLASSES DO TIPO *SORTAL*   Toda instância de uma classe sortal carrega um princípio de identidade que permite individuá-la e contá-la.
Sortal Rígido (*Rigid Sortal*)   Toda instância de uma classe sortal rígida não pode deixar de instanciá-la sem deixar de existir.
Sortal Final (*Ultimate Sortal*)   Toda instância de uma classe sortal final tem seu princípio de identidade provido por ela, e, uma vez que não pode ter diferentes princípios de identidade, também não pode ser instância de outra classe deste tipo. Assim, todas as classes deste tipo são disjuntas **(dsf1)**.   Além disso, toda instância das classes do tipo sortal final, a saber, Espécie (*Kind*), Quantidade (*Quantity*) e Coleção (*Collective*), são classificadas respectivamente como Complexo Funcional

(*FunctionalComplex*), Quantidade (*Quantity*) ou Coleção (*Collective*) **(dsf2)**. Finalmente, todo indivíduo (*object*) do domínio deve ter um princípio de identidade provido por alguma classe ultimate sortal. Portanto, as classes do tipo sortal final formam uma partição completa dos tipos de indivíduos que especializam **(dsf3)**.	
« kind » **K**	Sejam K_1 .. K_n as classes do tipo *kind* : **(dsf1) DisjointClasses**(K_1 .. K_n) **(dsf2) SubClassOf**(K_i FunctionalComplex), para i de 1 até n **(dsf3) EquivalentClasses**(FunctionalComplex **ObjectUnionOf**(K_1 .. K_n))
« quantity » **Q**	Sejam Q_1 .. Q_n as classes do tipo *quantity* : **(dsf1) DisjointClasses**(Q_1 .. Q_n) **(dsf2) SubClassOf**(Q_i Quantity), para i de 1 até n **(dsf3) EquivalentClasses**(Quantity **ObjectUnionOf** (Q_1 .. Q_n))
« collective » **C**	Sejam C_1 .. C_n as classes do tipo *collective* : **(dsf1) DisjointClasses**(C_1 .. C_n) **(dsf2) SubClassOf**(C_i Collective), para i de 1 até n **(dsf3) EquivalentClasses**(Collective **ObjectUnionOf** (C_1 .. C_n))
***Sortal* Não-Final** São classes rígidas que, porém, não provêem o princípio de identidade. Assim, toda instância de uma classe sortal não-final tem o princípio de identidade provido por uma (e apenas uma) classe sortal final. Em outras palavras, uma classe sortal não-final deve especializar direta ou indiretamente exatamente uma classe sortal final. Por isso são chamadas de sub-tipo (*subkind*). Assim, as classes deste tipo sempre são partição (direta) de classes sortais rígidas S_i, podendo ser uma sortal final e/ou uma ou mais sortais não-final (afora os *mixins*, que são tratados separadamente) **(dsnf1)**. Tais partições podem ser compostas, completamente ou não, por outras classes sortais não-finais, disjuntas ou não **(dsnf2)**.	
« subkind » **SK**	Para cada partição P_i de classes do tipo *subkind*, sejam SK_1 .. SK_n as classes da partição e S a classe sortal particionada: **(dsnf1) SubClassOf**(SK_i S), para i de 1 até n **(dsnf2)** Se P_i é disjunta: **DisjointClasses**(SK_1 .. SK_n) Se P_i é completa: **EquivalentClasses**(S **ObjectUnionOf** (SK_1 .. SK_n))
[continua]	

Primeiramente são mapeadas as classes necessárias/rígidas ditas sortais, de forma idêntica para ambos os cenários. Em particular, este mapeamento obedece à diretriz **d1** da abordagem de reificação temporal. A Figura 47 mostra a aplicação parcial do mapeamento no exemplo da Figura 44. Os conceitos do tipo *kind*, a saber, **Pessoa**, **Coração** e **Cérebro**, são mapeados como subclasses disjuntas e completas da classe *FunctionalComplex*. Assim pode-se dizer que todos os complexos funcionais do domínio são instâncias de uma dessas classes. Também os conceitos do tipo *subkind*, a saber, **Homem** e **Mulher**, são mapeados como subclasses de suas respectivas super-classes.

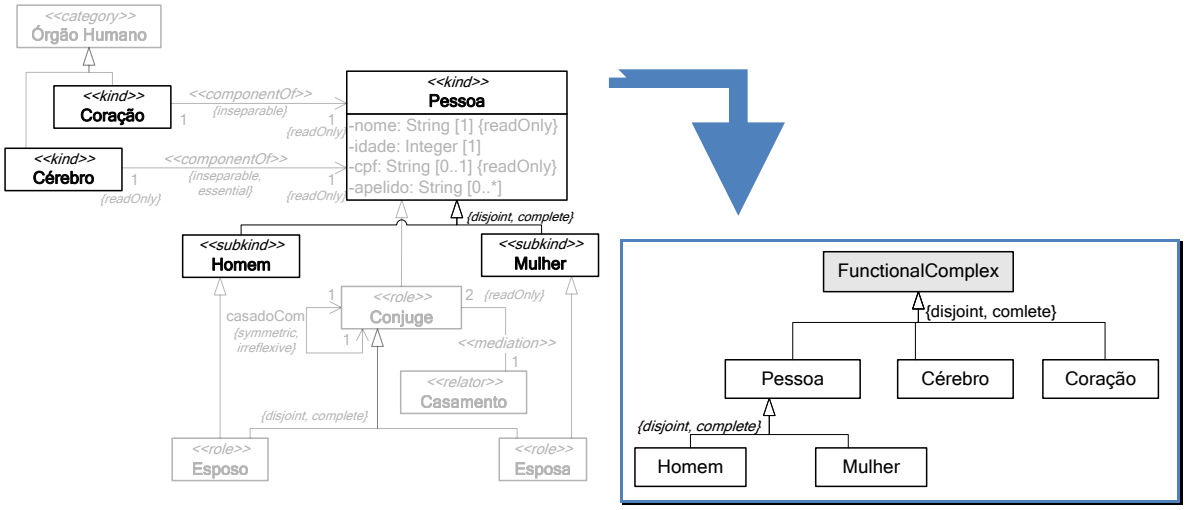


Figura 47. Aplicação parcial do mapeamento, para as classes do tipo sortal rígido do exemplo da Figura 44.

[continuando]

Sortal Anti-Rígido (*Anti-Rigid Sortal)*

Toda instância de uma classe do tipo sortal anti-rígido pode deixar de instanciá-la sem deixar de existir.

Tais classes não provêem o princípio de identidade. Assim, toda instância de uma classe sortal anti-rígida tem o princípio de identidade provido por uma (e apenas uma) classe sortal final. Em outras palavras, uma classe sortal anti-rígida especializa direta ou indiretamente exatamente uma classe sortal final.

As classes sortais anti-rígidas podem ser do tipo fase (*phase*) ou do tipo papel (*role*).

Cenário Estático Simples	Cenário Estático Reificado
No cenário estático, tal distinção não tem efeito algum, ou seja, as classes anti-rígidas são representadas analogamente àquelas sortais não-finais. Assim, as classes deste tipo sempre são partição (direta) de classes sortais S_i, podendo ser um sortal final e/ou uma ou mais sortais não-finais ou anti-rígidas (afora os *mixins*, que são tratados separadamente) **(dsar1)**. Tais partições podem ser compostas, completamente ou não, por outras classes sortais anti-rígidas, disjuntas ou não **(dsar2)**.	Tais classes caracterizam-se como propriedades contingentes de um indivíduo e são representadas no cenário estático reificado como *moments*, mais especificamente como classes de indivíduos-qua (*qua-individuals*) inerentes ao indivíduo. Classes anti-rígidas do tipo fase (*phase*) são reificadas como classes de indivíduos-qua fasais (*phased qua-individuals*), enquanto aquelas do tipo papel (*role*) são reificadas como classes de indivíduos-qua relacionais (*relational qua-individuals*) **(dsar0)**. Os indivíduos-qua das classes reificadas devem ser inerentes à exatamente um indivíduo, instância de alguma das classes sortais rígidas mais próximas na hierarquia **(dsar1)**. Uma classe deste tipo sempre é partição (direta) de classes sortais, podendo ser uma sortal final e/ou uma ou mais sortais não-final ou anti-rígida (afora os *mixins*, que são tratados separadamente). Tais partições podem ser compostas, completamente ou não, por outras classes

		sortais anti-rígidas, disjuntas ou não. Se a classe particionada S é rígida, tem-se que: (i) se a partição é disjunta, sendo um cenário estático, cada instância de S pode ter apenas um indivíduo-qua, instância de uma das classes reificadas que compõem a partição, inerente a ela; e (ii) se a partição é completa, cada instância de S deve ter ao menos um indivíduo-qua, instância de uma das classes reificadas que compõem a partição, inerente a ela. Caso contrário, se a classe particionada S é anti-rígida, ela é representada como QuaS e tem-se que: (i) cada instância das classes reificadas que compõem a partição deve ser existencialmente dependente de exatamente uma instância de QuaS; (ii) se a partição é disjunta, sendo um cenário estático, cada instância de QuaS pode ter apenas um indivíduo-qua, instância de uma das classes reificadas que compõem a partição, existencialmente dependente dela; e finalmente (iii) se a partição é completa, cada instância de QuaS deve ter ao menos um indivíduo-qua, instância de uma das classes reificadas que compõem a partição, existencialmente dependente dela **(dsar2)**.

« phase »
Ph

Fases sempre compõem partições disjuntas e completas de algum sortal. Significa que o indivíduo sortal deve estar em exatamente uma das fases num dado momento.

Cenário Estático Simples	Cenário Estático Reificado
Para cada partição P_i de classes do tipo *phase*, sejam Ph_1 .. Ph_n as classes da partição e S a classe sortal particionada: **(dsar1)** **SubClassOf**(Ph_i S), para i de 1 até n **(dsar2)** Se P_i é disjunta: **DisjointClasses**(Ph_1 .. Ph_n) Se P_i é completa: **EquivalentClasses**(S **ObjectUnionOf** (Ph_1 .. Ph_n))	Sejam Ph_1 .. Ph_n as classes do tipo *phase* : **(dsar0)** **SubClassOf**($QuaPh_i$ PhasedQuaIndividual), para i de 1 até n **(dsar1)** Sejam SR_{i1} .. SR_{im} as super-classes sortais rígidas mais próximas na hierarquia de Ph_i, para i de 1 até n **SubClassOf**($QuaPh_i$ **ObjectSomeValuesFrom**(inheresIn **ObjectUnionOf** (SR_{i1} .. SR_{im}))), para i de 1 até n {*inheresIn é funcional*} Para cada partição P de classes do tipo *phase*, sejam Ph_1 .. Ph_n as classes da partição e S a classe sortal particionada: **(dsar2)** Se S é uma classe rígida: **SubClassOf**(S **ObjectExactCardinality**(1 **InverseObjectProperty**(inheresIn) **ObjectUnionOf** ($QuaPh_1$ .. $QuaPh_n$))) Senão

	SubClassOf(QuaPh$_i$ **ObjectExactCardinality**(1 existentiallyDependentOf QuaS)), para i de 1 até n **SubClassOf**(QuaS **ObjectExactCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** (QuaPh$_1$ .. QuaPh$_n$)))

« role »
R

Cenário Estático Simples	Cenário Estático Reificado
Para cada partição P$_i$ de classes do tipo *role*, sejam R$_1$ .. R$_n$ as classes da partição e S a classe sortal particionada: **(dsar1)** **SubClassOf**(R$_i$ S), para i de 1 até n **(dsnf2)** Se P$_i$ é disjunta: **DisjointClasses**(R$_1$ .. R$_n$) Se P$_i$ é completa: **EquivalentClasses**(S **ObjectUnionOf** (R$_1$ .. R$_n$))	Sejam R$_1$ .. R$_n$ as classes do tipo *role*: **(dsar0)** **SubClassOf**(QuaR$_i$ RelationalQuaIndividual), para i de 1 até n **(dsar1)** Sejam SR$_{i1}$ .. SR$_{im}$ as super-classes sortais rígidas mais próximas na hierarquia de R$_i$, **SubClassOf**(QuaR$_i$ **ObjectSomeValuesFrom**(inheresIn **ObjectUnionOf** (SR$_{i1}$ .. SR$_{im}$))), para i de 1 até n {*inheresIn é funcional*} Para cada partição P de classes do tipo *role*, sejam R$_1$ .. R$_q$ as classes da partição e S a classe sortal particionada: **(dsar2)** Se S é uma classe rígida: Se P é disjunta: **SubClassOf**(S **ObjectMaxCardinality**(1 **InverseObjectProperty**(inheresIn) **ObjectUnionOf** (QuaR$_1$ .. QuaR$_q$))) Se P é completa: **SubClassOf**(S **ObjectMinCardinality**(1 **InverseObjectProperty**(inheresIn) **ObjectUnionOf** (QuaR$_1$ .. QuaR$_q$))) Senão (se S não é rígido, então é reificado) **SubClassOf**(QuaR$_k$ **ObjectExactCardinality**(1 existentiallyDependentOf QuaS)), para k de 1 até q Se P é disjunta: **SubClassOf**(QuaS **ObjectMaxCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** (QuaR$_1$ .. QuaR$_q$))) Se P é completa: **SubClassOf**(QuaS$_j$ **ObjectMinCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** (QuaR$_1$ .. QuaR$_q$)))

[continua]

Nesta etapa são mapeadas as classes contingentes/anti-rígidas. Em particular, este mapeamento obedece às diretrizes **d2**, e **d2.1 a d2.4** da abordagem de reificação. A Figura 48 e a Figura 49 mostram a aplicação parcial do mapeamento no exemplo da Figura 44 respectivamente para os

cenários estáticos simples e reificado. Os conceitos do tipo *role*, a saber, **Cônjuge**, **Esposo** e **Esposa**, especializam suas respectivas super-classes no cenário simples, enquanto são reificados como classes de indivíduos-qua no cenário reificado. Destaca-se que o conceito **Cônjuge**, por ser composto de forma disjunta e completa pelos conceitos **Esposo** e **Esposa**, quando reificado é necessariamente existencialmente dependente de no máximo uma e no mínimo uma instância reificada destas duas classes. Isso está representado no modelo estilo UML pela expressão owl **inv(exDep0f) exact 1 (QuaEsposo or QuaEsposa)**.

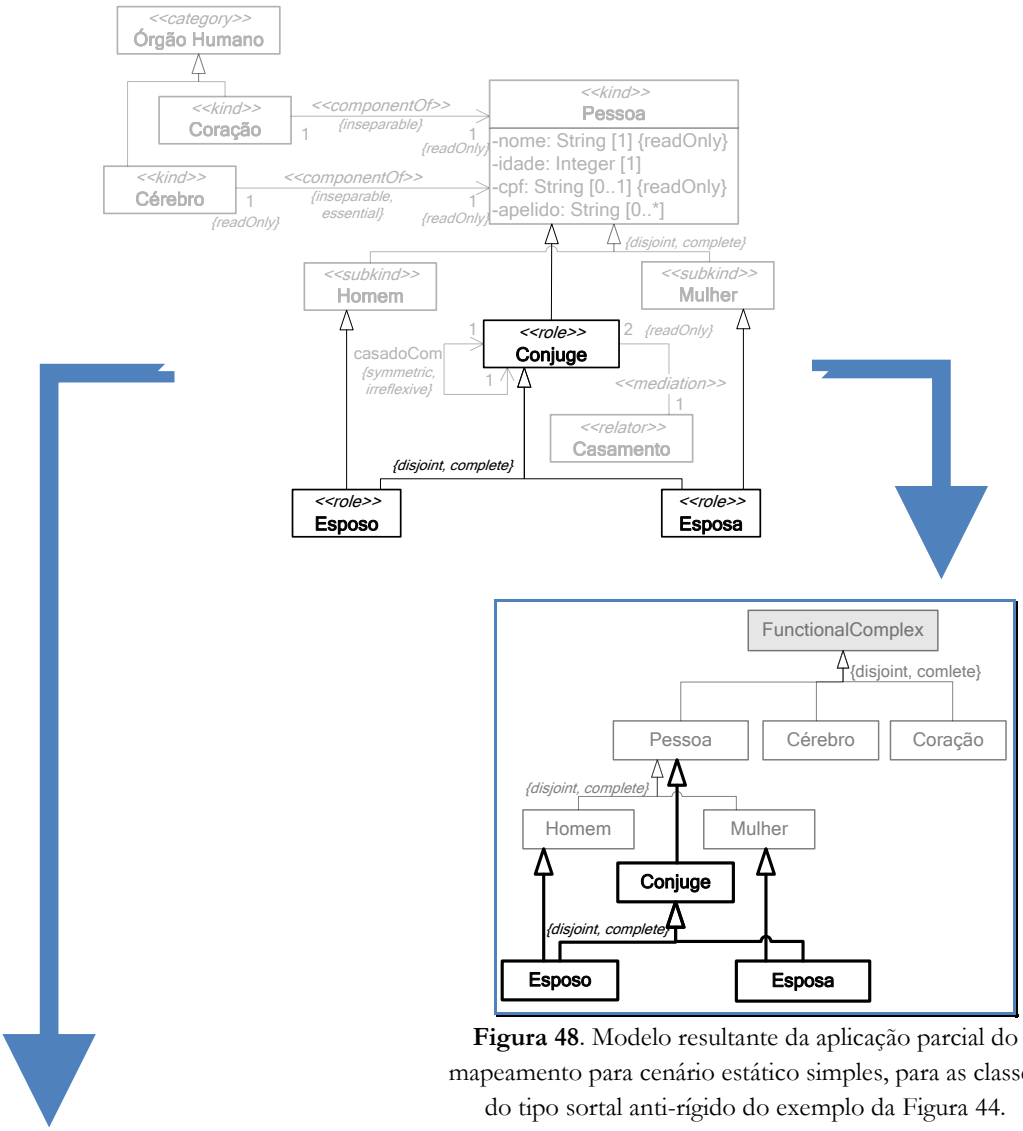


Figura 48. Modelo resultante da aplicação parcial do mapeamento para cenário estático simples, para as classes do tipo sortal anti-rígido do exemplo da Figura 44.

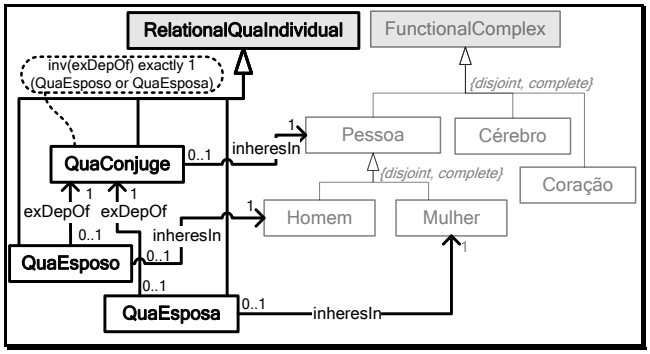


Figura 49. Modelo resultante da aplicação parcial do mapeamento para cenário estático reificado, para as classes do tipo sortal anti-rígido do exemplo da Figura 44.

[continuando]

CLASSES DO TIPO *MIXIN*

As classes do tipo *mixin* são sempre abstratas, ou seja, elas não provêem o princípio de identidade, mas agrupam indivíduos com princípios de identidade diferentes, providos por classes sortais finais. Tais classes são sempre completas em relação às classes que a especializam **(dmx1)**, e podem ser de três tipos: categorias (*caterory*), misturas (*mixin*) ou misturas de papéis (*role mixin*).

Mistura Rígida (*Rigid Mixin)*

Agrega instâncias de classes sortais rígidas, sendo, portanto, subclasse direta ou indireta da classe *Object*.

« category » **C**	Sejam C_1 .. C_n as classes do tipo *category*: **(dmx1)** Sejam S_{i1} .. S_{im} as classes sortais rígidas que C_i generaliza, **EquivalentClasses**(C_i **ObjectUnion**(S_{i1} .. S_{im})), para i de 1 até n

Mistura Não-Rígida (*Non-Rigid Mixin*)

Podem ser classes do tipo *mixin* ou *roleMixin*. As primeiras agregam classes sortais rígidas e anti-rígidas, enquanto as últimas agregam apenas classes sortais do tipo anti-rígido, particularmente do tipo *role*.

Cenário Estático Simples	Cenário Estático Reificado
Considerando-se que no cenário estático não se faz distinção entre classes rígidas e não-rígidas, tais classes são representadas de forma análoga àquelas do tipo *category*.	Podem ser classes do tipo *mixin* ou *roleMixin*. As primeiras agregam classes sortais rígidas e anti-rígidas, enquanto as últimas agregam apenas classes sortais do tipo anti-rígido, particularmente do tipo *role*. Assim, as classes do *mixin* deixam de fazer sentido no cenário reificado, pois representariam a agregação de indivíduos reificados e não reificados, sobre os quais não se aplicam as mesmas propriedades. Por outro lado, classes do tipo *roleMixin*, que são contingentes/anti-rígidas, são representadas como classes de indivíduos-qua (*qua-individuals*) relacionais *QuaRM*, e especializam a classe *RelationalQuaIndividual* **(dmx0)**. Essas classes são

	sempre particionadas de forma completa por outras classes anti-rígidas, que são representadas da mesma forma. Assim, tem-se que: (i) cada instância das classes reificadas que compõem a partição deve ser existencialmente dependente de exatamente uma instância de QuaRM; (ii) se a partição é disjunta, sendo um cenário estático, cada instância de QuaRM pode ter apenas um indivíduo-qua, instância de uma das classes reificadas que compõem a partição, existencialmente dependente dela; e finalmente (iii) cada instância de QuaRM deve ter ao menos um indivíduo-qua, instância de uma das classes reificadas que compõem a partição, existencialmente dependente dela **(dmx1)**.

« mixin »
M

Cenário Estático Simples	Cenário Estático Reificado
Sejam M_1 .. M_n as classes do tipo *mixin*: **(dmx1)** Sejam S_{i1} .. S_{im} as classes sortais que M_i generaliza, **EquivalentClasses**(M_i **ObjectUnionOf**(S_{i1} .. S_{im})), para i de 1 até n	Não é mapeada.

« roleMixin »
RM

Cenário Estático Simples	Cenário Estático Reificado
Sejam RM_1 .. RM_n as classes do tipo *roleMixin*: **(dmx1)** Sejam S_{i1} .. S_{im} as classes sortais anti-rigidas que RM_i generaliza, **EquivalentClasses**(RM_i **ObjectUnionOf**(S_{i1} .. S_{im})), para i de 1 até n	Para cada classe RM do tipo *roleMixin*: **(dmx0)** **SubClassOf**(QuaRM RelationalQuaIndividual) Para cada partição P de classes sortais anti-rigidas que RM generaliza, sejam S_1 .. S_m as classes da partição: **(dmx1)** **SubClassOf**($QuaS_i$ **ObjectExactCardinality**(1 existentiallyDependentOf QuaRM)), para i de 1 até m **SubClassOf**(QuaRM **ObjectMinCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** ($QuaS_i$ .. $QuaS_i$))) Se P é disjunta: **SubClassOf**(QuaRM **ObjectMaxCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** ($QuaS_1$ .. $QuaS_m$)))

Quadro 5. Diretrizes e mapeamento das classes OntoUML do tipo *Object* .

Nesta etapa são mapeadas as classes do tipo mixin, que podem ser tanto necessárias/rígidas quanto contingentes/anti-rígidas. Em particular, este mapeamento obedece às diretrizes **d1**, **d2**, e **d2.1 a d2.4** da abordagem de reificação. A Figura 50 e a Figura 51 mostram a aplicação parcial do mapeamento no exemplo da Figura 44 respectivamente para os cenários estáticos simples e reificado. O conceito **Órgão Humano** do tipo *category* é representado como generalização das classes **Coração** e **Cérebro** em ambos os cenários.

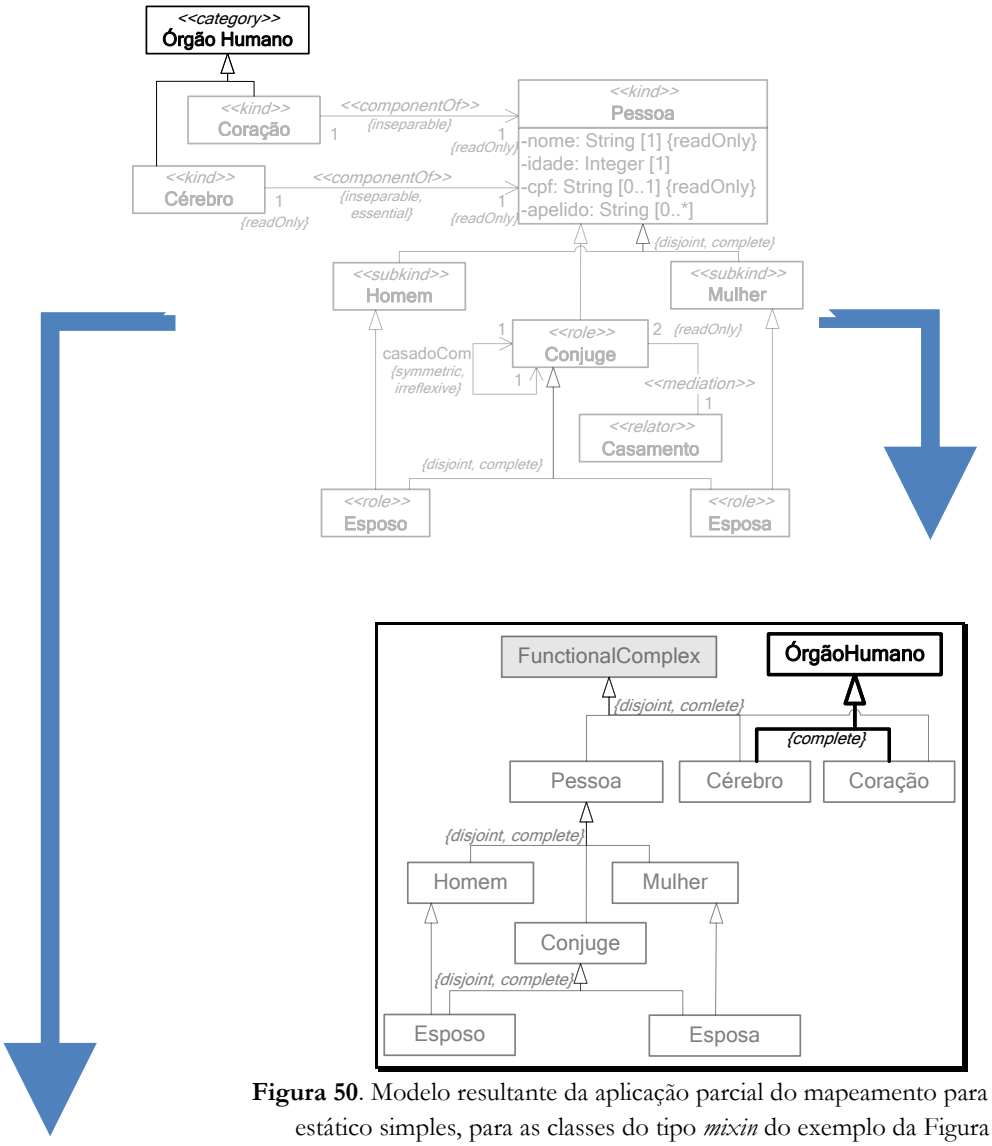


Figura 50. Modelo resultante da aplicação parcial do mapeamento para cenário estático simples, para as classes do tipo *mixin* do exemplo da Figura 44.

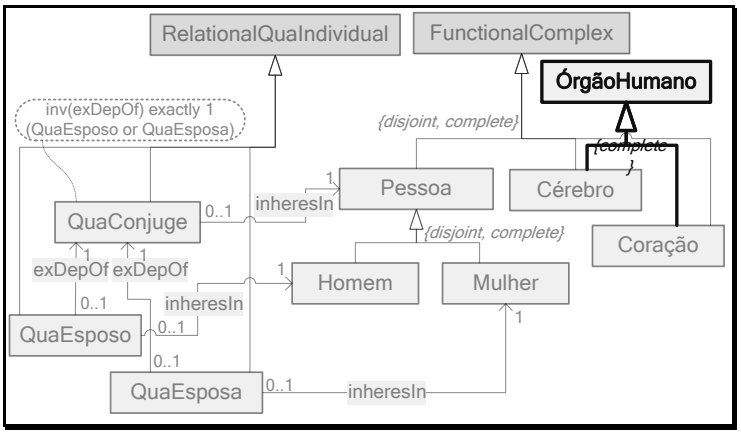


Figura 51. Modelo resultante da aplicação parcial do mapeamento para cenário estático reificado, para as classes do tipo *mixin* do exemplo da Figura 44.

CLASSES DO TIPO *MOMENT*	
Mode Agrega instâncias do tipo *mode* (modo), que representam uma característica intrínseca do indivíduo que não tem um valor num domínio concreto. Se tal classe não especializa outras, ela é subclasse direta da classe *Mode* **(dmA1)**. Caso contrário, tal classe é partição de uma ou mais classes do tipo *mode* **(dmB1)**. Tais partições P_i podem ser compostas, completamente ou não, por outras classes do tipo *mode*, disjuntas ou não **(dmB2)**.	
« mode » **M**	Para cada classe M do tipo *mode*: Se M não especializa outra classe **(dmA1) SubClassOf**(M Mode) Senão, para cada partição P de que M participa, sejam M_1 .. M_n as classes da partição e PM a classe particionada: **(dmB1) SubClassOf**(M PM) **(dmB2)** Se P_i é disjunta: **DisjointClasses**(M_1 .. M_n) Se P_i é completa: **EquivalentClasses**(PM **ObjectUnionOf**(M_1 .. M_n))
Relator Agrega instâncias do tipo *relator* (relacionador), sendo, portanto, subclasse direta ou indireta da classe *Relator*. Se tal classe não especializa outras, ela é subclasse direta da classe *Relator* **(drA1)**. Caso contrário, tal classe é partição de uma ou mais classes do tipo *relator* **(drB1)**. Tais partições P_i podem ser compostas, completamente ou não, por outras classes do tipo *relator*, disjuntas ou não **(drB2)**.	
« relator » **R**	Para cada classe R do tipo *relator*: Se R não especializa outra classe **(drA1) SubClassOf**(R Relator) Senão, para cada partição P de que R participa, sejam R_1 .. R_n as classes da partição e PR a classe particionada: **(drB1) SubClassOf**(R PR) **(drB2)** Se P é disjunta: **DisjointClasses**(R_1 .. R_n) Se P é completa: **EquivalentClasses**(PR **ObjectUnionOf**(R_1 .. R_n))

Quadro 6. Diretrizes e mapeamento para classes OntoUML do tipo *Moment*.

Nesta etapa são mapeadas as classes do tipo *moment*. Em particular, este mapeamento obedece à diretriz **d4** da abordagem de reificação. A Figura 52 e a Figura 53 mostram a aplicação parcial do mapeamento no exemplo da Figura 44 respectivamente para os cenários estáticos simples e reificado. O conceito **Casamento** do tipo relator é representado como especialização da classe *Relator* em ambos os cenários.

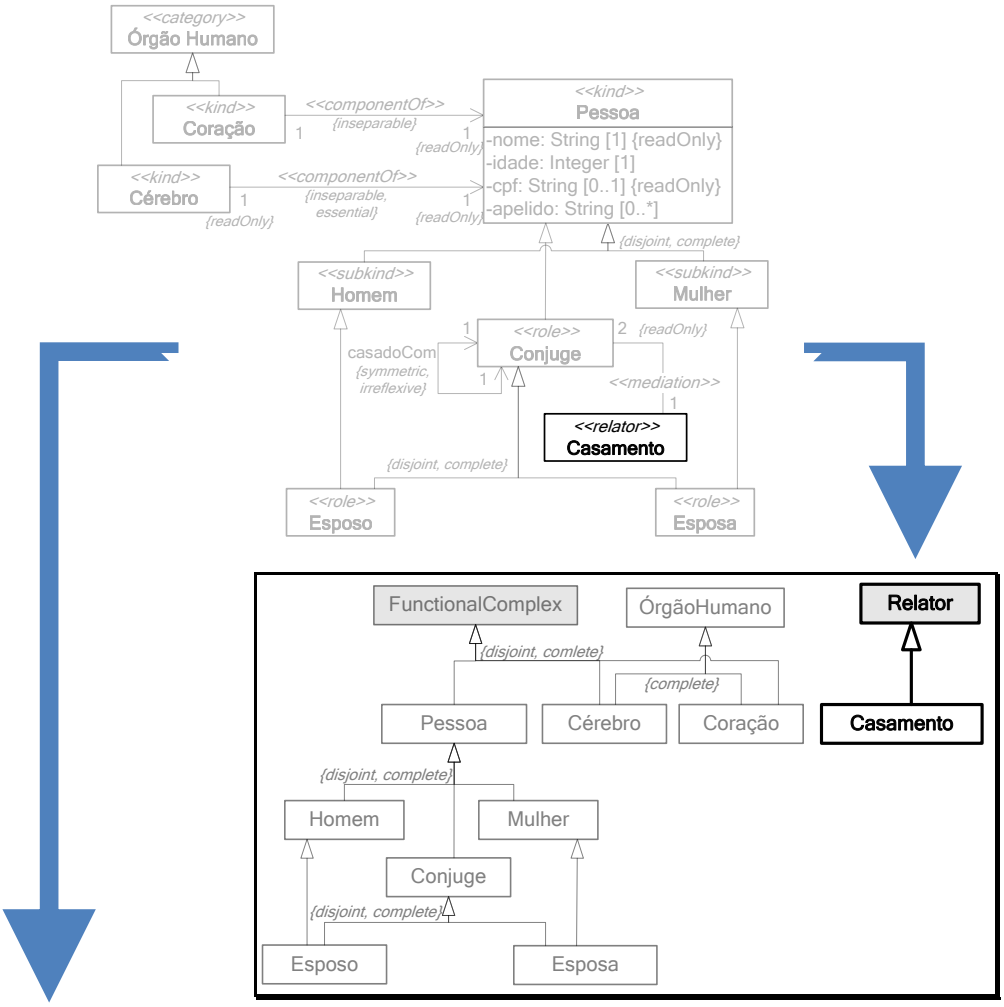


Figura 52. Modelo resultante da aplicação parcial do mapeamento para cenário estático simples, para as classes do tipo *moment* do exemplo da Figura 44.

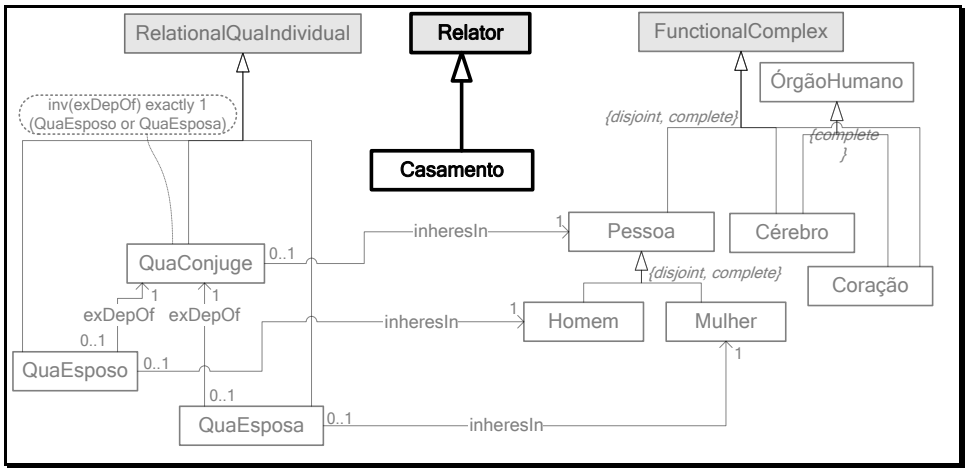


Figura 53. Modelo resultante da aplicação parcial do mapeamento para cenário estático reificado, para as classes do tipo *moment* do exemplo da Figura 44.

O Quadro 7 trata das diretrizes e mapeamento das relações entre os conceitos, representados como associações (*associations*) na OntoUML.

ASSOCIAÇÕES (*ASSOCIATIONS*)	
Considera-se apenas associações binárias direcionadas, sendo o domínio e a imagem referidos respectivamente pelas letras D e I, bem como os limites inferior e superior da cardinalidade do final de associação ligado ao domínio referidos respectivamente como minD e maxD, e, analogamente, para o final de associação ligado à imagem usa-se minI e maxI (max vale -1 se for infinito). As classes domínio D e imagem I devem ter sua cardinalidade restringida para cada associação **(da1)**. Se uma das classes de domínio ou imagem for rígida, e a relação for necessária e imutável para suas instâncias, então a relação implica em dependência existencial e deve instanciar direta ou inversamente a relação *existentiallyDependentOf* (existencialmente dependente de). **(da2)**	
ASSOCIAÇÃO FORMAL (Formal Association) Neste trabalho, são aquelas que valem pela simples existência dos indivíduos que ela relaciona, ou seja, não dependem de um indivíduo externo interventor para valer. A relação é representada com o nome que lhe é dado e seu domínio e imagem devem ser restringidos **(daf1)**	
Cenário Estático Simples	**Cenário Estático Reificado**
	Neste cenário considera-se que este tipo de relação sempre implica em dependência existencial entre os indivíduos que ela relaciona. Ou seja, deve sempre instanciar direta e/ou inversamente a relação *existentiallyDependentOf*. Assim, apenas um dentre o domínio e a imagem podem não ser rígidos. Caso um deles não seja rígido, a relação deve valer para as super-classes rígidas mais próximas na hierarquia **(daf0)**.
« formal » **R**	
Cenário Estático Simples	**Cenário Estático Reificado**

(daf1) **ObjectPropertyDomain** (R D) **ObjectPropertyRange** (R I) **(da1)** Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (R) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (R) D)) Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minD R I)) Se maxD > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxD R I)) **(da2)** Se *IsReadOnly* = *true* para o final de associação ligado a D e minD > 0 e I é do tipo rígido, então **SubObjectPropertyOf** (R invExistentiallyDependentOf) Se *IsReadOnly* = *true* para o final de associação ligado a I e minI > 0 e D é do tipo rígido, então **SubObjectPropertyOf** (R existentiallyDependentOf)	**(daf0)** Se D é rígido, D' ← D senão, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) Se I é rígido, I' ← I senão, sejam IR_1 .. IR_n as super-classes rígidas mais próximas na hierarquia de I. I' ← **ObjectUnionOf**(IR_1 .. IR_n) **(daf1)** **ObjectPropertyDomain** (R D') **ObjectPropertyRange** (R I') **(da1)** Se I é rígido, Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (R) D')) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (R) D')) Se D é rígido, Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minD R I')) Se maxD > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxD R I')) **(da2)** Se *IsReadOnly* = *true* para o final de associação ligado a D e minD > 0 e I é do tipo rígido, então **SubObjectPropertyOf** (R invExistentiallyDependentOf) Se *IsReadOnly* = *true* para o final de associação ligado a I e minI > 0 e D é do tipo rígido, então **SubObjectPropertyOf** (R existentiallyDependentOf)

ASSOCIAÇÃO MATERIAL (*Material Association*)

São aquelas que dependem de um indivíduo externo interventor para valer chamado *relator*.

Cenário Estático Simples	Cenário Estático Reificado
A relação é representada com o nome que lhe é dado e seu domínio e imagem devem ser restringidos **(dam0)** Ademais, esta relação é sempre irreflexiva,	Neste cenário considera-se que este tipo de relação sempre ocorre entre classes do tipo papel. Entretanto, esta relação não é representada diretamente, mas reificada. Neste caso, apenas restringe-se a relação

e é simétrica se domínio e imagem desempenham o mesmo papel, ou seja, se o domínio é igual à imagem. **(dam1)**	*externallyDependentOfQua* entre as classes do tipo *RelationalQuaIndividual* correspondentes aos papéis também reificados. **(dam0)**

« material »
M

Cenário Estático Simples	Cenário Estático Reificado
(dam0) **ObjectPropertyDomain** (M D) **ObjectPropertyRange** (M I) **(dam1)** **IrreflexiveObjectProperty**(M) Se D = I **SymmetricObjectProperty**(M) **(da1)** Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (M) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (M) D)) Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minD M I)) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI M I)) **(da2)** Se *IsReadOnly* = *true* para o final de associação ligado a D e minD > 0 e I é do tipo rígido, então **SubObjectPropertyOf** (M invExistentiallyDependentOf) Se *IsReadOnly* = *true* para o final de associação ligado a I e minI > 0 e D é do tipo rígido, então **SubObjectPropertyOf** (M existentiallyDependentOf)	Se D não é rígido e I não é rígido **(dam0)** Se minD > 0 **SubClassOf**(Qual **ObjectMinCardinality**(minD externallyDependentOfQua QuaD)) Se maxD > 0 **SubClassOf**(Qual **ObjectMaxCardinality**(maxD externallyDependentOfQua QuaD)) Se minI > 0 **SubClassOf**(QuaD **ObjectMinCardinality**(minI externallyDependentOfQua Qual)) Se maxI > 0 **SubClassOf**(QuaD **ObjectMaxCardinality**(maxI externallyDependentOfQua Qual))

[continua]

Nesta etapa são mapeadas as associações do tipo formal e material. Em particular, este mapeamento obedece às diretrizes **d3**, **d3.1**, **d3.2** e **d4.3** da abordagem de reificação. A Figura 54 e a Figura 55 mostram a aplicação parcial do mapeamento no exemplo da Figura 44

respectivamente para os cenários estáticos simples e reificado. A relação material **casado-com** é diretamente mapeada entre indivíduos da classe **Cônjuge** no primeiro caso, enquanto no segundo divido à sua reificação, é necessário relacionar os qua-indivíduos envolvidos na relação através da relação de dependência externa entre *qua-individuals*, a saber, *externallyDepedentOfQua*.

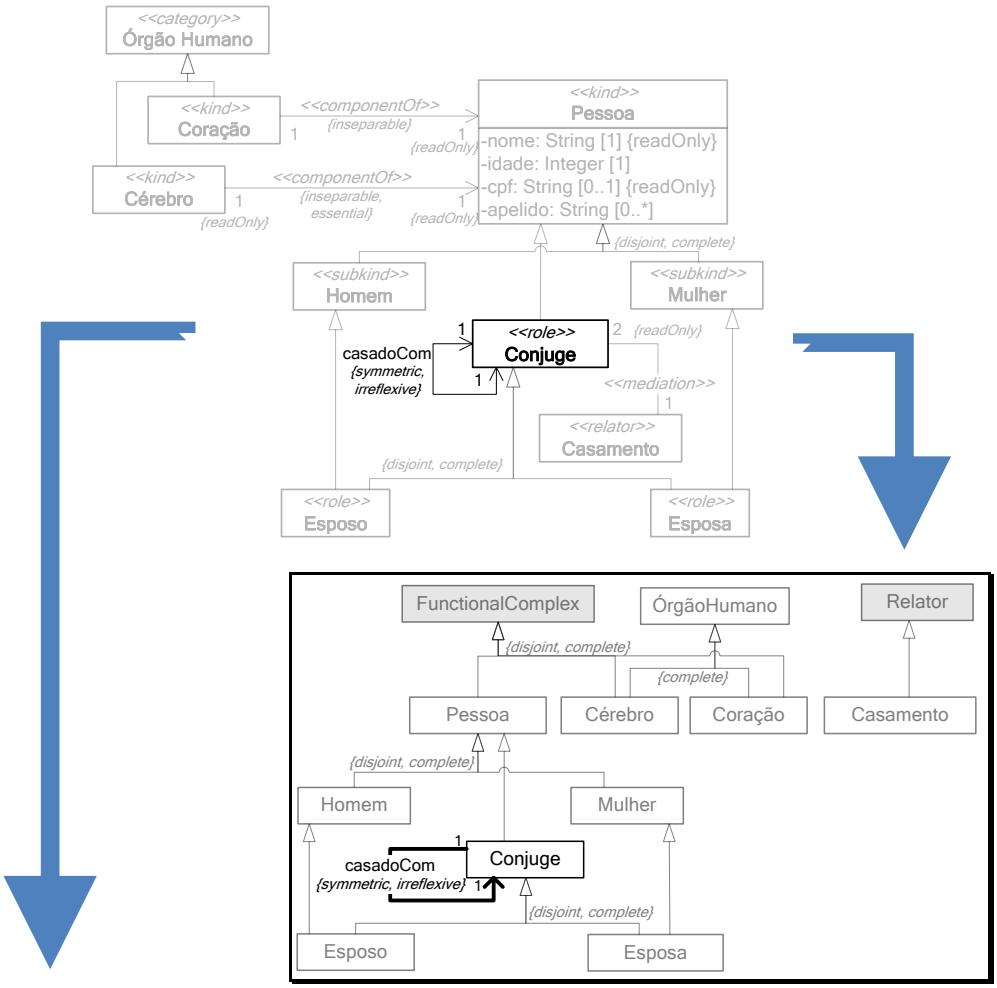


Figura 54. Modelo resultante da aplicação parcial do mapeamento para cenário estático simples, para as associações materiais do exemplo da Figura 44.

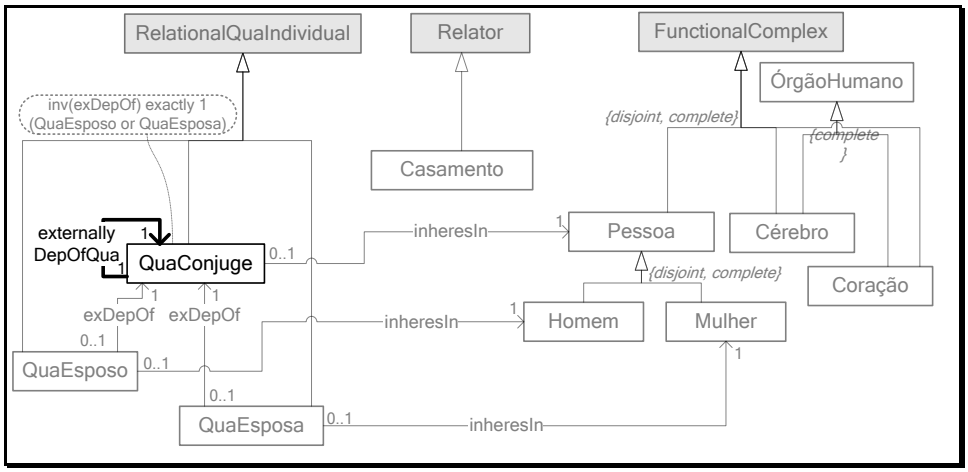


Figura 55. Modelo resultante da aplicação parcial do mapeamento para cenário estático reificado, para as associações materiais do exemplo da Figura 44.

[continuando]

ASSOCIAÇÃO BINÁRIA DIRECIONADA (*Directed Binary Relatioship)*

Relações de dependência (*Dependency Relationship*)

Tipos especiais de relações de dependência existencial chamados caracterização (*characterization*) e mediação (*mediation*).

Cenário Estático Simples	Cenário Estático Reificado
	Uma vez que o domínio é sempre uma classe do tipo *moment*, apenas a classe de imagem pode não ser rígida. Neste caso, considera-se então as super-classes rígidas mais próximas na hierarquia da imagem **(drd0)**.

« characterization » Esta relação ocorre entre uma classe do tipo *Mode* (D) que caracteriza a classe de imagem (I). É representada restringindo-se a propriedade funcional *inheresIn* (inerente a) entre domínio e imagem.

Cada instância de uma classe D do tipo *mode* deve ser inerente a exatamente um indivíduo, instância da classe I. **(dch1)**

Cenário Estático Simples	Cenário Estático Reificado
	(drd0) Se I é rígido, I' ← I senão, sejam IR_1 .. IR_n as super-classes rígidas mais próximas na hierarquia de I. I' ← **ObjectUnionOf**(IR_1 .. IR_n)
(dch1) SubClassOf(D **ObjectSomeValuesFrom** (inheresIn I)) {*inheresIn é funcional*}	**(dch1) SubClassOf**(D **ObjectSomeValuesFrom** (inheresIn I')) {*inheresIn é funcional*}
Se *IsReadOnly* = *true* para o final de associação ligado a D e minD > 0 e I é do tipo rígido, então	**SubClassOf**(D **ObjectAllValuesFrom** (inheresIn S)) Se *IsReadOnly* = *true* para o final de associação ligado a D e minD > 0 e I é do tipo rígido, então

SubObjectPropertyOf (inheresIn invExistentiallyDependentOf) **(da1)** Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (inheresIn) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (inheresIn) D))	**SubObjectPropertyOf** (inheresIn invExistentiallyDependentOf) **(da1)** Se I é rígido, Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (inheresIn) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (inheresIn) D)) senão (I não é rígido) Se maxD > 0, para cada IR dentre as super-classes rígidas mais próximas na hierarquia de I **SubClassOf**(IR **ObjectMaxCardinality**(maxD **ObjectInverseOf** (inheresIn) D))

« mediation » Esta relação ocorre entre uma classe do tipo *Relator* (D) que media uma outra classe (I), e é representada restringindo-se a propriedade funcional *mediates* (media) entre domínio e imagem.

Cenário Estático Simples	Cenário Estático Reificado
Se *IsReadOnly* = *true* para o final de associação ligado a D e minD > 0 e I é do tipo rígido, então **SubObjectPropertyOf** (mediates invExistentiallyDependentOf) **(da1)** Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (mediates) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (mediates) D)) **SubClassOf**(D **ObjectMinCardinality**(minI mediates I)) {*minI é sempre > 0*} Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI mediates I))	Se *IsReadOnly* = *true* para o final de associação ligado a D e minD > 0 e I é do tipo rígido, então **SubObjectPropertyOf** (mediates invExistentiallyDependentOf) **(da1)** Se I é rígido, Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (mediates) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (mediates) D)) **SubClassOf**(D **ObjectMinCardinality**(minI mediates I)) {*minI é sempre > 0*} Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI mediates I)) senão (I não é rígido) **SubClassOf**(Qual **ObjectSomeValuesFrom** (partOfRelator D)) {*partOfRelator é funcional*} **SubClassOf**(D **ObjectMinCardinality**(minI

	ObjectInverseOf (partOfRelator) Qual)) {*minI é sempre > 0*} Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI **ObjectInverseOf** (partOfRelator) Qual)) Para cada IR dentre as super-classes rígidas mais próximas na hierarquia de I Se maxD > 0 **SubClassOf**(IR **ObjectMaxCardinality**(maxD **ObjectInverseOf** (mediates) D)) Se minI > 0 **SubClassOf**(D **ObjectMinCardinality** (minI mediates **ObjectUnionOf**(IR_1 .. IR_n))) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality** (maxI mediates **ObjectUnionOf**(IR_1 .. IR_n)))

[continua]

Nesta etapa são mapeadas as associações do tipo *characteriztion* e *mediation*. Em particular, este mapeamento obedece às diretrizes **d4.1** e **d4.2** da abordagem de reificação. A Figura 56 e a Figura 57 mostram a aplicação parcial do mapeamento no exemplo da Figura 44 respectivamente para os cenários estáticos simples e reificado. A relação de mediação entre a classe **Casamento** do tipo *relator* e a classe **Cônjuge** do tipo *role* é representada diretamente no primeiro caso, enquanto no segundo a reificação da classe **Cônjuge** faz com que tal relação seja representada como uma relação opcional para a sua super-classe rígida, a saber, **Pessoa**, bem como implica que a relação **partOfRelator** seja mutuamente obrigatória entre sua classe reificada **QuaConjuge** e a classe **Casamento**, ou seja, toda instância qua-conjuge deve ser parte de um relator casamento, bem como todo relator deste tipo deve ser composto de indivíduos qua-conjuges.

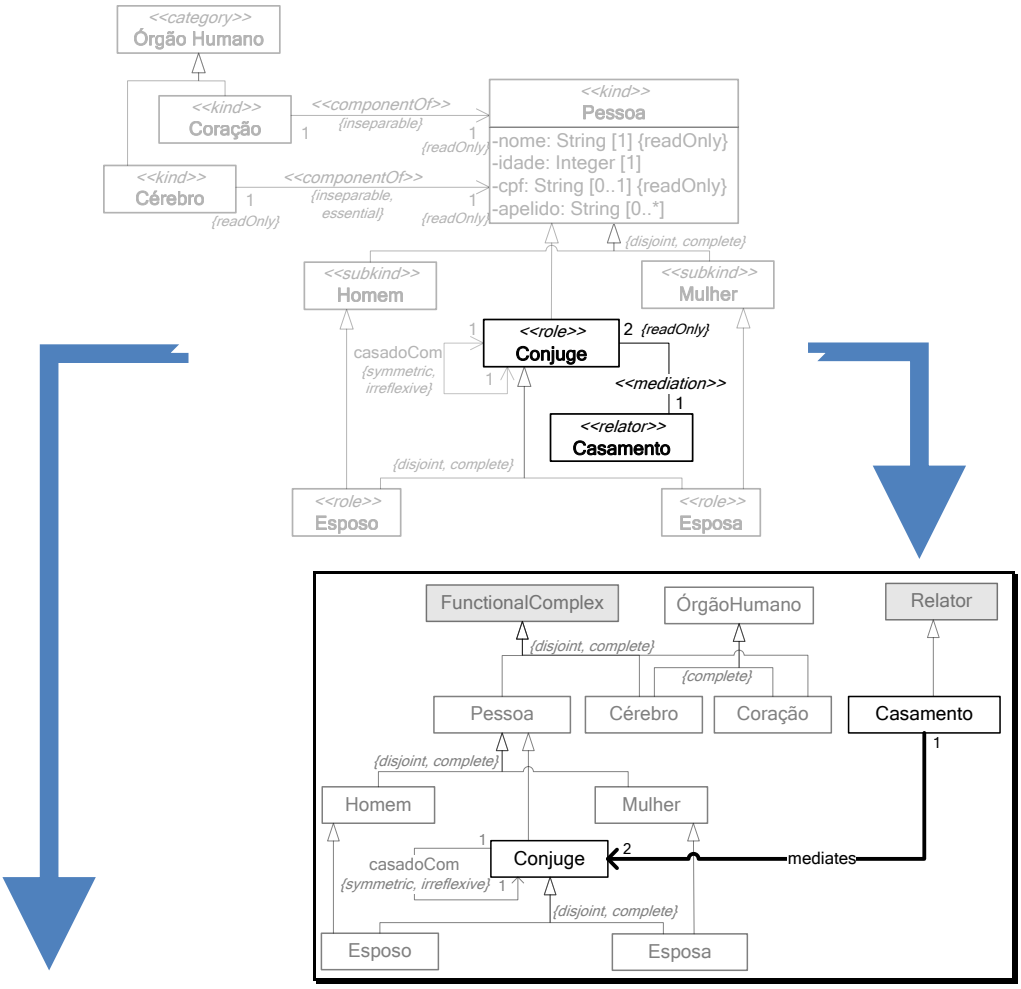


Figura 56. Modelo resultante da aplicação parcial do mapeamento para cenário estático simples, para as relações de dependência do exemplo da Figura 44.

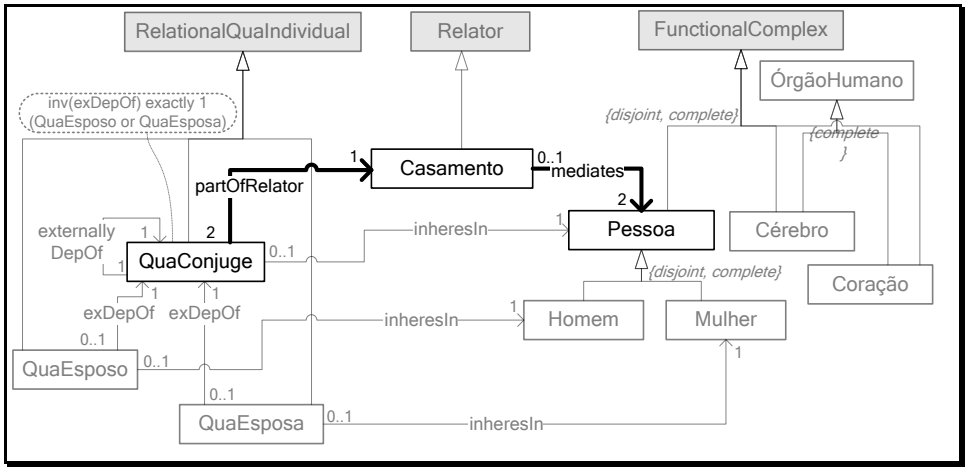


Figura 57. Modelo resultante da aplicação parcial do mapeamento para cenário estático reificado, para as relações de dependência do exemplo da Figura 44.

[continuando]

Relações Meronímicas (*Meronymic Relationship*)

São os possíveis tipos de relação todo-parte.

As relações são então representadas como especializações do respectivo subtipo de relação parte-de (*partOf*), a saber, componente-de (*componentOf*), membro-de (*member-of*), subcoleção-de (*subCollectiveOf*) e

subquantidade-de (*subQuantityOf*), especificando-se as classes de domínio e imagem. Além disso, o nome da relação é dado pelo tipo de relação todo parte, composto com os nomes das classes de domínio e imagem. **(darm1)**

As meta-propriedades consideradas são é-essencial (*isEssential*) e é-inseparável (*isInseparable*). No primeiro caso, a relação é necessária e imutável para o todo, no segundo, para a parte. Em cada caso, se o todo (ou a parte) é rígido, a relação implica em dependência existencial. **(darm2)**

Cenário Estático Simples	Cenário Estático Reificado
	No cenário reificado, considera-se apenas as relações meronímicas formais, ou seja, que implicam em dependência existencial. Assim, ou a relação é de parte inseparável e/ou de parte essencial. Se as classes de domínio ou imagem não forem rígidas, considera-se então como domínio ou imagem as super-classes rígidas mais próximas na hierarquia. **(darm0)**

Associação meronímica D →I

Se a associação é do tipo componente-de:

merRel ← componentOf

senão, se é do tipo subquantidade-de:

merRel ← subQuantityOf

senão, se é do tipo subcoleção-de:

merRel ← subCollectiveOf

senão, se é do tipo membro-de:

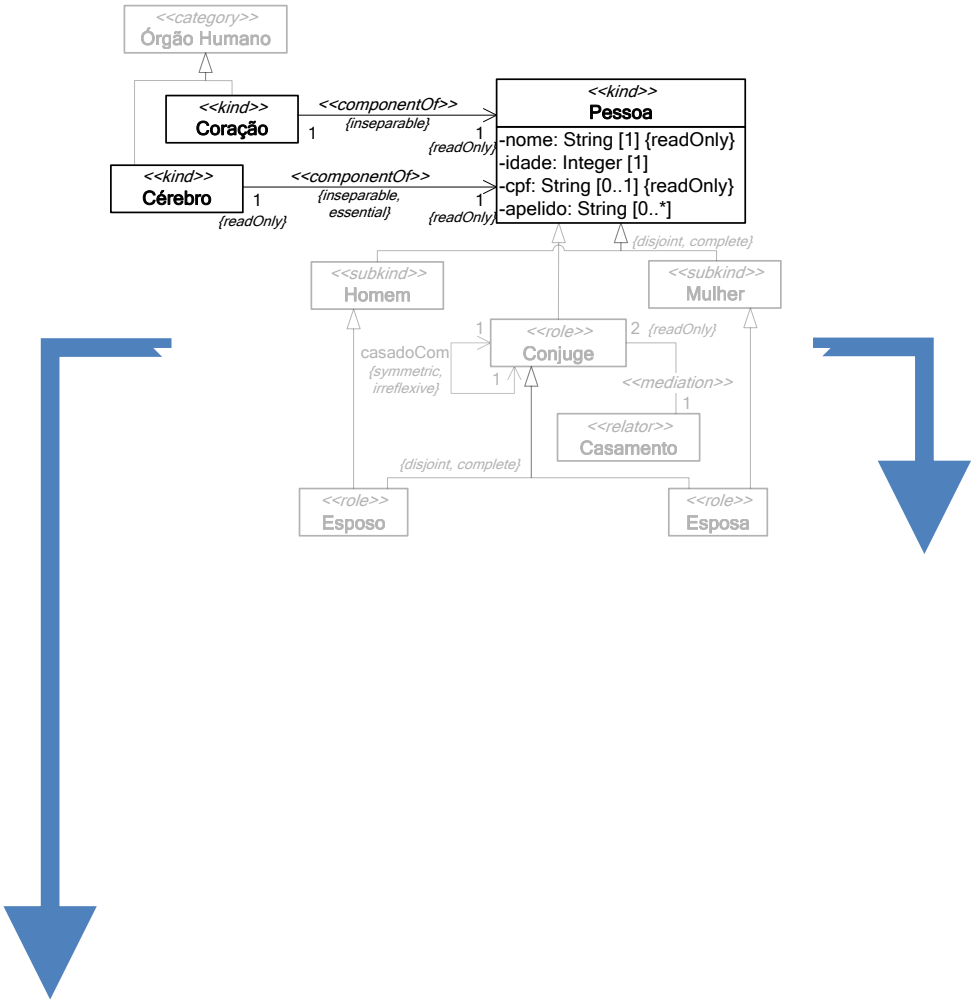
merRel ← memberOf

Cenário Estático Simples	Cenário Estático Reificado
(darm1) **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D) **ObjectPropertyRange** (merRelDI I)	**(darm0)** Se D é rígido, D' ← D senão, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) Se I é rígido, I' ← I senão, sejam IR_1 .. IR_n as super-classes rígidas mais próximas na hierarquia de I. I' ← **ObjectUnionOf**(IR_1 .. IR_n) **(darm1)** **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D') **ObjectPropertyRange** (merRelDI I')
(da2) (darm2) Se isEssential = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se isInseparable = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)	**(da2) (darm2)** Se isEssential = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se isInseparable = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)
(da1)	**(da1)**

Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D)) Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minI merRelDI I)) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI merRelDI I))	Se I é rígido, Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D')) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D')) Se D é rígido, Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minI merRelDI I')) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI merRelDI I'))

Quadro 7. Diretrizes e mapeamento para associações OntoUML

Nesta etapa são mapeadas as associações do tipo *meronymic*. Em particular, este mapeamento obedece às diretrizes **d3**, **d3.1** e **d3.2** da abordagem de reificação. A Figura 58 e a Figura 59 mostram a aplicação parcial do mapeamento no exemplo da Figura 44 respectivamente para os cenários estáticos simples e reificado. A relação meronímica **componente-de** entre as classes **Coração-Pessoa** e **Cérebro-Pessoa** são representadas diretamente em ambos os cenários, uma vez que se trata de classes rígidas.



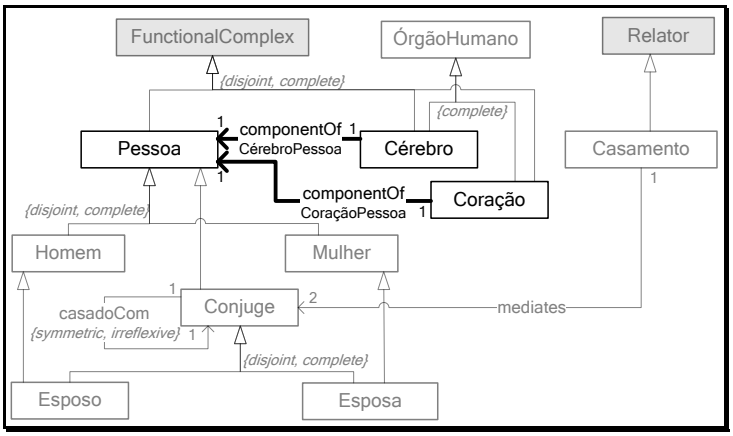


Figura 58. Modelo resultante da aplicação parcial do mapeamento para cenário estático simples, para as relações meronímicas do exemplo da Figura 44.

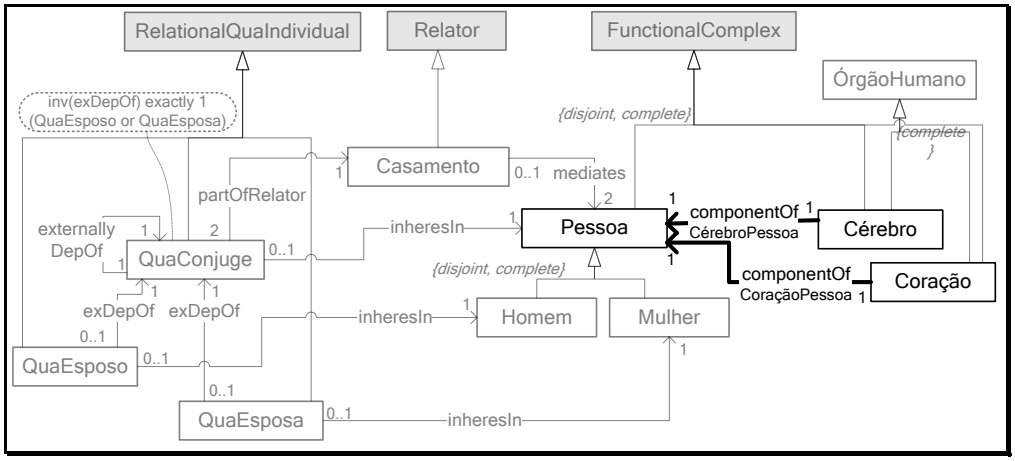


Figura 59. Modelo resultante da aplicação parcial do mapeamento para cenário estático reificado, para as relações meronímicas do exemplo da Figura 44.

O Quadro 8 trata das diretrizes e mapeamento das atributos dos conceitos, representados como propriedades (*properties*) na OntoUML.

PROPRIEDADES	
Uma propriedade é possuída por uma e apenas uma classe D. Os limites inferior e superior da cardinalidade são referidos respectivamente min e max (max vale -1 se for infinito). Possuem uma propriedade chamada *isReadOnly* que é verdadeira caso a propriedade seja imutável, e outra chamada *getType* corresponde ao tipo de dado.	
Cenário Estático Simples	**Cenário Estático Reificado**
A propriedade é representada por um tipo de relação chamado DatatypeProperty, e o padrão de nome sugerido neste mapeamento é o verbo "tem" seguido do nome da propriedade. O domínio é dado pela classe D, e a imagem é dada pelo tipo de dado da propriedade **(dp1)**. A classe D deve ter sua	Não é representada diretamente mas é reificada através de uma classe de *qualities* (indivíduos que representam qualidades) nomeada pelo nome da propriedade (P) seguido pelo nome da classe (D), e que especializa a classe Quality **(dp1)**. As instâncias desta classe são inerentes a um e apenas um indivíduo da classe D **(dp2)**, e possuem um e apenas um valor, representado pela relação

cardinalidade restringida para tal relação **(dp2)**.	*hasValue*, cuja a imagem é dada pelo tipo de dado da propriedade. **(dp3)** A quantidade de instância da classe CP que pode ser inerente a um indivíduo da classe D deve ser restringida conforme a cardinalidade da propriedade. **(dp4)**

property
Prop

Cenário Estático Simples	Cenário Estático Reificado
(dp1) **DataPropertyDomain** (temProp D) **DataPropertyRange** (temProp *getType*) **(dp2)** Se min > 0 **SubClassOf**(D **DataMinCardinality**(min temProp)) Se max > 0 **SubClassOf**(D **DataMaxCardinality**(max temProp))	**(dp1)** **SubDataPropertyOf** (PropD Quality) **(dp2)** Se D é rígido, D' ← D senão, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) **SubClassOf**(PropD **ObjectSomeValuesFrom** (inheresIn D')) {*inheresIn é funcional*} **(dp3)** **SubClassOf**(PropD **DataSomeValuesFrom** (hasValue *getType*)) {*hasValue é funcional*} **(dp4)** Se D é rígido, Se min > 0 **SubClassOf**(D **ObjectMinCardinality**(min **ObjectInverseOf** (inheresIn) PropD)) Se max > 0 **SubClassOf**(D **ObjectMaxCardinality**(max **ObjectInverseOf** (inheresIn) PropD))

Quadro 8. Diretrizes e mapeamento para propriedades/atributos OntoUML

Nesta etapa são mapeadas as propriedades/atributos dos conceitos OntoUML. Em particular, este mapeamento obedece à diretriz **d5** da abordagem de reificação. A Figura 60 e a Figura 61 mostram a aplicação parcial do mapeamento no exemplo da Figura 44 respectivamente para os cenários estáticos simples e reificado. As propriedades **nome**, **idade**, **CPF** e **apelido** da classe **Pessoa** são representadas diretamente como propriedades de tipos de dados (*datatype properties*) de **Pessoa** no primeiro caso, enquanto no segundo são reificadas como classes de indivíduos do tipo qualidade (*quality*). Tais indivíduos têm exatamente um valor correspondente ao seu tipo de dados, e são inerentes a exatamente uma instância de **Pessoa**.

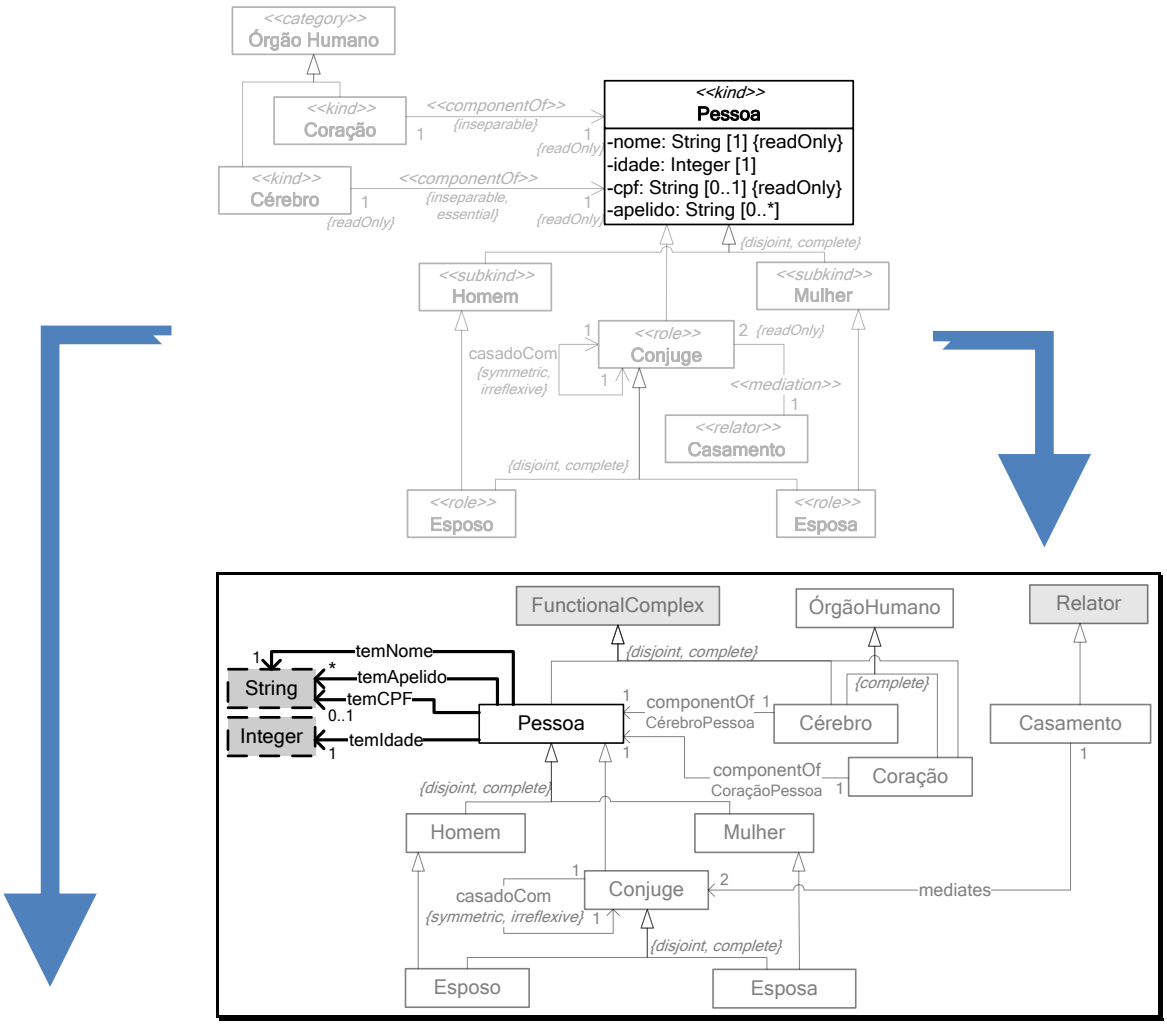


Figura 60. Modelo resultante da aplicação total do mapeamento para cenário estático simples para o exemplo da Figura 44.

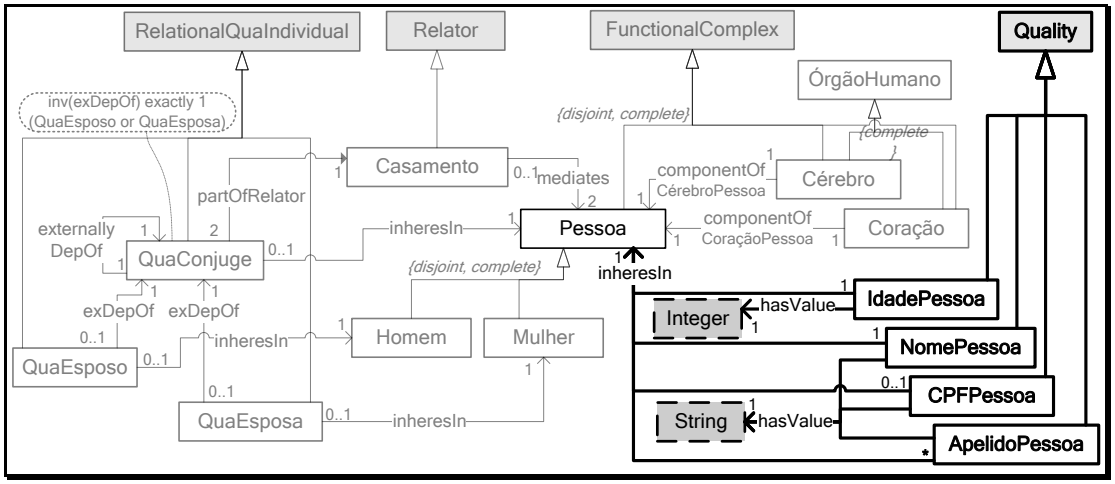


Figura 61. Modelo resultante da aplicação total do mapeamento para cenário estático reificado para o exemplo da Figura 44.

4.2. MAPEAMENTOS PARA CENÁRIO DINÂMICO BASEADOS NA ABORDAGEM 4D

Esta seção apresenta as alternativas de mapeamento seguindo a abordagem 4D para representação de informação temporal. A estrutura base é obtida estendendo-se aquela da abordagem 4D (classes em cinza claro e as relações entre elas), com a estrutura base do cenário estático simples apresentada na Figura 45, conforme mostrado na Figura 62.

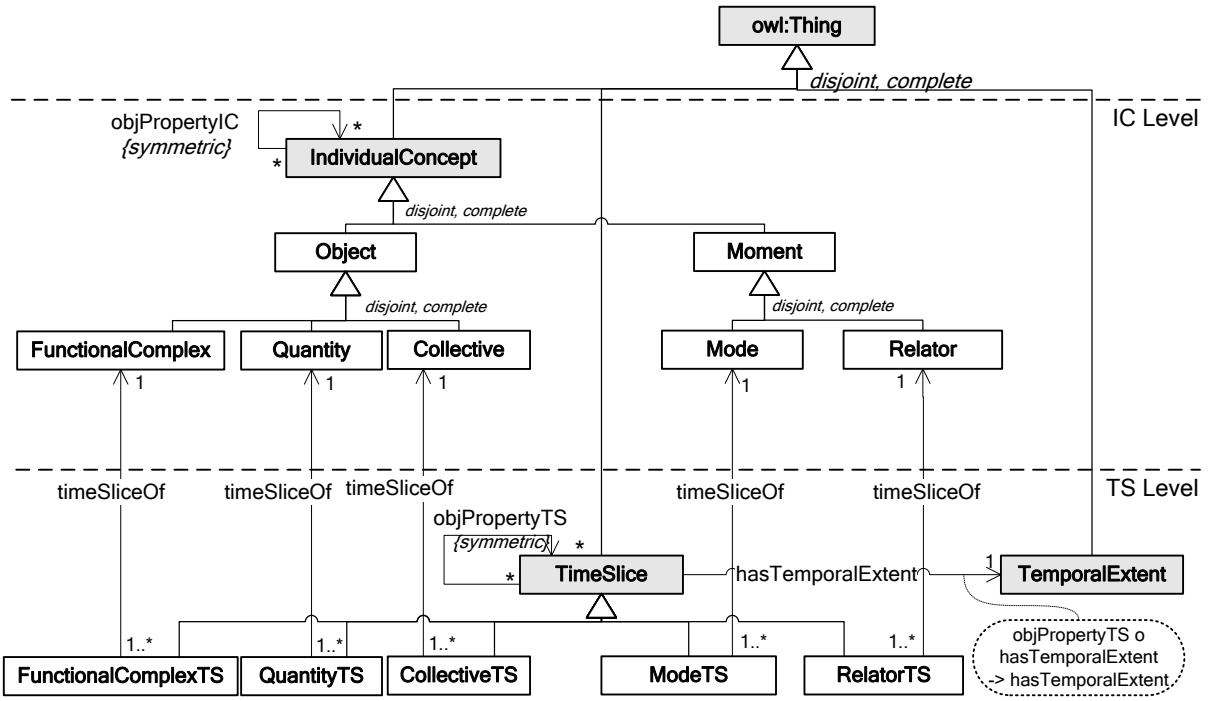


Figura 62. Esquema estilo UML para representação da abordagem 4D em OWL baseando-se na linguagem OntoUML.

Todas as diretrizes gerais da abordagem 4D são respeitadas nesta estrutura base. Embora a estrutura do cenário estático seja representada no nível IC, e apenas as classes-folha sejam refletidas no nível TS, considera-se atendida a diretriz **d1**, que em particular exige que todo indivíduo do nível IC tenha pelo menos uma parte temporal. Isso porque as classes não-folha são completas em relação à classe folha, ou seja, qualquer indivíduo de uma classe não-folha é instância de uma das classes folha e, portanto, tem obrigatoriamente uma parte temporal. As outras diretrizes são obviamente satisfeitas. As relações entre as classes não são contempladas na estrutura base porque elas valem em níveis diferentes dependendo da alternativa 4D adotada.

O mapeamento das classes necessárias, do tipo sortal rígido, *mixin* rígido e *moment* é idêntico para todas as alternativas: elas são representadas nos dois níveis. O Quadro 9 repete o mapeamento para classes do tipo *kind* no cenário estático simples (em cor de fundo cinza), e apresenta como fica o mapeamento no cenário dinâmico (em cor de fundo azul), para destacar o que efetivamente muda de um pro outro (em vermelho)[11]. Neste caso, basta repetir as diretrizes indicadas considerando-se também as classes do nível TS, e acrescentar as restrições quanto à instanciação da propriedade *timeSliceOf* **(dsf4)**, ou seja, cada classe do nível IC só pode ter como parte temporal indivíduos da sua contraparte no nível TS, bem como cada classe no nível TS só pode ser parte de um único indivíduo da sua contraparte no nível IC. Este mapeamento atende às diretrizes gerais **d1.1** e **d1.2**, e às específicas de cada alternativa **a_.1** e **a_.2**[12].

« kind » K	**Cenário Estático**	**Cenário Dinâmico - Abordagem 4D**
	Sejam K_1 .. K_n as classes do tipo *kind*: **(dsf1) DisjointClasses**(K_1 .. K_n) **(dsf2) SubClassOf**(K_i FunctionalComplex), para i de 1 até n **(dsf3) EquivalentClasses**(FunctionalComplex **ObjectUnionOf**(K_1 .. K_n))	Sejam K_1 .. K_n as classes do tipo *kind*: **(dsf1) DisjointClasses**(K_1 .. K_n) **DisjointClasses**(KTS_1 .. KTS_n) **(dsf2) SubClassOf**(K_i FunctionalComplex) **SubClassOf**(KTS_i FunctionalComplexTS), para i de 1 até n **(dsf3) EquivalentClasses**(FunctionalComplex **ObjectUnionOf**(K_1 .. K_n)) **EquivalentClasses**(FunctionalComplexTS **ObjectUnionOf**(KTS_1 .. KTS_n)) **(dsf4) SubClassOf**(K_i **ObjectAllValuesFrom** (**ObjectInverseOf** (timeSliceOf) KTS_i)) **SubClassOf**(KTS_i **ObjectSomeValuesFrom** (timeSliceOf K_i)), para i de 1 até n {*timeSliceOf é funcional*}

Quadro 9. Mapeamento das classes OntoUML do tipo *kind* para o cenário dinâmico segundo a abordagem 4D.

[11] Este padrão é repetido nos outros quadro de mapeamento apresentados neste capítulo,

[12] Notação abreviada para as repectivas diretrizes de cada alternativa, e.g, **a1.1** e **a1.2.**

Por outro lado, as classes do tipo sortal anti-rígido e *mixin* não-rígido são aquelas contingentes representadas apenas no nível dinâmico. Conforme ilustrado no Quadro 10 apenas para classes do tipo *role*, basta seguir as diretrizes indicadas apenas considerando-se as classes com sufixo TS. Este mapeamento atende às diretrizes gerais **d1.2**, e às específicas de cada alternativa **a_.2**.

« role » R	Cenário Estático	Cenário Dinâmico - Abordagem 4D
	Para cada partição P_i de classes do tipo *role*, sejam R_1 .. R_n as classes da partição e S a classe sortal particionada: **(dsar1)** **SubClassOf**(R_i S), para i de 1 até n **(dsnf2)** Se P_i é disjunta: **DisjointClasses**(R_1 .. R_n) Se P_i é completa: **EquivalentClasses**(S **ObjectUnionOf** (R_1 .. R_n))	Para cada partição P_i de classes do tipo *role*, sejam R_1 .. R_n as classes da partição e S a classe sortal particionada: **(dsar1)** **SubClassOf**(RTS_i STS), para i de 1 até n **(dsnf2)** Se P_i é disjunta: **DisjointClasses**(RTS_1 .. RTS_n) Se P_i é completa: **EquivalentClasses**(STS **ObjectUnionOf** (RTS_1 .. RTS_n))

Quadro 10. Mapeamento das classes OntoUML do tipo *role* para o cenário dinâmico segundo a abordagem 4D.

Alternativa 4D-A0

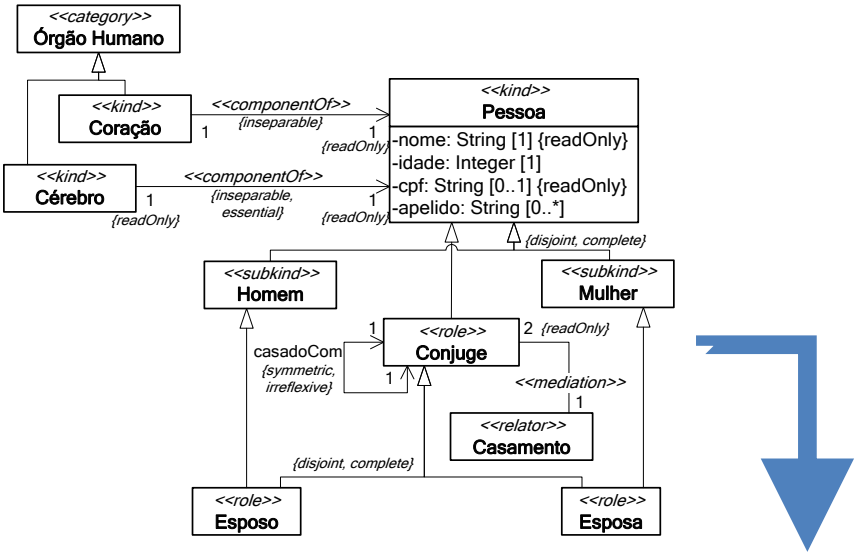
Esta alternativa é a que possui mapeamento mais imediato: basta seguir as diretrizes do mapeamento das relações e atributos para cenário estático simples no nível dinâmico. Na prática, se todas as relações, inclusive as da estrutura, valem no nível dinâmico, então devem especializar a propriedade *objProperyTS*, e também as classes domínio e imagem das propriedades de objeto devem ser consideradas com o sufixo TS. Analogamente, a classe-domínio das propriedades de dados devem ser aquelas do nível TS. O Quadro 11 mostra como fica o mapeamento, apenas para relações do tipo *componentOf*. Este mapeamento atende à diretriz geral **d2**, e à específica de alternativa **a0.2**.

« componentOf »	Cenário Estático	Cenário Dinâmico - Alternativa 4D-A0
	merRel ← componentOf **(darm1)** **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D) **ObjectPropertyRange** (merRelDI I) **(da2) (darm2)** Se isEssential = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se isInseparable = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)	merRel ← componentOf **SubObjectPropertyOf** (merRelDI objectPropertyTS) **(darm1)** **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI DTS) **ObjectPropertyRange** (merRelDI ITS) **(da2) (darm2)** Se isEssential = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se isInseparable = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)

(da1) Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D)) Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minI merRelDI I)) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI merRelDI I))	**(da1)** Se minD > 0 **SubClassOf**(ITS **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) DTS)) Se maxD > 0 **SubClassOf**(ITS **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) DTS)) Se minI > 0 **SubClassOf**(DTS **ObjectMinCardinality**(minI merRelDI ITS)) Se maxI > 0 **SubClassOf**(DTS **ObjectMaxCardinality**(maxI merRelDI ITS))

Quadro 11. Mapeamento das associações OntoUML do tipo *componentOf* para o cenário dinâmico segundo a alternativa 4D-A0.

O modelo de exemplo da Figura 44 mapeado segundo a alternativa 4D-A0 é apresentado na Figura 63. A mesma representação do cenário estático simples da Figura 60 aparece agora no nível dinâmico.



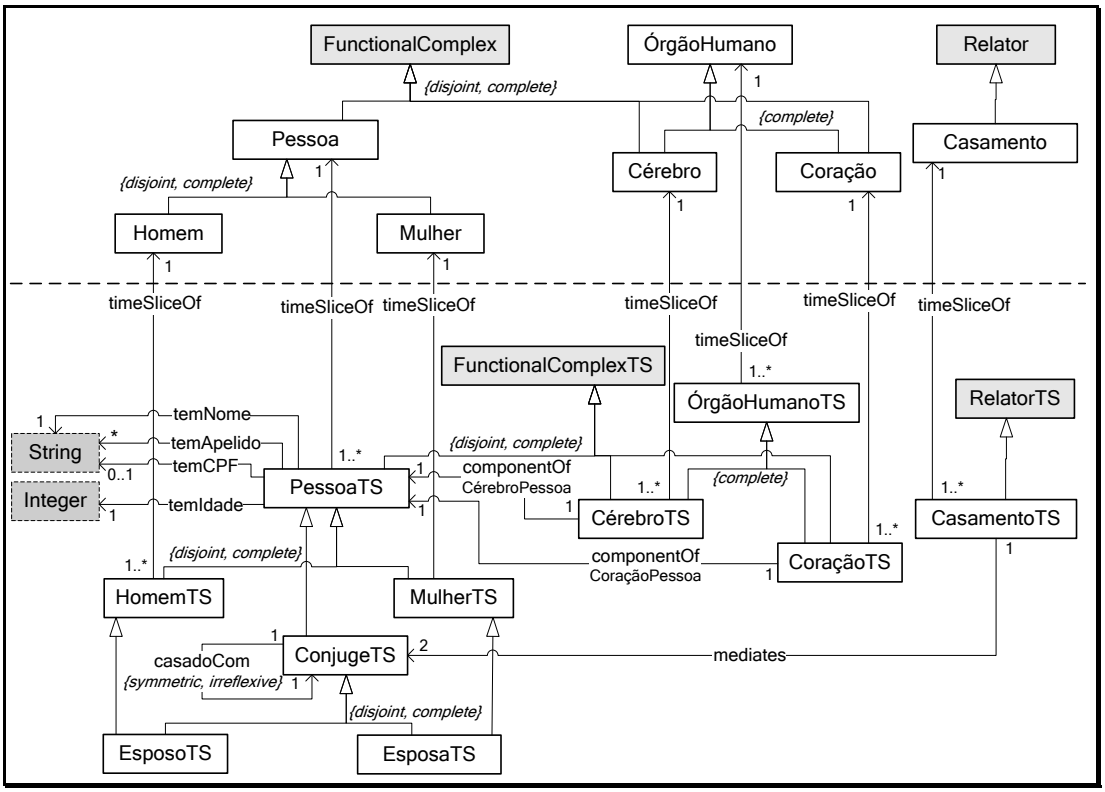


Figura 63. Modelo resultante da aplicação do mapeamento referente à alternativa 4D-A0 para o exemplo da Figura 44.

Alternativa 4D-A1

Esta alternativa, por sua vez, prevê a representação das relações que implicam em dependência existencial mútua e dos atributos necessários e imutáveis no nível estático. Na prática, se as relações implicam em dependência existencial mútua, elas valem no nível estático, e portanto especializam *objPropertyIC*, ou, caso contrário, elas valem no nível dinâmico, e portanto especializam a propriedade *objProperyTS*, sendo as classes referidas consideradas com o sufixo TS. O Quadro 12 mostra como fica o mapeamento, apenas para relações do tipo *componentOf*. Este mapeamento atende à diretriz geral **d2**, e às específicas de alternativa **a1.1** e **a1.2**.

« componentOf »	Cenário Estático	Cenário Dinâmico - Alternativa 4D-A1
	merRel ← componentOf **(darm1)** **SubObjectPropertyOf** (merRelDI merRel)	merRel ← componentOf Se [*isEssential* = *true*] E [*isInseparable* = *true*], então **SubObjectPropertyOf** (merRelDI objectPropertyIC) D' ← D I' ← I Senão **SubObjectPropertyOf** (merRelDI objectPropertyTS) D' ← DTS I' ← ITS **(darm1)**

	ObjectPropertyDomain (merRelDI D) **ObjectPropertyRange** (merRelDI I) **(da2) (darm2)** Se *isEssential* = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se *isInseparable* = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf) **(da1)** Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D)) Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minI merRelDI I)) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI merRelDI I))	**SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D') **ObjectPropertyRange** (merRelDI I') **(da2) (darm2)** Se *isEssential* = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se *isInseparable* = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf) **(da1)** Se minD > 0 **SubClassOf**(I' **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D')) Se maxD > 0 **SubClassOf**(I' **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D')) Se minI > 0 **SubClassOf**(D' **ObjectMinCardinality**(minI merRelDI I')) Se maxI > 0 **SubClassOf**(D' **ObjectMaxCardinality**(maxI merRelDI I'))

Quadro 12. Mapeamento das associações OntoUML do tipo *componentOf* no cenário simples dinâmico segundo a alternativa 4D-A1.

No caso das propriedades, se forem necessárias e imutáveis para uma classe rígida, elas são atribuídas no nível estático, caso contrário, são atribuídas às respectivas classes no nível dinâmico, ou seja, com sufixo TS. O Quadro 13 ilustra como fica o mapeamento das propriedades. Este mapeamento atende à diretriz geral **d2**, e às específicas de alternativa **a1.1** e **a1.2**.

Property Prop	**Cenário Estático**	**Cenário Dinâmico - Alternativa 4D-A1**
		Se *IsReadOnly* = *true* e D é do tipo rígido, então D' ← D Senão D' ← DTS
	(dp1) **DataPropertyDomain** (temProp D) **DataPropertyRange** (temProp *getType*)	**(dp1)** **DataPropertyDomain** (temProp D') **DataPropertyRange** (temProp *getType*)
	(dp2) Se min > 0 **SubClassOf**(D **DataMinCardinality**(min temProp)) Se max > 0 **SubClassOf**(D **DataMaxCardinality**(max temProp))	**(dp2)** Se min > 0 **SubClassOf**(D' **DataMinCardinality**(min temProp)) Se max > 0 **SubClassOf**(D' **DataMaxCardinality**(max temProp))

Quadro 13. Mapeamento para propriedades/atributos OntoUML no cenário dinâmico segundo a alternativa 4D-A1.

O modelo de exemplo da Figura 44 mapeado segundo a alternativa 4D-A1 é apresentado na Figura 64. Em comparação com o modelo da Figura 63, resultante da aplicação da alternativa A0, a relação *componentOf* entre **Cérebro** e **Pessoa**, que implica em dependência existencial mútua, é representada no nível estático, bem como a propriedade (atributo) **nome**, que é necessária e imutável.

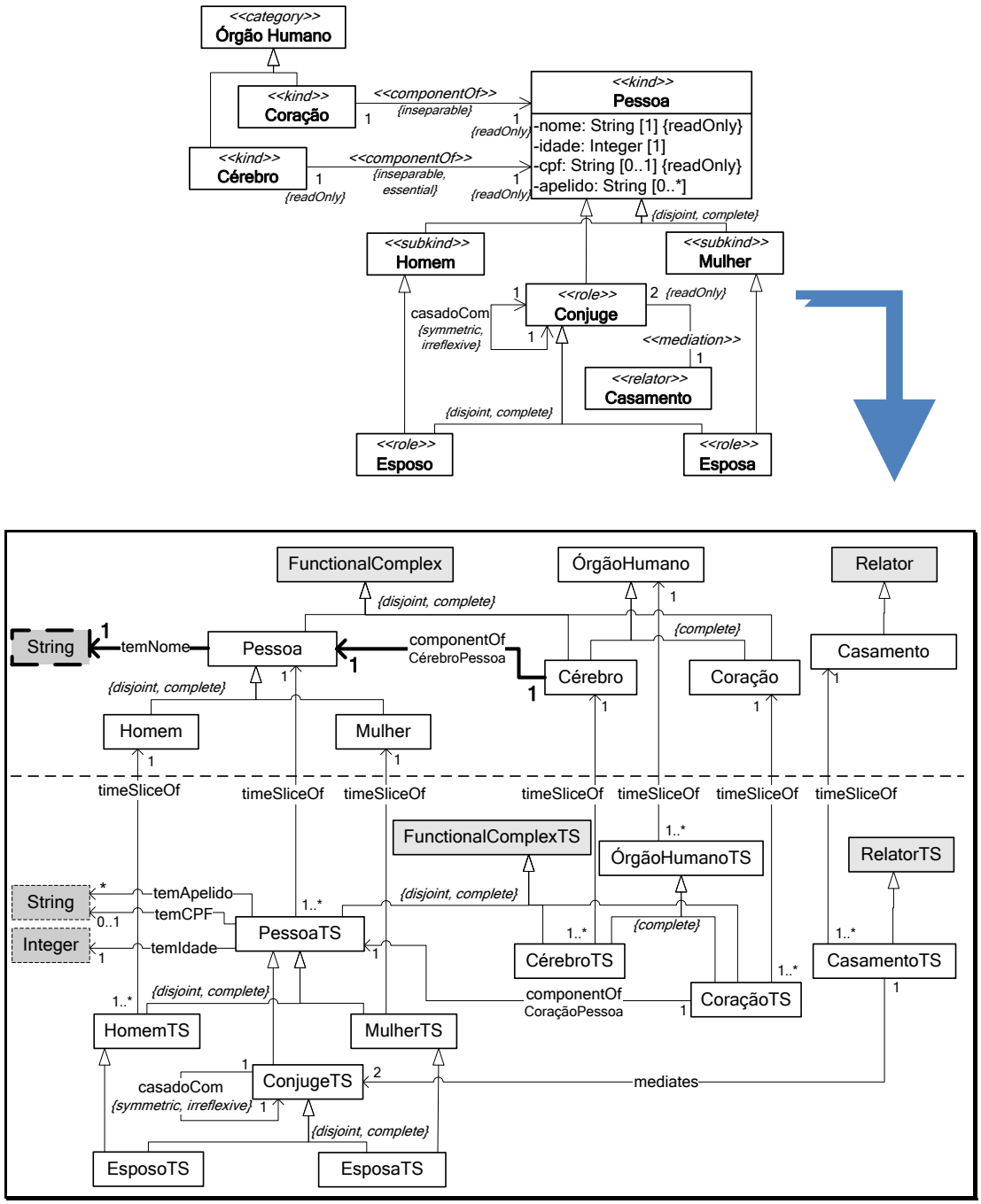


Figura 64. Modelo resultante da aplicação do mapeamento referente à alternativa 4D-A1 para o exemplo da Figura 44.

Alternativa 4D-A2

Finalmente, a alternativa 4D-A2 prevê a representação das relações que implicam em dependência existencial e dos atributos necessários e imutáveis no nível estático. Na prática, se as relações implicam em dependência existencial, elas valem no nível estático, e, portanto, especializam a propriedade *objPropertyIC*, ou, caso contrário, elas valem no nível dinâmico, e, portanto, especializam a propriedade *objProperyTS*, sendo as classes referidas consideradas com o sufixo TS. As relações de dependência existencial unilateral têm a restrição de cardinalidade máxima flexibilizada para o final de associação ligado ao indivíduo dependente da relação (se for mutável), de forma que o indivíduo não-dependente pode vir a instanciar a mesma relação com outros indivíduos. Além disso, se o domínio ou imagem destas relações no nível estático não é rígido, deve-se ser considerar as super-classses rígidas mais próximas na hierarquia, que têm representação neste nível, sem porém restringir-lhes a cardinalidade mínima, pois tais relações lhes são opcionais e não necessárias.

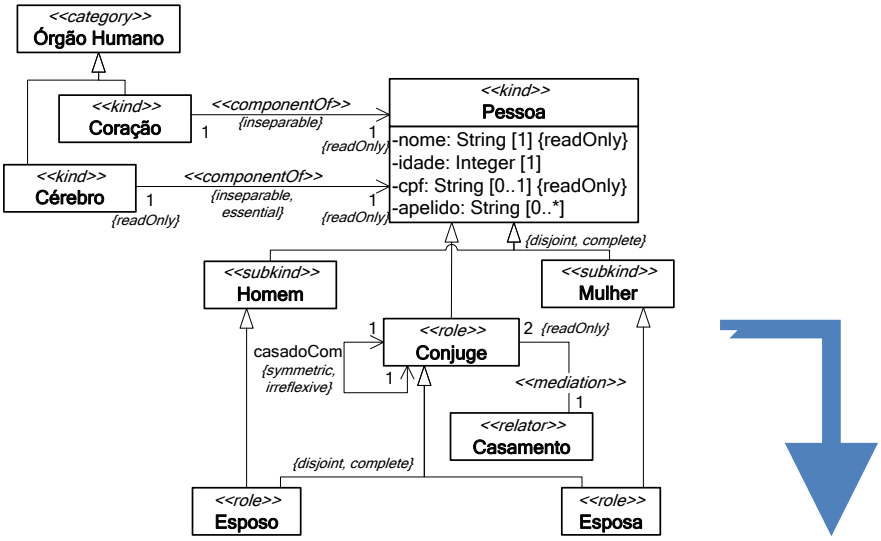
O Quadro 14 mostra como fica o mapeamento, apenas para relações do tipo *componentOf*. As mesmas condições devem ser averiguadas para os outros tipos de relações, considerando-se que para as relações não-meronímicas não existem as meta-propriedades *isEssential*, *isInseparable*. Neste caso, a dependência existencial é verificada se a relação é necessária (cardinalidade mínima maior que zero) e imutável (*isReadOnly* tem valor verdadeiro) para domínio ou imagem rígidos. Este mapeamento atende à diretriz geral **d2**, e às específicas de alternativa **a2.1**, **a2.2** e **a2.3**.

« componentOf »	Cenário Estático Simples	Cenário Dinâmico - Alternativa 4D-A2
	merRel ← componentOf	merRel ← componentOf Se [*isEssential* = *true*] OU [*isInseparable* = *true*], então **SubObjectPropertyOf** (merRelDI objectPropertyIC) D' ← D I' ← I Se [*isReadOnly* = *false*] para o final de associação ligado a D, então maxD ← -1 Se [*isReadOnly* = *false*] para o final de associação ligado a I, então maxI ← -1 Se D não é rígido, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) Se I não é rígido, sejam IR_1 .. IR_n as super-classes rígidas mais próximas na hierarquia de D. I' ← **ObjectUnionOf**(IR_1 .. IR_n) Senão

	SubObjectPropertyOf (merRelDI objectPropertyTS) D' ← DTS I' ← ITS
(darm1) **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D) **ObjectPropertyRange** (merRelDI I)	**(darm1)** **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D') **ObjectPropertyRange** (merRelDI I')
(da2) (darm2) Se *isEssential* = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se *isInseparable* = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)	**(da2) (darm2)** Se *isEssential* = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se *isInseparable* = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)
(da1) Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D)) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D)) Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minI merRelDI I)) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI merRelDI I))	**(da1)** Se [*isEssential* = *false* e *isInseparable* = *false*] ou [D é rígido e I é rígido] Se I é rígido Se minD > 0 **SubClassOf**(I' **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D')) Se maxD > 0 **SubClassOf**(I' **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D')) Se D é rígido Se minI > 0 **SubClassOf**(D' **ObjectMinCardinality**(minI merRelDI I')) Se maxI > 0 **SubClassOf**(D' **ObjectMaxCardinality**(maxI merRelDI I')) Senão {*dep. Exist. Unilateral*} Se D não é rígido Se minD > 0 **SubClassOf**(I' **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D')) Se maxD > 0 **SubClassOf**(I' **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D')) Se maxI > 0 **SubClassOf**(DRi **ObjectMaxCardinality**(maxI merRelDI I')), para i de 1 até n Se I não é rígido Se maxD > 0 **SubClassOf**(IRi **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D')), para i de 1 até n Se minI > 0 **SubClassOf**(D' **ObjectMinCardinality**(minI merRelDI I')) Se maxI > 0 **SubClassOf**(D' **ObjectMaxCardinality**(maxI merRelDI I'))

Quadro 14. Mapeamento das associações OntoUML do tipo *componentOf* no cenário simples dinâmico segundo a alternativa 4D-A2

O mapeamento das propriedades/atributos é idêntico ao da alternativa 4D-A1. O modelo de exemplo da Figura 44 mapeado segundo a alternativa 4D-A2 é apresentado na Figura 65. Em relação ao modelo da Figura 64, resultante da aplicação da alternativa A1, a relação *componentOf* entre **Coração** e **Pessoa**, que implica em dependência existencial unilateral, também é representada no nível estático, bem como a relação de mediação entre **Casamento** e **Pessoa**. Porém, por serem mutáveis com respeito ao final de associação ligado ao domínio, tais relações tem a cardinalidade máxima deste flexibilizada (destacadas em vermelho no modelo),de forma que uma **pessoa** possa ter mais de um **coração** (ou **casamento**) em diferentes instantes de tempo. Particularmente, a relação de mediação tem originalmente como imagem o conceito **Cônjuge** que é anti-rígido. Assim, considera-se como imagem no nível estático a sua super-classe rígida **Pessoa**. Entretanto, neste caso, a restrição de cardinalidade mínima também é flexibilizada, pois esta relação é opcional para a classe **Pessoa**.



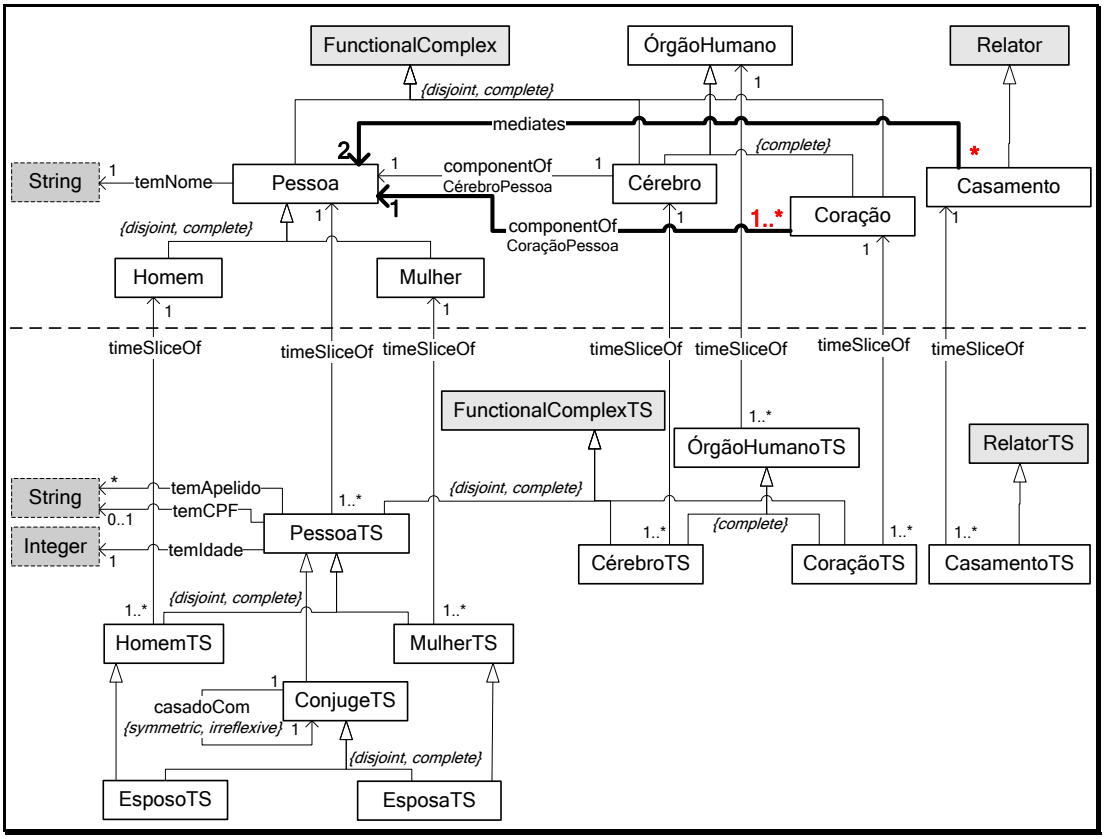


Figura 65. Modelo resultante da aplicação do mapeamento referente à alternativa 4D-A2 para o exemplo da Figura 44.

4.3. MAPEAMENTO PARA CENÁRIO DINÂMICO BASEADO NA ABORDAGEM DE REIFICAÇÃO TEMPORAL

Esta seção apresenta o mapeamento seguindo a abordagem de Reificação Temporal para representação de informação temporal. Para adaptar a estrutura base do cenário estático reificado apresentada na Figura 46 para representação de cenários dinâmicos conforme a Abordagem de Reificação Temporal, basta atribuir uma propriedade que represente o tempo de existência dos indivíduos, conforme apresentado na Figura 66 (as classes em cinza claro e as relações entre elas são da estrutura de cenário estático).

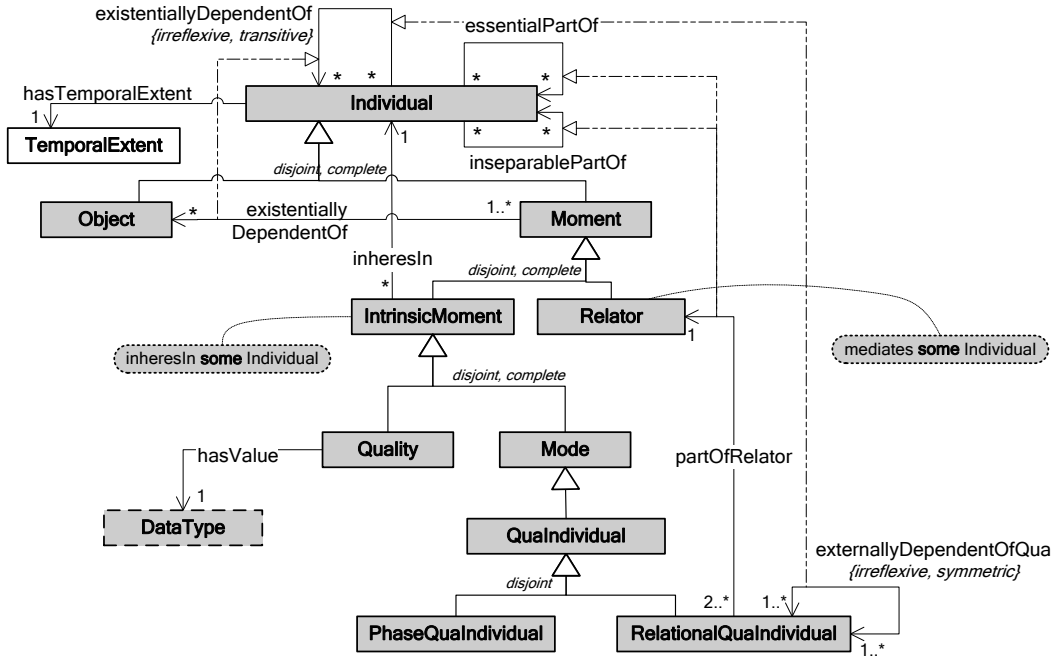


Figura 66. Esquema estilo UML para representação da abordagem de Reificação em OWL baseando-se na linguagem OntoUML, estendendo a estrutura OWL da Figura 46.

O mapeamento das classes necessárias, do tipo sortal rígido, *mixin* rígido e *moment* é idêntico ao do cenário estático reificado. Por outro lado, para as classes contingentes, do tipo sortal anti-rígido e *mixin* não-rígido, existe uma pequena mudança. Ela consiste da necessidade de remover a restrição de que dada uma partição disjuntas de classes contingentes, pode haver apenas uma instância de *qua-individual* relativo a tais classes a cada momento, inerente a uma instância de S, caso S seja rígido, ou dependente existencialmente de QuaS, caso S seja anti-rígido. Conforme ilustrado no Quadro 15 apenas para classes do tipo *role*, basta remover as respectivas partes do mapeamento.

« role » R	Cenário Estático Reificado	Cenário Dinâmico Reificado
	Sejam R_1 .. R_n as classes do tipo *role*: **(dsar0)** **SubClassOf**(QuaR$_i$ RelationalQuaIndividual), para i de 1 até n **(dsar1)** Sejam SR_{i1} .. SR_{im} as super-classes sortais rígidas mais próximas na hierarquia de R_i, **SubClassOf**(QuaR$_i$ **ObjectSomeValuesFrom**(inheresIn **ObjectUnionOf** (SR_{i1} .. SR_{im}))), para i de 1 até n {*inheresIn é funcional*} Para cada partição P de classes do tipo *role*,	Sejam R_1 .. R_n as classes do tipo *role*: **(dsar0)** **SubClassOf**(QuaR$_i$ RelationalQuaIndividual), para i de 1 até n **(dsar1)** Sejam SR_{i1} .. SR_{im} as super-classes sortais rígidas mais próximas na hierarquia de R_i, **SubClassOf**(QuaR$_i$ **ObjectSomeValuesFrom**(inheresIn **ObjectUnionOf** (SR_{i1} .. SR_{im}))), para i de 1 até n {*inheresIn é funcional*} Para cada partição P de classes do tipo *role*,

	sejam R_1 .. R_q as classes da partição e S a classe sortal particionada: **(dsar2)** Se S é uma classe rígida: Se P é disjunta: **SubClassOf**(S **ObjectMaxCardinality**(1 **InverseObjectProperty**(inheresIn) **ObjectUnionOf** ($QuaR_1$ .. $QuaR_q$))) Se P é completa: **SubClassOf**(S **ObjectMinCardinality**(1 **InverseObjectProperty**(inheresIn) **ObjectUnionOf** ($QuaR_1$ .. $QuaR_q$))) Senão (se S não é rígido, então é reificado) **SubClassOf**($QuaR_k$ **ObjectExactCardinality**(1 existentiallyDependentOf QuaS)), para k de 1 até q Se P é disjunta: **SubClassOf**(QuaS **ObjectMaxCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** ($QuaR_1$ .. $QuaR_q$))) Se P é completa: **SubClassOf**($QuaS_j$ **ObjectMinCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** ($QuaR_1$ .. $QuaR_q$)))	sejam R_1 .. R_q as classes da partição e S a classe sortal particionada: **(dsar2)** Se S é uma classe rígida: Se P é completa: **SubClassOf**(S **ObjectMinCardinality**(1 **InverseObjectProperty**(inheresIn) **ObjectUnionOf** ($QuaR_1$ .. $QuaR_q$))) Senão (se S não é rígido, então é reificado) **SubClassOf**($QuaR_k$ **ObjectExactCardinality**(1 existentiallyDependentOf QuaS)), para k de 1 até q Se P é completa: **SubClassOf**($QuaS_j$ **ObjectMinCardinality**(1 invExistentiallyDependentOf **ObjectUnionOf** ($QuaR_1$ .. $QuaR_q$)))

Quadro 15. Mapeamento das classes OntoUML do tipo *role* para o cenário dinâmico segundo a abordagem de Reificação Temporal.

Ademais, uma vez que se assume que todas as relações implicam em dependência existencial, basta interpretá-las como válidas durante o tempo de existência do dependente. Entretanto, para as relações mutáveis, ou que não implicam em dependência mútua, deve-se também flexibilizar a cardinalidade máxima do final de associação ligado ao indivíduo dependente da relação, no sentindo de que o indivíduo independente pode vir a instanciar a mesma relação com outros indivíduos. O Quadro 16 mostra como fica o mapeamento, apenas para relações do tipo *componentOf*. As mesmas condições devem ser averiguadas para os outros tipos de relações, considerando-se que para as relações não-meronímicas não existem as meta-propriedades *isEssential*, *isInseparable*. Neste caso, a dependência existencial é verificada se a relação é necessária (cardinalidade mínima maior que zero) e imutável (*isReadOnly* tem valor verdadeiro) para domínio ou imagem rígidos. Este mapeamento atende à diretriz **d3.3** da abordagem de Reficação Temporal.

« componentOf »	Cenário Estático Reificado	Cenário Dinâmico Reificado
	merRel ← componentOf	merRel ← componentOf Se [*isReadOnly* = *false*] para o final de associação ligado a D, então maxD ← -1 Se [*isReadOnly* = *false*] para o final de associação ligado a I, então maxI ← -1
	(darm0) Se D é rígido, D' ← D senão, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) Se I é rígido, I' ← I senão, sejam IR_1 .. IR_n as super-classes rígidas mais próximas na hierarquia de I. I' ← **ObjectUnionOf**(IR_1 .. IR_n)	**(darm0)** Se D é rígido, D' ← D senão, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) Se I é rígido, I' ← I senão, sejam IR_1 .. IR_n as super-classes rígidas mais próximas na hierarquia de I. I' ← **ObjectUnionOf**(IR_1 .. IR_n)
	(darm1) **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D') **ObjectPropertyRange** (merRelDI I')	**(darm1)** **SubObjectPropertyOf** (merRelDI merRel) **ObjectPropertyDomain** (merRelDI D') **ObjectPropertyRange** (merRelDI I')
	(da2) (darm2) Se *isEssential* = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se *isInseparable* = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)	**(da2) (darm2)** Se *isEssential* = true então **SubObjectPropertyOf**(merRelDI essentialPartOf) Se *isInseparable* = true então **SubObjectPropertyOf**(merRelDI inseparablePartOf)
	(da1) Se I é rígido, Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D')) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D')) Se D é rígido, Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minI merRelDI I')) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI merRelDI I'))	**(da1)** Se I é rígido, Se minD > 0 **SubClassOf**(I **ObjectMinCardinality**(minD **ObjectInverseOf** (merRelDI) D')) Se maxD > 0 **SubClassOf**(I **ObjectMaxCardinality**(maxD **ObjectInverseOf** (merRelDI) D')) Se D é rígido, Se minI > 0 **SubClassOf**(D **ObjectMinCardinality**(minI merRelDI I')) Se maxI > 0 **SubClassOf**(D **ObjectMaxCardinality**(maxI merRelDI I'))

Quadro 16. Mapeamento das associações OntoUML do tipo *componentOf* no cenário dinâmico segundo a abordagem de Reificação Temporal.

No caso das propriedades, se forem mutáveis, deve-se também flexibilizar a cardinalidade máxima relativa à quantidade de *qualities* deste tipo que podem ser inerentes ao indivíduo. O Quadro 17 ilustra como fica o mapeamento das propriedades. Este mapeamento atende à diretriz geral **d5.1** da abordagem de Reificação Temporal.

Property Prop	**Cenário Estático Reificado**	**Cenário Dinâmico Reificado**
		Se [*isReadOnly* = *false*] max ← -1
	(dp1) **SubDataPropertyOf** (PropD Quality)	**(dp1)** **SubDataPropertyOf** (PropD Quality)
	(dp2) Se D é rígido, D' ← D senão, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) **SubClassOf**(PropD **ObjectSomeValuesFrom** (inheresIn D')) {*inheresIn é funcional*}	**(dp2)** Se D é rígido, D' ← D senão, sejam DR_1 .. DR_n as super-classes rígidas mais próximas na hierarquia de D. D' ← **ObjectUnionOf**(DR_1 .. DR_n) **SubClassOf**(PropD **ObjectSomeValuesFrom** (inheresIn D')) {*inheresIn é funcional*}
	(dp3) **SubClassOf**(PropD **DataSomeValuesFrom** (hasValue *getType*)) {*hasValue é funcional*}	**(dp3)** **SubClassOf**(PropD **DataSomeValuesFrom** (hasValue *getType*)) {*hasValue é funcional*}
	(dp4) Se D é rígido, Se min > 0 **SubClassOf**(D **ObjectMinCardinality**(min **ObjectInverseOf** (inheresIn) PropD)) Se max > 0 **SubClassOf**(D **ObjectMaxCardinality**(max **ObjectInverseOf** (inheresIn) PropD))	**(dp4)** Se D é rígido, Se min > 0 **SubClassOf**(D **ObjectMinCardinality**(min **ObjectInverseOf** (inheresIn) PropD)) Se max > 0 **SubClassOf**(D **ObjectMaxCardinality**(max **ObjectInverseOf** (inheresIn) PropD))

Quadro 17. Mapeamento para propriedades/atributos OntoUML no cenário dinâmico segundo a abordagem de Reificação Temporal.

O modelo de exemplo da Figura 44 mapeado segundo a abordagem de reificação é apresentado na Figura 67. O resultado é semelhante ao mapeamento para cenário estático reificado apresentado na Figura 61, porém com as restrições de cardinalidade máxima flexibilizadas quando se trata de relações ou atributos mutáveis, por exemplo, uma **pessoa** pode ter mais de um **coração** durante sua existência, e também pode ser mediada por mais de um **casamento**.

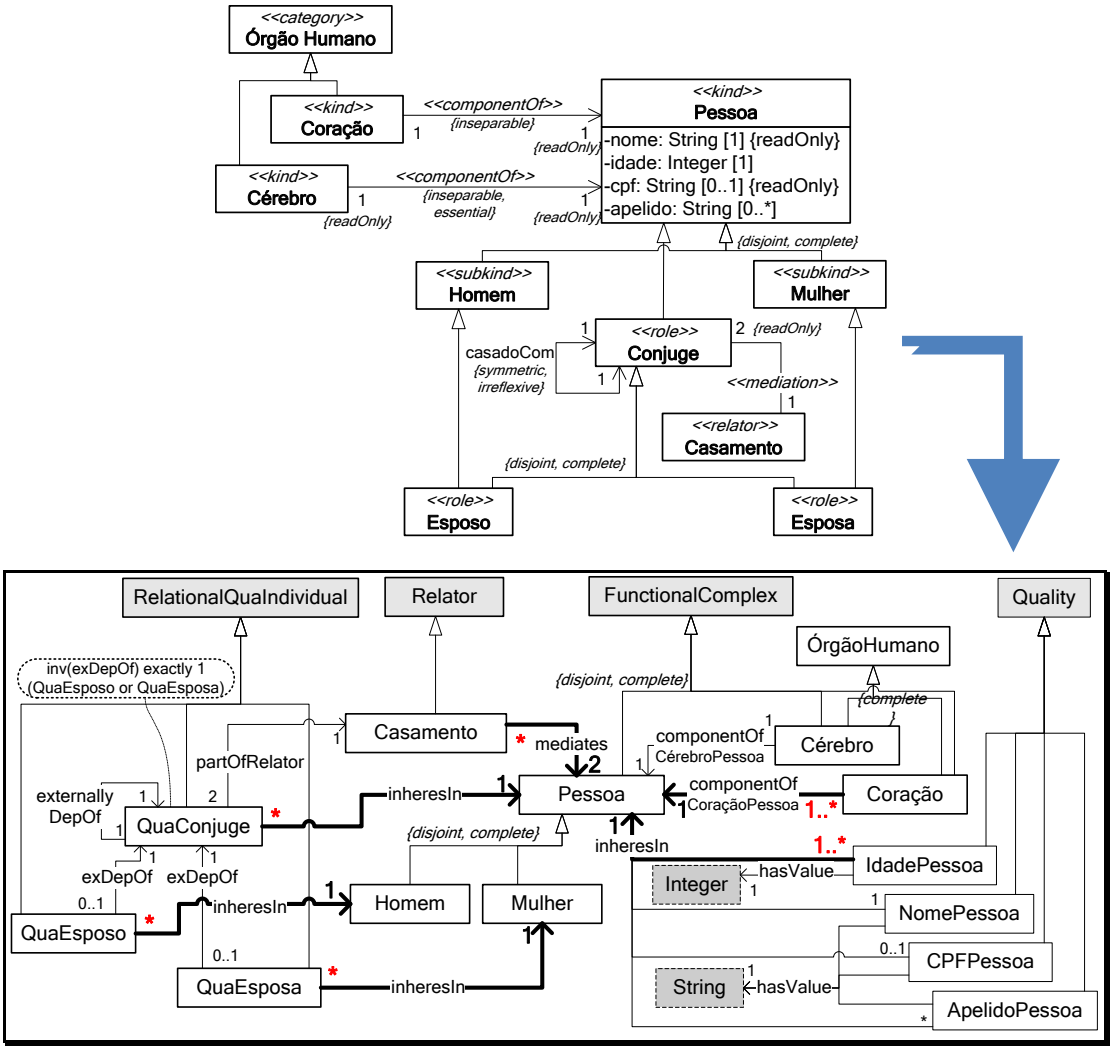


Figura 67. Modelo resultante da aplicação do mapeamento referente à abordagem de reificação para o exemplo da Figura 44.

4.4. IMPLEMENTAÇÃO DOS MAPEAMENTOS

Os mapeamentos propostos estão implementados na linguagem Java, como complemento à infra-estrutura OntoUML proposta por Carraretto (2010) e os arquivos de código estão no apêndice I. A Figura 68 ilustra o sistema de mapeamento, que recebe como entrada um modelo OntoUML e um parâmetro indicando o tipo de mapeamento, e produz como saída um arquivo no formato OWL segundo a sua sintaxe de serialização XML (*eXtensible Markup Language*) (MOTIK; PATEL-SCHNEIDER; BECHHOFER; ET AL., 2009), seguindo as diretrizes conforme o tipo indicado.

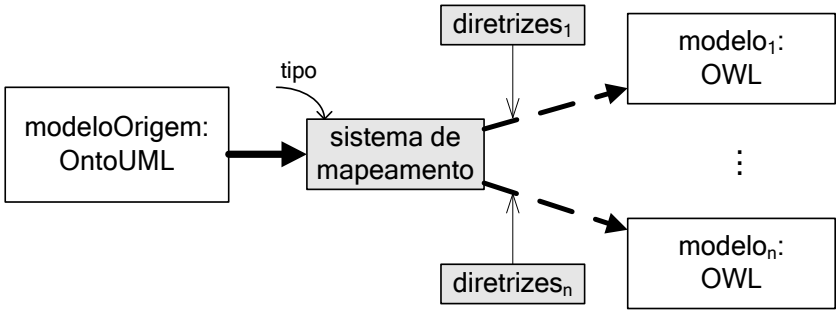


Figura 68. Sistema de mapeamento que transforma um modelo de origem OntoUML em um certo modelo OWL seguindo as diretrizes conforme o tipo de mapeamento indicado.

Um diagrama de sequência (simplificado), que ilustra as principais entidades envolvidas e qual papel elas desempenham no mapeamento, é apresentado na Figura 69. Uma instância da classe **OntoUML2OWL** é que coordena o mapeamento. Primeiramente é criada uma instância **tp** da Classe **TreeProcessor** a partir do modelo original (passo 1). Cria-se uma estrutura de árvore de especialização de classes (passo 1.1), cujos nós são instâncias de **NodeClass**, de forma que para cada classe sabe-se quais são seus atributos, os nós-pais, as partições e os nós-filhos. Cria-se também uma lista das associações binárias (passo 1.2), cujos nós são instâncias de **NodeBinAssociation** e indicam os respectivos nós-classe de origem e destino da associação.

Em seguida é criada uma instância **owl** da classe **OWLStructure** (passo 2), que é responsável por realizar o mapeamento dos elementos do modelo, contidos em **tp**, para a sintaxe xml de OWL segundo o tipo de mapeamento indicado. Para tanto, **owl** inicia a estrutura base criando internamente representações para as classes e propriedades conforme o tipo de mapeamento, que serão especializadas pelas entidades do modelo original, bem como as restrições de instanciação (passo 2.1). Então é chamada a sua função de mapeamento (passo 3), que consiste em criar internamente representações para as classes, atributos e associações do modelo também conforme o tipo de mapeamento. Finalmente, a estrutura owl resultante é verbalizada em termos da sintaxe concreta de OWL (passo 4) e seu resultado é escrito em arquivo (passos 5 e 6).

Os modelos OWL relativos ao mapeamento do modelo OntoUML da Figura 44, seguindo-se os mapeamentos baseados nas abordagem 4D e de Reficação, apresentados no apêndice II, foram automaticamente gerados por este sistema de mapeamento.

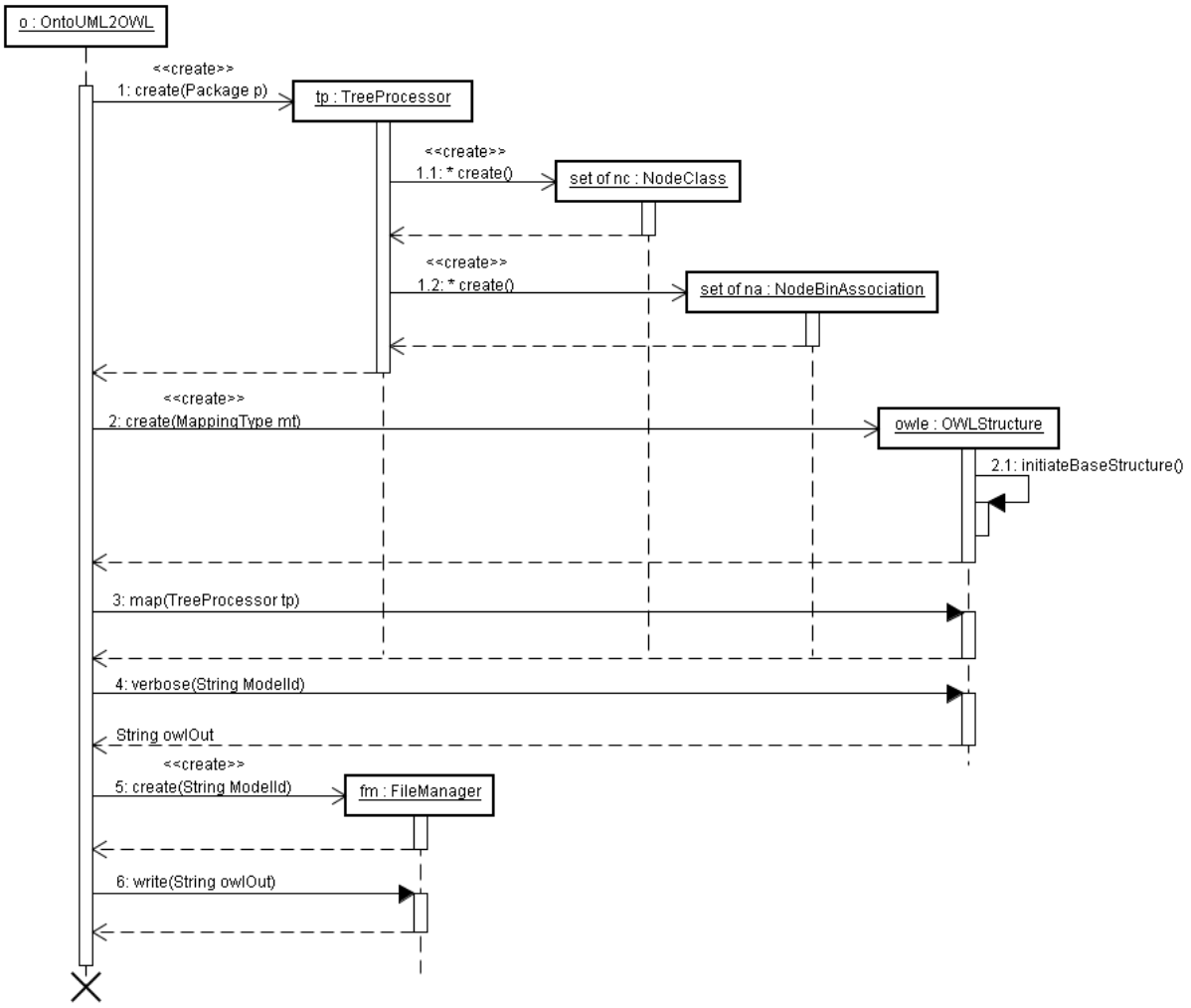


Figura 69. Diagrama de sequência simplificado do sistema de mapeamento.

4.5. CONCLUSÕES DO CAPÍTULO

Conclui-se que, uma vez adotada uma abordagem de Engenharia de Ontologias, em que o domínio é inicialmente modelado sem preocupações computacionais utilizando-se uma linguagem de nível ontológico, para então ser implementado em uma linguagem de nível epistemológico de acordo com os requisitos da aplicação, é possível estabelecer mapeamentos sistemáticos para realizar a implementação de forma semi-automática. A qualidade do modelo OWL resultante é consequência direta da qualidade dos seguintes itens: (i) da ontologia de referência de domínio, que está relacionada à qualidade da linguagem; (ii) da escolha do mapeamento, em função das características deste e dos requisitos da aplicação, (iii) da especificação do mapeamento em si, que está relacionada à justificação da estrutura e das diretrizes; e (iv) da implementação do mapeamento.

Assim, as principais contribuições deste capítulo são (i) especificação de 2 alternativas de mapeamento para cenário estático, e 4 alternativas para cenário dinâmico, justificados com base na ontologia de fundamentação da linguagem de origem OntoUML e em duas abordagens ontologicamente fundamentadas para capturar informação temporal do modelo estrutural em OWL; e (ii) implementação do mapeamento;

5. Estudo de Caso: Ontologia de ECG

5. ESTUDO DE CASO: ONTOLOGIA DE ECG

Este capítulo descreve um estudo de caso no domínio de Eletrocardiografia. Este domínio foi escolhido pois, além de ter sido objeto de estudo em trabalhos anteriores (ver seção 1.2), é também um domínio interessante para exercitar as propriedades que se quer verificar neste trabalho. Assim, seção 5.1 introduz brevemente este domínio e apresenta uma versão simplificada de ontologia de ECG. Então, a seção 5.2 propõe a escolha de um tipo de transformação baseando-se nos requisitos apontados. Os resultados são apresentados na seção 5.3 e finalmente a seção 5.4 traz as conclusões do capítulo.

5.1. ELETROCARDIOGRAFIA E ONTOLOGIA DE ECG

Eletrocardiografia é a técnica de registro de sinais elétricos gerados pela atividade do coração. Enquanto um impulso elétrico cardíaco é conduzido no coração humano, correntes elétricas percorrem os tecidos próximos a ele e alcançam levemente a superfície do corpo. Assim, o potencial elétrico gerado pelo coração pode ser medido através de pequenos sensores (eletrodos) colocados em pontos opostos da superfície do corpo (GUYTON; HALL, 1996 apud GONÇALVES ET AL., 2007).

Chama-se de eletrocardiograma (ECG) o registro resultante da aplicação dessa técnica. Sabe-se ainda que várias cardiopatias podem ser identificadas por anomalias específicas em algumas características do ECG. Essas características são: (i) a morfologia das ondas e complexos que compõem um ciclo cardíaco; e (ii) o tempo dos eventos e variações de padrão ao longo de várias batidas (GESELOWITZ, 1989 apud GONÇALVES ET AL., 2007).

O avanço tecnológico permitiu a representação digital de registros de ECG. Em especial, de acordo com Geselowitz (1989 apud GONÇALVES ET AL., 2007), o ECG foi possivelmente o primeiro sinal de diagnóstico a ser estudado com o propósito de se fazer interpretação automática por programas computacionais. Finalmente, o avanço das tecnologias de informação e comunicação permitiu o surgimento da Telecardiologia, sendo a transmissão de ECG's um serviço médico de Cardiologia que pode ser fornecidos remotamente, de uma forma econômica e eficiente. "Esses serviços vão desde um simples monitoramento remoto, através da transmissão de um ECG (por meio das tecnologias móveis e sem fio), até a transmissão de ECG ambulatorial, possibilitando o telemonitoramento cardíaco a qualquer hora e em qualquer lugar" (GONÇALVES ET AL., 2007,

p. 2). Assim, utilizando-se a transmissão e interpretação automática de ECG's, torna-se possível agilizar a identificação de vários tipos de doenças cardíacas.

Contudo, nota-se que, para melhor explorar o potencial desses serviços, faz-se importante a padronização do armazenamento e transmissão dos registros ECG. Segundo Gonçalves et al., os formatos considerados padrões de referência em Cardiologia não favorecem a interoperabilidade semântica entre aplicações baseadas em ECG. Isso se dá, em geral, por se representar o domínio com modelos conceituais de baixa expressividade e clareza, e/ou por haver um forte acoplamento com ambientes de programação específicos e linguagens cujos metamodelos misturam conceitos de domínio com os de apresentação da informação.

Tendo em vista essas questões, Gonçalves et al. (2007) desenvolveram uma ontologia com o objetivo principal de se obter uma teoria de eletrocardiograma independente de aplicações específicas do domínio. Posteriormente, esta teoria foi aprimorada (GONÇALVES; ZAMBORLINI; GUIZZARDI, 2009) e usada para prover interoperabilidade entre padrões de ECG existentes (GONÇALVES ET AL., 2010). A ontologia de ECG apresentada nesta seção é uma simplificação desta teoria, que porém preserva as principais informações relevantes para servir como estudo de caso para este trabalho, a saber, informações a respeito do funcionamento do coração a partir da verificação dos ciclos do ECG.

Um ciclo completo de ECG é ilustrado na forma de onda contida na Figura 70. A forma de onda representa os valores de potencial elétrico medidos em cada observação em função do tempo. Assim, uma série de observações seqüenciais constitui uma forma de onda, na qual podem-se identificar ciclos. Um ciclo, que representa uma batida do coração, é composto por formas elementares que mapeiam os impulsos elétricos. Em particular, as formas chamadas onda P e complexo QRS mapeiam respectivamente os impulsos elétricos chamados SA, responsável pela contração dos átrios, e HisPurkinje, que se soma ao impulso SA e é responsável pela contração dos ventrículos.

Neste trabalho, a ontologia simplificada abstrai estes detalhes, representando-se apenas os ciclos, classificados de duas formas: (i) ciclo completo, quando mapeia o impulso elétrico cardíaco dito completo, que consiste do SA e do HisPurkinje, ou seja, quando o ciclo possui ambas as formas elementares onda P e complexo QRS então ocorre a contração total do coração; ou (ii) ciclo parcial, quando mapeia o impulso elétrico cardíaco dito parcial, que consiste apenas do HisPurkinje, ou seja, quando o ciclo possui o complexo QRS porém não a onda P, então ocorre a

contração parcial do coração que corresponde à contração apenas dos ventrículos (GONÇALVES ET AL., 2007; GONÇALVES; ZAMBORLINI; GUIZZARDI, 2009).

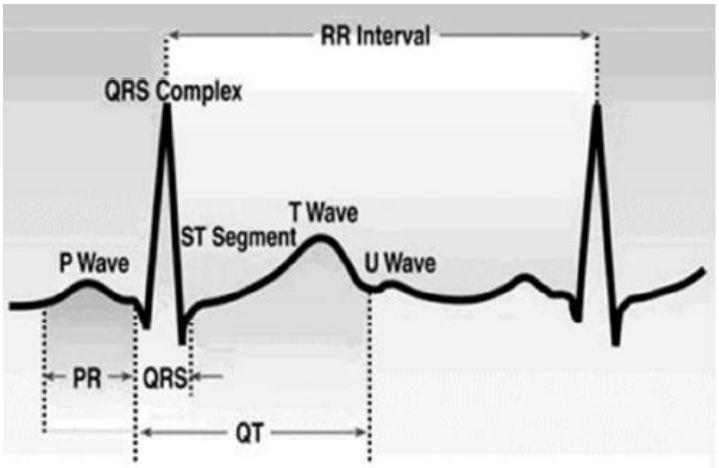


Figura 70. A forma de onda em um ciclo completo de ECG.
Fonte: Gonçalves et al. (2007, p. 7)

A ontologia de ECG apresentada na Figura 71. **Ontologia de ECG (simplificada)**

foi criada utilizando-se uma ferramenta existente baseada em modelos para OntoUML (CARRARETTO, 2010), em que foram implementados os mapeamentos propostos (ver seção 4.4). Esta ontologia descreve um cenário em que uma pessoa (**Person**), que tem (**componentOf**) necessariamente um coração (**Heart**), possui (**belongsTo**) opcionalmente um registro de ECG (**ECGRecord**). Este pode ser caracterizado (**characterization**) por um ciclo (**Cycle**) que necessária e imutavelmente mapeia (**maps**) um impulso elétrico cardíaco (**CardiacElectricalImpulse**). Neste cenário, destaca-se dois tipos de ciclo (**Cycle**), completo (**CompleteCycle**) e parcial (**PartialCycle**), que mapeiam respectivamente os tipos de impulso elétrico, completo (**CompleteCardiacElectricalImpulse**) e parcial (**PartialCardiacElectricalImpulse**).

Estes dois impulsos instanciam necessária e imutavelmente a relação material de contrair (**contracts**) o coração, derivada dos *relators* contração completa e parcial do coração (**CompleteHeartContraction** e **PartialHeartContraction**). Neste caso, diz-se que o coração desempenha ou o papel de coração funcionando completamente (**CompletelyFunctioningHeart**) ou o de coração funcionando parcialmente (**PartiallyFunctioningHeart**). As restrições de cardinalidade, quando omitidas, significam que cardinalidade é de exatamente um. O domínio da relação é aquele cujo final de associação tem o rótulo **src** (do inglês *source*) enquanto a imagem tem o rótulo **dst** (do inglês *destination*).

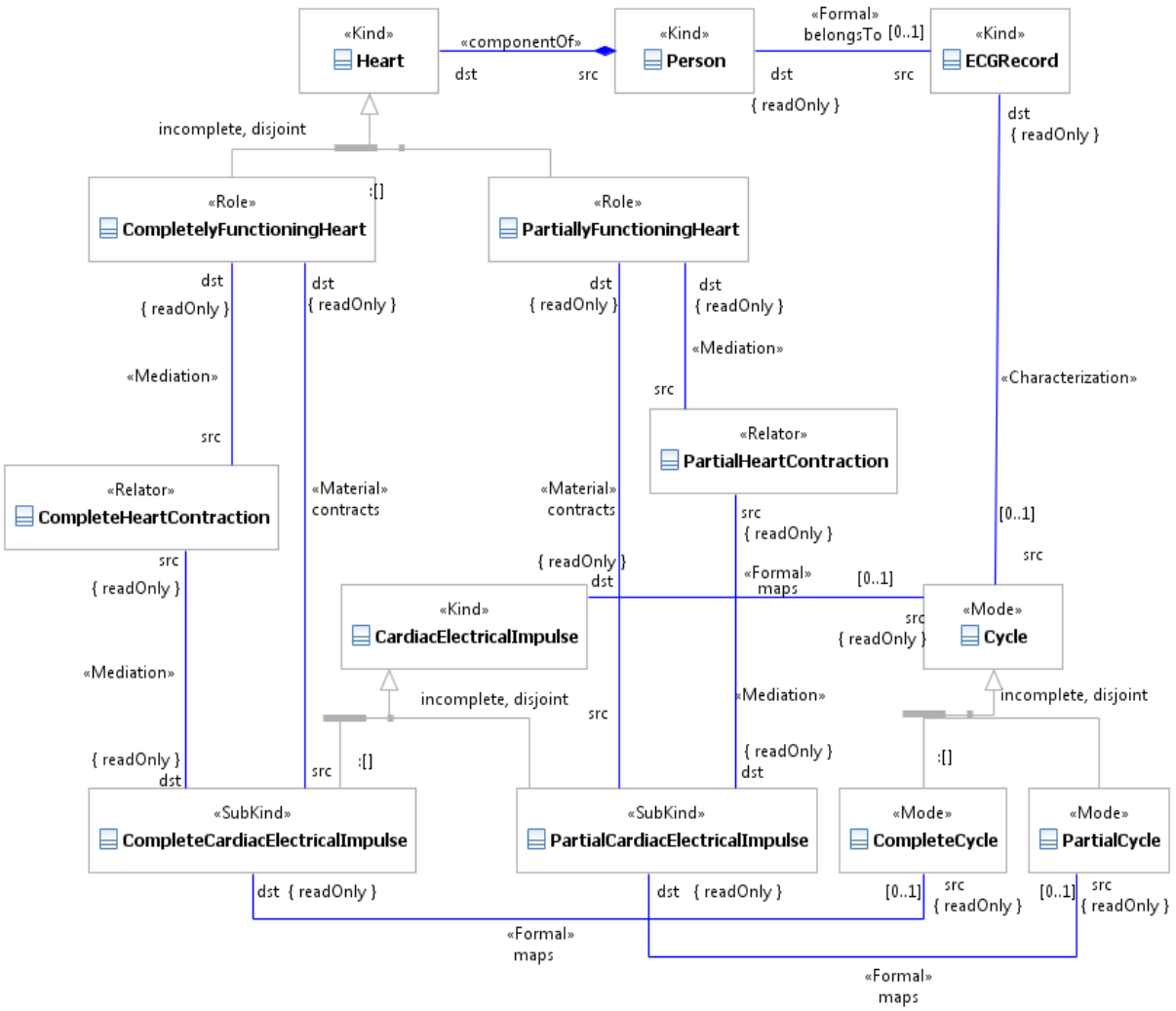


Figura 71. Ontologia de ECG (simplificada)

Por fim, destaca-se que não é o intuito da ontologia de ECG detectar as anomalias, mas prover a informação que viabilize tal intento. Conforme citado anteriormente, a identificação das anomalias envolve a verificação do funcionamento do coração ao longo do tempo baseando-se, por exemplo, nos dados dos ciclos registrados pelo ECG.

5.2. ESCOLHA DA TRANSFORMAÇÃO

Tendo em vista os requisitos apontados na seção anterior para aplicações de Eletrocardiografia, é de suma importância que o modelo de ECG, que dá suporte a tais aplicações, permita representar informações que mudam temporalmente. O modelo deve fornecer informação sobre o funcionamento do coração de uma pessoa a partir dos dados dos ciclos que caracterizam o seu

registro de ECG a cada momento. Consideram-se então como dados conhecidos a pessoa, o coração, o registro de ECG e os ciclos.

Com relação à escolha da abordagem, sendo o objetivo principal da aplicação inferir sobre o funcionamento do coração, tem-se que a abordagem 4D é indicada em detrimento da abordagem de Reificação, uma vez que esta não é apropriada para inferir informações contingentes.

Considerando-se a escolha da abordagem 4D, deve-se então escolher uma das alternativas. Como dentre as relações entre os dados disponíveis, que serão usados para inferência, há duas são de dependência existencial unilateral (**belongsTo** e **characterization**), a alternativa 4D-A2 torna-se desaconselhável, pois essas relações valeriam apenas no nível IC, o que requereria utilizar regras que capturassem a sobreposição das extensões temporais das fatias temporais, extrapolando o poder de expressividade de OWL. Neste caso, como tais relações não são de dependência existencial mútua, servem tanto a alternativa A0 quanto a A1.

5.3. RESULTADOS

Utilizando-se a alternativa 4D-A0, tem-se como resultado da aplicação do mapeamento automático à ontologia de ECG (Figura 71. **Ontologia de ECG (simplificada)**

) o modelo OWL apresentado na Figura 72. A característica principal desta alternativa é representar no nível TS um modelo similar ao estático, consistindo assim de um agrupamento de cenários estáticos em função dos conceitos individuais representados no nível IC.

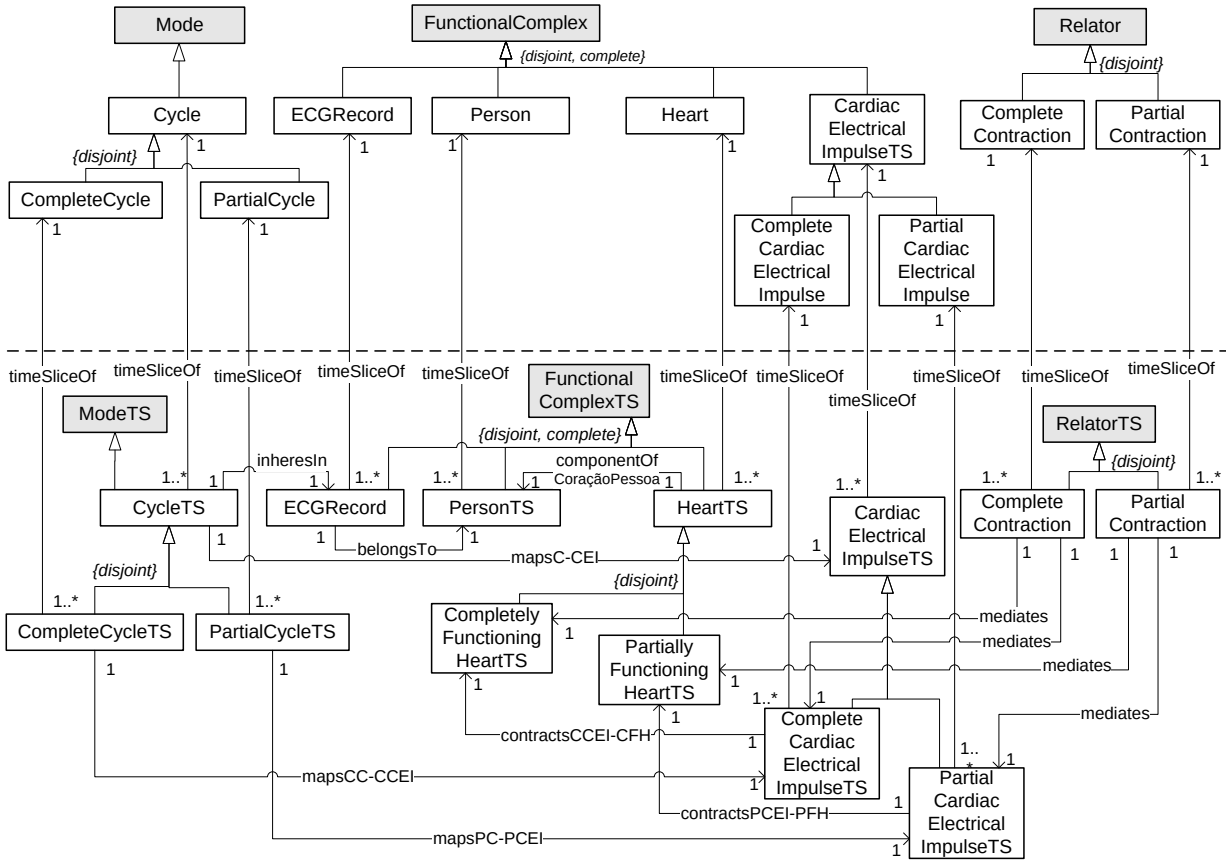


Figura 72. Esquema estilo UML da aplicação da alternativa A0 à ontologia da Figura 71. Ontologia de ECG (simplificada)

.

Para inferir o funcionamento do coração a partir dos dados fornecidos pela aplicação, é necessário acrescentar às classes **CompletelyFunctioningHeartTS** e **PartiallyFunctioningHeartTS** as seguintes restrições de classe equivalente:

$$\text{CompletelyFunctioningHeartTS} \equiv \text{componentOf some (PersonTS and inv(belongsTo) some (ECGRecordTS and inv(inheresIn) some CompleteCycleTS))}$$

$$\text{PartiallyFunctioningHeartTS} \equiv \text{componentOf some (PersonTS and inv(belongsTo) some (ECGRecordTS and inv(inheresIn) some PartialCycleTS))}$$

A Figura 73 ilustra uma possível instanciação do modelo da Figura 72. Os elementos mostrados em tons mais claros ilustram os dados não explicitamente declarados. Observa-se que é possível concluir que o coração em questão está funcionando completa e parcialmente no segundo e terceiro intervalos, uma vez que os ciclos ligados às partes temporais do coração nestes intervalos são do tipo completo e parcial respectivamente.

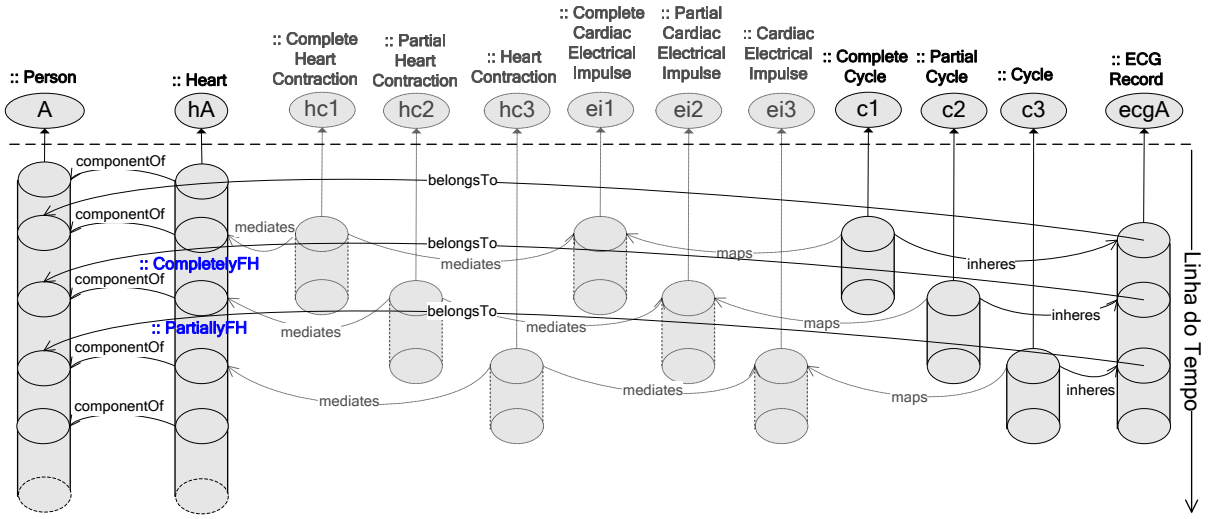


Figura 73. Esquema ilustrativo de instanciação do modelo da Figura 72.

O código OWL relativo ao modelo da Figura 72, gerado automaticamente, incluindo-se as restrições supracitas e a instanciação correspondente à apresentada na Figura 73, incluídas manualmente, é apresentado no apêndice III.

5.4. CONSIDERAÇÕES DO CAPÍTULO

Verifica-se neste estudo de caso particular que o uso de uma abordagem de Engenharia de Ontologias com mapeamentos sistemáticos pré-estabelecidos, dos quais são conhecidas as características, favorece alcançar o objetivo de um sistema no contexto de Eletrocardiografia. Tal procedimento torna possível fornecer informação sobre o funcionamento do coração não de forma estática, retratando um momento isolado, como outrora fora feito (ZAMBORLINI, 2008), mas de uma forma dinâmica, como uma sequência de momentos, que provê então a informação que viabiliza a detecção de possíveis anomalias.

Vale ressaltar ainda uma importante característica deste domínio que, porém, não é abordada aqui por estar fora do escopo deste trabalho. Esta característica é a imprecisão (*vagueness*) intrínseca ao conceito do coração em funcionamento. De fato, esta classificação no mundo real não é feita de forma simplesmente binária, como verdadeira ou falsa, mas é algo gradativo, no sentido de que o coração (geralmente) vai deixando de funcionar adequadamente, até eventualmente parar, ao invés de estar funcionando e de repente não estar mais. Uma incipiente contribuição deste trabalho neste sentido foi classificar tal conceito do coração funcionando como completo ou parcial, baseando-se na idéia de completude do ciclo em relação às suas formas de onda principais. Entretanto, dar um tratamento adequado a esta questão é de suma importância para aplicações que pretendem monitorar o funcionamento do coração.

6. CONCLUSÕES E CONSIDERAÇÕES FINAIS

6. CONCLUSÕES E CONSIDERAÇÕES FINAIS

Com aplicação dos resultados advindos da disciplina filosófica de Ontologia Formal, este trabalho apresenta um estudo de alternativas de mapeamentos de modelos descritos na linguagem de nível ontológico OntoUML para modelos descritos na linguagem de nível epistemológico OWL. Estes mapeamentos, aplicáveis segundo uma abordagem de Engenharia de Ontologias, permitem representar importantes características do domínio, como informação temporal.

Em resposta aos objetivos desta pesquisa, um primeiro resultado, apresentado no capítulo 3, foi a fundamentação ontológica de estratégias baseadas em teorias filosóficas sobre temporalidade para representação de informação temporal em OWL, a saber, as teorias Perdurantista 4D e de Reificação Temporal. Para cada teoria foram propostas estruturas-base ontologicamente fundamentadas com diretrizes para guiar decisões de modelagem. As propostas foram avaliadas em relação aos aspectos temporais que permitem capturar e, em seguida, comparadas entre si, e com trabalhos relacionados.

Outro resultado, apresentado no capítulo 4, foram as propostas de mapeamento sistemático de modelos OntoUML para OWL, justificados com base na UFO, ontologia que fundamenta a OntoUML. Esses mapeamentos permitem representar tanto cenários estáticos quanto dinâmicos, seguindo-se as propostas ontologicamente fundamentadas para a representação de informação temporal em OWL, apresentadas no capítulo 3. Esses mapeamentos foram implementados como complemento a uma Infra-estrutura OntoUML.

Para verificação dos resultados desta pesquisa, um estudo de caso no domínio de Eletrocardiografia foi realizado seguindo um processo de Engenharia de Ontologias, codificando-se automaticamente uma ontologia de referência de ECG na linguagem OWL, permitindo, então, representar a informação temporal. Verifica-se assim uma melhoria substancial nos resultados outrora alcançados, representando-se a informação sobre o funcionamento do coração não mais de forma estática, retratando um momento isolado, mas de forma dinâmica, retratando uma sequência de momentos, permitindo que esta informação seja utilizada para detecção de possíveis anomalias cardíacas a partir de registros de ECG.

Enfim, os resultados obtidos neste trabalho são relevantes no contexto de Engenharia de Ontologias por prover mapeamentos sistemáticos que: (i) uma vez implementados, permitem que a codificação da ontologia de referência seja realizada de forma automatizada, eficiente, que

preserva a qualidade do modelo de referência e evita erros decorrentes do processo de codificação manual; (ii) contornam uma limitação da linguagem alvo, OWL, permitindo representar a informação temporal.

6.1. LIMITAÇÕES E PERSPECTIVAS FUTURAS

Apesar de tratar da **representação de informação temporal**, este trabalho não aborda **representações do tempo** e suas relações, nem consultas para **recuperação da informação**. A primeira limitação reflete uma restrição imposta pela própria linguagem OWL, que permite a representação do tempo apenas como um tipo de dado. Há, porém, extensões de OWL que oferecem uma representação elaborada do tempo e que podem ser estudadas e aplicadas como extensão deste trabalho (HOBBS; PAN, 2004). A segunda limitação se deu pela restrição de tempo para incorporar ao trabalho alguns padrões de consultas já elaborados (ZAMBORLINI; GUIZZARDI, 2010b), mas também devido ao fato de alguns tipos de consultas dependerem de uma solução para a primeira limitação aqui descrita, uma vez que requerem verificar sobreposição temporal de indivíduos.

Com relação aos mapeamentos, outro ponto é que os estereótipos da linguagem OntoUML não foram totalmente contemplados, pois não foram consideradas as relações formais derivadas, como **mais-velho-que**. Entretanto, entende-se que uma solução para esta limitação depende da solução das duas limitações supracitadas, uma vez que tais relações não precisam ser instanciadas diretamente, mas podem ser tratadas como consultas a partir das informações existentes.

Também as alternativas híbridas das abordagens 4D e de Reificação Temporal foram abordadas superficialmente por uma questão de limitação de escopo. Uma perspectiva para trabalhos futuros seria aprofundá-las, verificando outras variações, sistematizando os mapeamentos a partir da linguagem OntoUML e implementá-las.

Outra limitação deste trabalho é quanto à avaliação da eficiência computacional das abordagens propostas. O que se tem são indícios teóricos de que a abordagem de Reificação Temporal é mais escalável que a Perdurantista 4D, e que as alternativas 4D tornam-se mais escaláveis à medida que se diminui a proliferação de partes temporais. Uma avaliação mais precisa deste aspecto é fundamental, pois muito se requer de eficiência computacional em aplicações *Web*. Contudo, este tópico é bastante complexo, e também por uma questão de limitação de escopo não pôde ser abordado neste trabalho.

Neste contexto, um estudo de continuidade interessante é avaliar a relevância do uso de determinados tipos de construtos OWL para o objetivo da aplicação, uma vez que a inserção/remoção de certos construtos, como a transitividade, podem causar um impacto considerável na eficiência computacional. Nesta linha, alguns trabalhos recentes estudam a relação entre construtos de DL e o aumento da complexidade dos modelos (ARTALE; CALVANESE; ET AL., 2010), e, a mesma relação para DL temporal (ARTALE; KONTCHAKOV; ET AL., 2010).

Outros ponto a se considerar é generalização das abordagens e dos mapeamentos para cobrir outras linguagens baseadas em DL, ou ainda com diferentes características, como hipótese de mundo fechado e de nomeação única (e.g., F-Logic).

Por fim, embora o uso adequado de um processo de Engenharia Ontologias permita amenizar o conflito entre expressividade e complexidade computacional, observa-se que todas essas questões a serem consideradas, como diferentes paradigmas, linguagens de codificação, abordagens de representação, ou ainda combinação de construtos da linguagem, torna a fase de projeto, ou seja, a escolha por uma alternativa de codificação, uma tarefa bastante complexa. De fato, um ponto em aberto na área de Engenharia de Ontologias é a sistematização do espaço de projeto (*design space*) (GUIZZARDI, 2010a), como acontece em Engenharia de Domínio (TEKINERDOGAN; AKSIT, 2001). Tal sistematização guiaria a escolha por uma alternativa de codificação levando em conta tanto características do modelo de referência, como tamanho e complexidade, quanto características da aplicação, como eficiência e escabilidade.

Neste contexto, uma contribuição deste trabalho é a avaliação e comparação das abordagens que dão indícios para escolha de uma alternativa apropriada. Assim, um trabalho futuro é elaborar na direção de sugerir automaticamente uma alternativa de mapeamento, baseando-se nas características do modelo e da aplicação.

REFERÊNCIAS

7. REFERÊNCIAS

ANTONIOU, G.; HARMELEN, F. V. Web Ontology Language: OWL. In: S. Staab R. Studer (Orgs.); **Handbook on Ontologies**, International Handbooks Information System. p.67-91. Springer-Verlag, 2003.

ARTALE, A.; CALVANESE, D.; IBÁÑEZ-GARCÍA, A. Full Satisfiability of UML Class Diagrams. In: J. Parsons; M. Saeki; P. Shoval; C. Woo; Y. Wand (Orgs.); **Conceptual Modeling – ER 2010**, Lecture Notes in Computer Science. v. 6412, p.317-331. Springer Berlin / Heidelberg. doi: 10.1007/978-3-642-16373-9_23, 2010.

ARTALE, A.; KONTCHAKOV, R.; RYZHIKOV, V.; ZAKHARYASCHEV, M. Complexity of Reasoning over Temporal Data Models. In: J. Parsons; M. Saeki; P. Shoval; C. Woo; Y. Wand (Orgs.); **Conceptual Modeling – ER 2010**, Lecture Notes in Computer Science. v. 6412, p.174-187. Springer Berlin / Heidelberg. doi: 10.1007/978-3-642-16373-9_13, 2010.

BRACHMAN, R. On the epistemological status of semantic networks. In: **N. V. Findler (Ed.), Associative Networks: Representation and Use of Knowledge by Computers**. Academic Press, 1979.

BRACHMAN, R.; LEVESQUE, H. **Knowledge Representation and Reasoning**. Morgan Kaufmann Publishers Inc, 2004.

BUNGE, M. **Ontology I: The Furniture of the World**, Treatise on Basic Philosophy. 1º ed., v. 3. New York: D. Reidel Publishing, 1977.

CARRARETTO, R. **A Modeling Infrastructure for OntoUML**. Graduation Thesis, Vitória, ES, Brazil: Universidade Federal do Espírito Santo, 2010, Julho.

DEGEN, W.; HELLER, B.; HERRE, H.; SMITH, B. GOL: toward an axiomatized upper-level ontology. In: Proceedings of the international conference on Formal Ontology in Information Systems - FOIS '01. **Anais...** . p.34-46. Ogunquit, Maine, USA. doi: 10.1145/505168.505173, 2001.

GALTON, A. Operators vs. Arguments: The Ins and Outs of Reification. **Synthese**, v. 150, p. 415-441, 2006.

GANGEMI, A. Ontology Design Patterns for Semantic Web Content. In: International Semantic Web Conference'05. **Anais...** . p.262–276. doi: 10.1007/11574620_21, 2005.

GESELOWITZ, D. B. On the theory of the electrocardiogram. **Proceedings of the IEEE**, v. 77, n. 6, p. 857–876. doi: 10.1109/5.29327, 1989.

GONÇALVES, B.; GUIZZARDI, G.; FILHO, J. G. P. Using an ECG reference ontology for semantic interoperability of ECG data. **Journal of Biomedical Informatics**, v. In Press, Corrected Proof. doi: 10.1016/j.jbi.2010.08.007, 2010.

GONÇALVES, B.; PEREIRA FILHO, J. G.; GUIZZARDI, G. An Electrocardiogram (ECG) Domain Ontology. In: Proceedings of the 2nd Workshop on Ontologies and Metamodels for Software and Data Engineering (WOMSDE'07). **Anais...** . João Pessoa, Brazil, 2007.

GONÇALVES, B.; ZAMBORLINI, V.; GUIZZARDI, G. Using a Lightweight Ontology of Heart Electrophysiology in an Interactive Web Application. In: XIV Simposio Brasileiro de Sistemas Multimidia e Web (WEBMIDIA 2008). **Anais...** . Vila Velha (ES), Brazil, 2008.

GONÇALVES, B.; ZAMBORLINI, V.; GUIZZARDI, G. An ontological analysis of the electrocardiogram. **ELECTRONIC JOURNAL OF COMMUNICATION, INFORMATION AND INNOVATION IN HEALTH**, Ontologies, Semantic Web and Health., v. 3, n. 1, p. 1-26. doi: 10.3395/reciis.v3i1.242en, 2009.

GONÇALVES, B.; ZAMBORLINI, V.; GUIZZARDI, G.; PEREIRA FILHO, J. G. An ontology-based application in heart electrophysiology: representation, reasoning and visualization on the web. In: Proceedings of the 2009 ACM symposium on Applied Computing. **Anais...** . p.816-820. Honolulu, Hawaii: ACM. doi: 10.1145/1529282.1529456, 2009.

GRAU, B. C.; HORROCKS, I.; MOTIK, B.; ET AL. OWL 2: The next step for OWL. **Web Semant.**, v. 6, n. 4, p. 309-322. doi: http://dx.doi.org/10.1016/j.websem.2008.05.001, 2008.

GUARINO, N. The Ontological Level. **PHILOSOPHY AND THE COGNITIVE SCIENCES**, p. 443--456, 1994.

GUARINO, N. The Ontological Level: Revisiting 30 Years of Knowledge Representation. **Conceptual Modeling: Foundations and Applications**, p. 52-67. doi: 10.1007/978-3-642-02463-4_4, 2009.

GUIZZARDI, G. **Ontological foundations for structural conceptual models**. PhD Thesis, Enschede: CTIT, Centre for Telematics and Information Technology, 2005.

GUIZZARDI, G. Agent Roles, Qua Individuals and the Counting Problem. In: A. F. Garcia; R. Choren; C. J. P. D. Lucena; et al. (Orgs.); Software Engineering for Multi-Agent Systems IV (SELMAS), Research Issues and Practical Applications. **Anais...** , Lecture Notes in Computer Science. v. 3914, p.143-160. Springer, 2006.

GUIZZARDI, G. On Ontology, ontologies, Conceptualizations, Modeling Languages and (Meta)Models. In: Databases and Information Systems IV - Selected Papers from the Seventh International Baltic Conference DB&IS'2006. **Anais...** , Frontiers in Artificial Intelligence and Applications. v. 155, p.18-39. Vilnius, Lithuania: Olegas Vasilecas, Johann Eder, Albertas Caplinskas, 2007.

GUIZZARDI, G. The Problem of Transitivity of Part-Whole Relations in Conceptual Modeling Revisited. In: Proceedings of the 21st International Conference on Advanced Information Systems Engineering. **Anais...** . p.94-109. Amsterdam, The Netherlands: Springer-Verlag, 2009.

GUIZZARDI, G. Theoretical Foundations and Engineering Tools for Building Ontologies as Reference Conceptual Models. **Semantic Web**, v. 1, n. 1-2, p. 3-10. doi: 10.3233/SW-2010-0015, 2010a.

GUIZZARDI, G. Representing Collectives and Their Members in UML Conceptual Models: An Ontological Analysis. In: J. Trujillo; G. Dobbie; H. Kangassalo; et al. (Orgs.); Proceedings of 6th International Workshop on Foundations and Practice of UML (FP-UML). **Anais...** , Lecture Notes in Computer Science. v. 6413, p.265-274. Vancouver, Canada: Springer, 2010b.

GUIZZARDI, G. On the Representation of Quantities and their Parts in Conceptual Modeling. In: Proceeding of the 2010 conference on Formal Ontology in Information Systems: Proceedings of the Sixth International Conference (FOIS 2010). **Anais...** . p.103–116. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2010c.

GUIZZARDI, G.; HALPIN, T. Ontological foundations for conceptual modelling. **Appl. Ontol.**, v. 3, p. 1–12, 2008.

GUIZZARDI, G.; MASOLO, C.; BORGO, S. In Defense of a Trope-Based Ontology for Conceptual Modeling: An Example with the Foundations of Attributes, Weak Entities and Datatypes. In: D. Embley; A. Olivé; S. Ram (Orgs.); Conceptual Modeling - ER 2006. **Anais...** , Lecture Notes in Computer Science. v. 4215, p.112-125. doi: 10.1007/11901181_10, 2006.

GUIZZARDI, G.; WAGNER, G. Some Applications of a Unified Foundational Ontology in Business Modeling. In: M. Rosemann P. Green (Orgs.); . p.345-367. IDEA Publisher, 2005.

GUIZZARDI, G.; WAGNER, G. What's in a Relationship: An Ontological Analysis. In: Proceedings of the 27th International Conference on Conceptual Modeling. **Anais...** , ER '08. p.83–97. Berlin, Heidelberg: Springer-Verlag. doi: 10.1007/978-3-540-87877-3_8, 2008.

GUIZZARDI, G.; WAGNER, G. Using the Unified Foundational Ontology (UFO) as a Foundation for General Conceptual Modeling Languages. In: R. Poli; M. Healy; A. Kameas (Orgs.); **Theory and Applications of Ontology: Computer Applications**. p.175-196. Germany: Springer Netherlands. doi: 10.1007/978-90-481-8847-5_8, 2010.

GUIZZARDI, G.; WAGNER, G.; GUARINO, N.; SINDEREN, M. V. An Ontologically Well-Founded Profile for UML Conceptual Models. In: A. Persson J. Stirna (Orgs.); CAiSE. **Anais...** , Lecture Notes in Computer Science. v. 3084, p.112-126. Latvia: Springer, 2004.

GUYTON, A. C.; HALL, J. E. Textbook of Medical Physiology. In: . 9 ed., p.803-813. Philadelphia: University of Mississippi, 1996.

HOBBS, J. R.; PAN, F. An ontology of time for the semantic web. **ACM Transactions on Asian Language Information Processing (TALIP)**, v. 3, n. 1, p. 66-85. doi: 10.1145/1017068.1017073, 2004.

HOEKSTRA, R. **Ontology Representation: Design Patterns and Ontologies that Make Sense - Volume 197 Frontiers in Artificial Intelligence and Applications**. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009.

HORROCKS, I.; KUTZ, O.; SATTLER, U. The Even More Irresistible SROIQ. In: Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR2006). **Anais...** . p.57–67. AAAI Press, 2006.

HORROCKS, I.; PATEL-SCHNEIDER, P. F.; HARMELEN, F. V. From SHIQ and RDF to OWL: the making of a Web Ontology Language. **Web Semantics: Science, Services and Agents on the World Wide Web**, v. 1, n. 1, p. 7 - 26. doi: DOI: 10.1016/j.websem.2003.07.001, 2003.

KRIEGER, H. Where Temporal Description Logics Fail: Representing Temporally-Changing Relationships. In: KI 2008: Advances in Artificial Intelligence, 31st Annual German Conference on AI. **Anais...** , Lecture Notes in Computer Science. v. 5243, p.249-257.

Kaiserslautern, Germany,: Springer. doi: 10.1007/978-3-540-85845-4_31, 2008.

KRIPKE, S. **Naming and Necessity**. Harvard University Press, 1982.

LUTZ, C.; WOLTER, F.; ZAKHARYASCHEV, M. Temporal Description Logics: A Survey. In: Proceedings of the 2008 15th International Symposium on Temporal Representation and Reasoning. **Anais...** . p.3-14. IEEE Computer Society, 2008.

MASOLO, C.; BORGO, S.; GANGEMI, A.; GUARINO, N.; OLTRAMARI, A. Ontology Library. **Wonderweb deliverable D18**, v. 33052, 2003.

MASOLO, C.; GUIZZARDI, G.; VIEU, L.; BOTTAZZI, E.; FERRARIO, R. Relational roles and qua-individuals. In: Procs. of AAAI Fall Symposium Roles. **Anais...** . v. 5. Virginia, USA, 2005.

MEALY, G. H. Another look at data. In: Proceedings of the November 14-16, 1967, fall joint computer conference. **Anais...** , AFIPS '67 (Fall). p.525–534. New York, NY, USA: ACM. doi: http://doi.acm.org/10.1145/1465611.1465682, 1967.

MOTIK, B.; PATEL-SCHNEIDER, P. F.; BECHHOFER, S.; ET AL. **OWL 2 Web Ontology Language: XML Serialization**. W3C. Recuperado Junho 15, 2010, de http://www.w3.org/TR/2009/REC-owl2-xml-serialization-20091027/, 2009.

MOTIK, B.; PATEL-SCHNEIDER, P. F.; GRAU, B. C. **OWL 2 Web Ontology Language: Direct Semantics**. W3C. Recuperado Junho 15, 2010, de http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/, 2009.

MOTIK, B.; PATEL-SCHNEIDER, P. F.; PARSIA, B.; ET AL. **OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax**. W3C. Recuperado Junho 15, 2010, de http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/, 2009.

O'CONNOR, M.; DAS, A. A Method for Representing and Querying Temporal Information in OWL. In: Biomedical Engineering Systems and Technologies (Selected Papers). **Anais...** , Communications in Computer and Information Science. Springer, 2011.

QUINE, W. V. Events and reification. In: **Actions and Events: Perspectives on the Philosophy of Davidson**. p.162-71. Blackwell, 1985.

Reifying. .**Merriam-Webster Online Dictionary**. Recuperado Outubro 10, 2010, de http://www.merriam-webster.com/dictionary/reifying, 2010.

SMITH, B.; WELTY, C. Ontology: towards a new synthesis. In: Proceedings of the international conference on Formal Ontology in Information Systems - FOIS '01. **Anais...** . p.3-9. Ogunquit, Maine, USA. doi: 10.1145/505168.505201, 2001.

TEKINERDOGAN, B.; AKSIT, M. Synthesis - Based Software Architecture Design. In: M. Askit (Org.); **Software Architecture and Component Technology**. p.143–174. Dordrecht: Kluwer Academic Publishers, 2001.

VARZI, A. C. Naming the stages. **Dialectica**, v. 57, n. 4, p. 387–412, 2003.

VIEU, L.; BORGO, S.; MASOLO, C. Artefacts and Roles: Modelling Strategies in a Multiplicative Ontology. In: Proceeding of the 2008 conference on Formal Ontology in Information Systems: Proceedings of the Fifth International Conference (FOIS 2008).

Anais... . p.121-134. IOS Press, 2008.

WAND, Y.; WEBER, R. On the deep structure of information systems. **Information Systems Journal**, v. 5, n. 3, p. 203–223. doi: 10.1111/j.1365-2575.1995.tb00108.x, 1995.

WELTY, C.; FIKES, R. A Reusable Ontology for Fluents in OWL. In: Proceeding of the 2006 conference on Formal Ontology in Information Systems: Proceedings of the Fourth International Conference (FOIS 2006). **Anais...** . p.226-236. IOS Press, 2006.

WELTY, C.; GUARINO, N. Supporting ontological analysis of taxonomic relationships. **Data Knowl. Eng.**, v. 39, n. 1, p. 51–74. doi: 10.1016/S0169-023X(01)00030-1, 2001.

ZAMBORLINI, V. **Implementação de uma Ontologia de Referência de Eletrocardiograma com Análise da Perda de Expressividade**. Projeto Final de Graduação, Vitória, ES, Brazil: Universidade Federal do Espírito Santo, 2008, Fevereiro.

ZAMBORLINI, V.; GONÇALVES, B.; GUIZZARDI, G. Codification and Application of a Well-Founded Heart-ECG Ontology,. In: 3rd Workshop on Ontologies and Metamodels in Software and Data Engineering. In: Proceedings of the 3rd Workshop on Ontologies and Metamodels for Software and Data Engineering (WOMSDE'08). **Anais...** . Campinas, Brazil, 2008.

ZAMBORLINI, V.; GUIZZARDI, G. On the representation of temporally changing information in OWL. In: EDOC Workshops - VORTE'10. **Anais...** . p.6. Vitória, ES, Brazil: IEEE Computer Society, 2010a.

ZAMBORLINI, V.; GUIZZARDI, G. **An Ontologically-founded Reification Approach for Representing Temporally Changing Information in OWL**. p.11. NEMO - Ontology and Conceptual Modeling Research Group, 2010b.

APÊNDICES

APÊNDICE I – Código Java da Implementação dos Mapeamentos

```
package br.ufes.inf.nemo.ontouml.transformation.ontouml2owl;
import java.io.File;
import org.eclipse.emf.common.util.URI;
import org.eclipse.emf.ecore.EObject;
import org.eclipse.emf.ecore.resource.Resource;
import org.eclipse.emf.ecore.resource.ResourceSet;
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;
import RefOntoUML.Package;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.auxiliary.MappingType;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.auxiliary.OWLStructure;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree.TreeProcessor;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.verbose.FileManager;

public class OntoUML2OWL
{ static FileManager myfile;
  private static OWLStructure owl;

  public static void main(String[] args)
  { String fileAbsolutePath = args[0];
    String modelName = args[0].replace(".refontouml", ".owl");
    // Configure ResourceSet
    ResourceSet resourceSet = new ResourceSetImpl();
    resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put(
                          Resource.Factory.Registry.DEFAULT_EXTENSION, new XMIResourceFactoryImpl());
    resourceSet.getPackageRegistry().put(
                         RefOntoUML.RefOntoUMLPackage.eNS_URI,RefOntoUML.RefOntoUMLPackage.eINSTANCE);
    // Open the model
    File sourceFile = new File(fileAbsolutePath);
    if (!sourceFile.isFile())
    { System.out.println("Error accessing: " + sourceFile.getAbsolutePath());
      return; }
    URI uri = URI.createFileURI(sourceFile.getAbsolutePath());
    try
    { // Read the objects in the model
      Resource resource = resourceSet.getResource(uri, true);
      EObject p1 = resource.getContents().get(0);
      Package p = (Package) p1;

      // Processing the OntoUML model as a structure containing a
      // specialization tree for the Classes and a list of Binary Associations
      TreeProcessor tp = new TreeProcessor(p);

      // mapping the OntoUML-based structure into an OWL-based structure
      // according to a certain mapping type
      map2OWLStructure(tp, MappingType.WORM_VIEW_A1);

      // Writing transformed model into owl file
      myfile = new FileManager(modelName);
      myfile.write(owl.verbose(modelName));
      myfile.done();
    }
    catch (Exception e) { e.printStackTrace(); }
  }

  public static void map2OWLStructure (TreeProcessor tp, MappingType mt)
  { owl = new OWLStructure(mt);
```

```
    owl.map(tp); }
}

////////////////////////////////////////////////////////////////////////////////

package br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import RefOntoUML.*;
import RefOntoUML.Class;
import RefOntoUML.Package;

public class TreeProcessor
{ public LinkedList<NodeClass> nodes;
  public List<NodeBinAssociation> assocNodes;
  static HashMap<Class, NodeClass> class2node;
  public static List<String> kindsNames = null;
  public static List<String> collectivesNames = null;
  public static List<String> quantitiesNames = null;
  public static List<String> relatorsNames = null;
  public static List<String> modesNames = null;
  public static List<String> qualitiesNames = null;
  public static List<String> relationalquasNames = null;
  public static List<String> phasedquasNames = null;

  public TreeProcessor (Package p)
  { nodes = new LinkedList<NodeClass>();
    class2node = new HashMap<Class, NodeClass>();
    assocNodes = new LinkedList<NodeBinAssociation>();
    // Pre Process
    for (PackageableElement pe : p.getPackagedElement())
    { if (pe instanceof Class) ProcessClass((Class) pe);
      if (pe instanceof Association) ProcessAssociation((Association) pe); }
    // Set up the specialization tree
    ProcessNodes();
  }

  private void ProcessClass (Class c)
  { NodeClass n = new NodeClass(c);
    nodes.add(n);
    class2node.put(c, n);
    if (c instanceof Kind)
    { if (kindsNames == null) kindsNames = new LinkedList<String>();
      kindsNames.add(c.getName()); }
    else if (c instanceof Collective)
    { if (collectivesNames == null collectivesNames = new LinkedList<String>();
      collectivesNames.add(c.getName()); }
    else if (c instanceof Quantity)
    { if (quantitiesNames == null) quantitiesNames = new LinkedList<String>();
      quantitiesNames.add(c.getName()); }
    else if (c instanceof Relator)
    { if (relatorsNames == null) relatorsNames = new LinkedList<String>();
      relatorsNames.add(c.getName()); }
    else if (c instanceof Mode)
    { if (modesNames == null) modesNames = new LinkedList<String>();
      modesNames.add(c.getName()); }
  }

  private void ProcessAssociation(Association a)
  { NodeBinAssociation na = new NodeBinAssociation(a);
```

```
    assocNodes.add(na);
    String domain = na.getDomain().getName(), range = na.getRange().getName();
    if (a instanceof MaterialAssociation)
    { na.mappingName = a.getName() + domain + range;
      na.addSuperAssociation(a.getName()); }
    else if (a instanceof FormalAssociation)
    { na.mappingName = a.getName() + domain + range;
      na.addSuperAssociation(a.getName()); }
    else if (a instanceof Meronymic)
    { if (a instanceof componentOf) na.mappingName = "componentOf";
      else if (a instanceof subCollectionOf) na.mappingName = "subCollectionOf";
      else if (a instanceof subQuantityOf) na.mappingName = "subQuantityOf";
      else if (a instanceof memberOf) na.mappingName = "memberOf";
      //adding superProperty
      na.addSuperAssociation(na.mappingName);
      //renaming association
      na.mappingName += domain + range;
    }
    else if (a instanceof DependencyRelationship)
    { //na.mappingName = null => the property is not created
      if (a instanceof Mediation) na.addSuperAssociation("mediates"); //to be restricted
      else if (a instanceof Characterization) na.addSuperAssociation("inheresIn"); //to be restricted
    }
  }

  private void ProcessNodes ()
  { // For every node, add children and child partitions
    for (NodeClass n : nodes)
    { // Get the related class
      Class c = n.getRelatedClass();
      // Get the generalizations of the class
      for (Generalization g : c.getGeneralization())
      { // Get the parent
        Classifier parent = g.getGeneral();
        // Get the parent's node
        NodeClass parentNode = class2node.get(parent);
        // Add the node as a child of the parent
        parentNode.addChild(n);
        if (g.getGeneralizationSet().size() != 0) // Has partition
        { GeneralizationSet gs = g.getGeneralizationSet().get(0);
          // Add the gs as a child partition of the parent
          parentNode.addChildPartition(gs, n); }
        else // Does not have partition
        { // Add the node as a Solitary Child of the parent
           parentNode.addSChild(n); }
      }
      if (n.isUltimateSortal())
      { if (c instanceof Kind) n.setDisjointSiblingsNames(kindsNames);
        else if (c instanceof Quantity) n.setDisjointSiblingsNames(quantitiesNames);
        else if (c instanceof Collective) n.setDisjointSiblingsNames(collectivesNames); }
    }
  }

  public List<NodeClass> getNodes()
  { return nodes; }

  public static NodeClass getNode (Class c)
  { return class2node.get(c); }
}
////////////////////////////////////////////////////////////////////////////////
package br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree;
```

```
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import org.eclipse.emf.common.util.EList;
import RefOntoUML.*;
import RefOntoUML.Class;

public class NodeClass
{ RefOntoUML.Class myclass;
  LinkedList<NodeClass> children; // All children
  LinkedList<NodeClass> schildren; // Children without partition
  LinkedList<NodeClass> pchildren; // Children with partition
  List<String> siblings = null;
  LinkedList<Association> associations; // Has the node as member end
  LinkedList<Association> ownedAssociations; // Has the node as first member end
  LinkedList<ChildPartition> childPartitions;
  HashMap<GeneralizationSet, ChildPartition> gs2partition;

  public NodeClass (Class c)
  { myclass = c;
    children = new LinkedList<NodeClass>();
    schildren = new LinkedList<NodeClass>();
    pchildren = new LinkedList<NodeClass>();
    childPartitions = new LinkedList<ChildPartition>();
    associations = new LinkedList<Association>();
    ownedAssociations = new LinkedList<Association>();
    gs2partition = new HashMap<GeneralizationSet, ChildPartition>();
  }

  public String getName()
  { return myclass.getName(); }

  public String getName(Boolean addTS)
  { return myclass.getName() + (addTS ? "TS" : ""); }

  public String getTSName()
  { return getName(true); }

  public String getReifiedName()
  { if (isRigid()) return myclass.getName();
    else return "Qua" + myclass.getName(); }

  public Boolean isRigid()
  { return ((myclass instanceof RigidMixinClass)
        || (myclass instanceof RigidSortalClass) || (myclass instanceof MomentClass)); }

  public Boolean isUltimateSortal()
  { return (myclass instanceof SubstanceSortal); }

  public Boolean isMoment()
  { return (myclass instanceof MomentClass); }

  public Class getRelatedClass()
  { return myclass; }

  public void addChild (NodeClass child)
  { children.add(child); }

  public void addSChild (NodeClass child)
  { schildren.add(child); }
```

```
public void addChildPartition (GeneralizationSet gs, NodeClass child)
{ ChildPartition cp = gs2partition.get(gs);
  if (cp == null)
  { cp = new ChildPartition(gs);
    gs2partition.put(gs, cp);
    childPartitions.add(cp); }
  cp.addChild(child);
  pchildren.add(child);
}

public List<ChildPartition> getChildPartitions()
{ return childPartitions; }

public List<NodeClass> getChildren()
{ return children; }

public List<String> getDisjointSiblingsNames(Boolean addTS, Boolean reification)
{ if (isUltimateSortal())
  { if (!addTS) return siblings;
    else
    { List<String> siblingsTS = new LinkedList<String>();
      for (String s : siblings) siblingsTS.add(s + "TS");
      return siblingsTS; }
  }
  List<String> disjointClasses = new LinkedList<String>();
  // For every Generalization of the Class
  for (Generalization g : myclass.getGeneralization())
  { EList<GeneralizationSet> gsets = g.getGeneralizationSet();
    // If the Generalization has a Generalization Set
    if (gsets.size() > 0)
    { GeneralizationSet gs = gsets.get(0);
      // If the Generalization Set is Disjoint
      if (gs.isIsDisjoint())
      { // For every generalization of the brothers)
        for (Generalization bg : gs.getGeneralization())
          if (bg != g)
          { NodeClass n = TreeProcessor.getNode((Class) bg.getSpecific());
            if (reification) disjointClasses.add(n.getReifiedName());
            else disjointClasses.add(n.getName(addTS)); }
      }
    }
  }
  if (disjointClasses.size() > 0) return disjointClasses;
  else return null;
}

public void setDisjointSiblingsNames(List<String> lsiblings)
{ if (lsiblings == null) return;
  if (siblings == null) siblings = new LinkedList<String>();
  String name = getName();
  for (String s : lsiblings) if (!name.equals(s)) siblings.add(s); }

public String getStructuralParent()
{ Class c = myclass;
  if (c instanceof Kind) return "FunctionalComplex";
  else if (c instanceof Collective) return "Collective";
  else if (c instanceof Quantity) return "Quantity";
  else if (c instanceof Relator) return "Relator";
  else if (c instanceof Mode) return "Mode";
  else if (c instanceof Role) return "RelationalQuaIndividual";
  else if (c instanceof Phase) return "PhasedQuaIndividual";
```

```
  else return null; }

public List<String> getParentsNames(Boolean addTS)
{ if (isUltimateSortal())
  { List<String> lParents = new LinkedList<String>();
    lParents.add(getStructuralParent() + (addTS ? "TS" : "") );
    return lParents; }
  return getParentsNames(false, addTS); }

public List<String> getRigidParentsNames(Boolean addTS)
{ return getParentsNames(true, addTS); }

private List<String> getParentsNames(Boolean justRigid, Boolean addTS)
{ List<String> lStParents = null;
  if (isMoment())
  { lStParents = new LinkedList<String>();
    lStParents.add(getStructuralParent() + (addTS ? "TS" : "")); }
  List<String> lparents = null;
  if (myclass.parents() != null)
  { lparents = new LinkedList<String>();
    List<Classifier> classifierList = new LinkedList<Classifier>();
    List<Classifier> auxList = new LinkedList<Classifier>();
    // recursively looking for non-reified parent(s)
    classifierList.addAll(myclass.parents());
    while ((lparents.isEmpty()) && !classifierList.isEmpty())
    { for (Classifier cp : classifierList)
      { NodeClass n = TreeProcessor.getNode((Class) cp);
        if (!justRigid || n.isRigid()) lparents.add(n.getName(addTS));
        auxList.addAll(cp.parents()); }
      //if a suitable parent is not found, it is to be looked for among the parents of parents
      if (lparents.isEmpty())
      { classifierList.clear();
        classifierList.addAll(auxList);
        auxList.clear(); }
    }
  }

  if ((lStParents != null) && !lStParents.isEmpty())
  { if (lparents != null) lStParents.addAll(lparents);
    return lStParents; }
  else if ((lparents != null) && !lparents.isEmpty()) return lparents;
  else return null;
}

public List<String> getReifiedParentsNames()
{ List<String> lparents = null;
  Class c = myclass;
  if (c.parents() != null)
  { lparents = new LinkedList<String>();
    List<Classifier> classifierList = new LinkedList<Classifier>();
    List<Classifier> auxList = new LinkedList<Classifier>();
    // recursively looking for reified parent(s)
    classifierList.addAll(c.parents());
    while ((lparents.isEmpty()) && !classifierList.isEmpty())
    { for (Classifier cp : classifierList)
      { NodeClass n = TreeProcessor.getNode((Class) cp);
        if (!n.isRigid()) lparents.add(n.getReifiedName());
        auxList.addAll(cp.parents()); }
      //if a suitalbe parent is not found, it is to be looked for among the parents of parents
      if (lparents.isEmpty())
      { classifierList.clear();
```

```
            classifierList.addAll(auxList);
            auxList.clear(); }
        }
      }
      if (lparents.size() > 0) return lparents;
      else return null;
    }

    public List<List<String>> getCompleteChildren (Boolean addTS, Boolean is4DView, Boolean isReifView)
    { Boolean complete = true;
      //Mixin Classes are always defined as the conjunction of its children
      if (myclass instanceof MixinClass)
      { complete = false;
        List<List<String>> allChildren = getChildrenByPartition(complete, addTS, is4DView, isReifView);
        List<List<String>> result = null;
        if (allChildren != null)
        { List<String> children = new LinkedList<String>();
          for (List<String> ls : allChildren) children.addAll(ls);
          result = new LinkedList<List<String>>();
          result.add(children); }
        return result;
      }
      else return getChildrenByPartition(complete, addTS, is4DView, isReifView);
    }

    private List<List<String>> getChildrenByPartition (Boolean complete, Boolean addTS, Boolean is4DView,
                                                       Boolean isReifView)
    { List<List<String>> childrenByPartition = null;
      // For each Generalization Set
      for (ChildPartition cp : getChildPartitions())
      { if (cp.getGS().isIsCovering() || !complete)
        { if (childrenByPartition == null) childrenByPartition = new LinkedList<List<String>>();
          List<String> names = new LinkedList<String>();
          // Children in the Generalization Set
          for (NodeClass child : cp.getChildren())
          { if ((is4DView && !addTS && !child.isRigid()) || (isReifView && !child.isRigid()))
            { //if it is required the complete partition but one does not fit
              //the it return none of them
              if (complete) names.clear();
              break; }
            names.add(child.getName(addTS)); }
          if (!names.isEmpty()) childrenByPartition.add(names);
        }
      }
      if ((childrenByPartition == null) || (childrenByPartition.size() == 0)) return null;
      else  return childrenByPartition;
    }
}

////////////////////////////////////////////////////////////////////////////////
package br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree;
import java.util.LinkedList;
import java.util.List;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.auxiliary.MappingType;
import RefOntoUML.Association;
import RefOntoUML.Characterization;
import RefOntoUML.Class;
import RefOntoUML.MaterialAssociation;
import RefOntoUML.Mediation;
import RefOntoUML.Meronymic;
import RefOntoUML.Property;
```

```
public class NodeBinAssociation {
  public Association myAssociation;
  Boolean existDep = false, invExistDep = false;
  public String mappingName = null;
  public List<String> listSuperAssociations = null;
  public List<Restriction> listRestrictions = null;
  Property sourceEnd, targetEnd;

  public NodeBinAssociation (Association a)
  { myAssociation = a;
    if (a instanceof Meronymic)
    { //originally the part is the range and whole is the domain exchanging domain and range
      sourceEnd = myAssociation.getMemberEnd().get(1);
      targetEnd = myAssociation.getMemberEnd().get(0); }
    else
    { sourceEnd = myAssociation.getMemberEnd().get(0);
      targetEnd = myAssociation.getMemberEnd().get(1); }
    Boolean especDep = ((targetEnd.lowerBound() >= 1) && (targetEnd.isIsReadOnly()));
    Boolean invEspecDep = ((sourceEnd.lowerBound() >= 1) && (sourceEnd.isIsReadOnly()));
    existDep = (especDep && hasRigidDomain());
    invExistDep = (invEspecDep && hasRigidRange());
  }

  public NodeClass getDomain()
  { return TreeProcessor.getNode((Class) sourceEnd.getType()); }

  public NodeClass getRange()
  { return TreeProcessor.getNode((Class) targetEnd.getType()); }

  private Boolean hasRigidDomain()
  { return getDomain().isRigid(); }

  private Boolean hasRigidRange()
  { return getRange().isRigid(); }

  public void addSuperAssociation(String superAssociation)
  { if (superAssociation == null) return;
    if (listSuperAssociations == null)  listSuperAssociations = new LinkedList<String>();
    listSuperAssociations.add(superAssociation); }

  //if it implies existential dependence
  public Boolean isOntFormalRelation()
  { return (existDep || invExistDep); }

  public void setRestrictions(MappingType mtype)
  { Integer tmin = targetEnd.lowerBound(), tmax = targetEnd.upperBound(),
            smin = sourceEnd.lowerBound(), smax = sourceEnd.upperBound();
    NodeClass ndomain = getDomain(),  nrange = getRange();
    String restrictedProperty = mappingName, restrictedDomain, restrictedRange;
    List<String> restrictedList = new LinkedList<String>();
    Boolean addTS = false, inverseProp, equivRestr;

    if (myAssociation instanceof Characterization) restrictedProperty = "inheresIn";
    else if (myAssociation instanceof Mediation) restrictedProperty = "mediates";

    if ((mtype == MappingType.REIFICATION) ||
       ((mtype == MappingType.WORM_VIEW_A2) && (existDep || invExistDep) ))
    { if (!targetEnd.isIsReadOnly()) tmax = -1;
      if (!sourceEnd.isIsReadOnly()) smax = -1; }
```

```
addTS = ((mtype == MappingType.WORM_VIEW_A0) ||
      ((mtype == MappingType.WORM_VIEW_A1) && !(existDep && invExistDep) ) ||

// if both classes are rigid or
// if it is neither the case of an ic-level property of 4D-A2 nor reification
// then the restriction are imposed over the original domain and range
if ( (ndomain.isRigid() && nrange.isRigid()) ||
   !(((mtype == MappingType.WORM_VIEW_A2) && (existDep || invExistDep)) ||
     (mtype == MappingType.REIFICATION) ) )
{ //adding cardinality restriction over the domain class
  if ((tmin > 0) || (tmax > 0))
  { restrictedDomain = ndomain.getName(addTS);
    restrictedRange = nrange.getName(addTS);
    inverseProp = false;
    equivRestr = false;
    addRestriction(restrictedDomain, restrictedProperty, restrictedRange, equivRestr, inverseProp,
                                                                                    tmin, tmax);
    if (myAssociation instanceof Meronymic)
      if (!((Meronymic) myAssociation).isIsShareable())
        addRestriction(restrictedDomain, listSuperAssociations.get(0), restrictedRange, equivRestr,
                                                                        inverseProp, 0, 1);
  }

  //adding cardinality restriction over the range class
  if ((smin > 0) || (smax > 0))
  { restrictedDomain = nrange.getName(addTS);
    restrictedRange = ndomain.getName(addTS);
    inverseProp = true;
    equivRestr = false;
    addRestriction(restrictedDomain, restrictedProperty, restrictedRange, equivRestr, inverseProp,
                                                                          smin, smax); }
}
//otherwise, some of them, domain or range, must be rigid
//since it is either the ic-level of 4D-A2 view, and thus
//the existential dependence holds for some of them
//or the reification view, and thus the property must imply existential dependence
else //( REIFICATION) || (WORM_VIEW_A2))
{ if (!ndomain.isRigid() && (nrange.isRigid()))
  { restrictedList = ndomain.getRigidParentsNames(addTS);
    if ((smin > 0) || (smax > 0))
    { restrictedDomain = nrange.getName(addTS);
      inverseProp = true;
      equivRestr = false;
      addRestriction(restrictedDomain, restrictedProperty, restrictedList, equivRestr, inverseProp,
                                                                          smin, smax); }
  }
  else if (!nrange.isRigid() && (ndomain.isRigid()))
  { restrictedList = nrange.getRigidParentsNames(addTS);
    if ((tmin > 0) || (tmax > 0))
    { restrictedDomain = ndomain.getName(addTS);
      inverseProp = false;
      equivRestr = false;
      addRestriction(restrictedDomain, restrictedProperty, restrictedList, equivRestr, inverseProp,
                                                                          tmin, tmax); }
  }
  if (mtype == MappingType.REIFICATION)
  { tmax = targetEnd.upperBound(),
    smax = sourceEnd.upperBound();
    if (myAssociation instanceof MaterialAssociation)
      if (!nrange.isRigid() && !ndomain.isRigid())
      { restrictedProperty = "existentiallyDependentOfQua";
```

```
          inverseProp = false;
          equivRestr = false;
          if ((smin > 0) || (smax > 0))
          { restrictedDomain = nrange.getReifiedName();
            restrictedRange = ndomain.getReifiedName();
            addRestriction(restrictedDomain, restrictedProperty, restrictedRange, equivRestr,
                                                                    inverseProp, smin, smax); }
          if ((tmin > 0) || (tmax > 0))
          { restrictedDomain = ndomain.getReifiedName();
            restrictedRange = nrange.getReifiedName();
            addRestriction(restrictedDomain, restrictedProperty, restrictedRange, equivRestr,
                                                                    inverseProp, tmin, tmax); }
        }
      if (myAssociation instanceof Mediation) //always implies exist. dep.
        if (!nrange.isRigid())
        { restrictedProperty = "partOfRelator";
          restrictedDomain = nrange.getReifiedName();
          restrictedRange = ndomain.getName();
          inverseProp = false;
          equivRestr = false;
          addRestriction(restrictedDomain, restrictedProperty, restrictedRange, equivRestr,
                                                                        inverseProp, 1, 0);
          if ((tmin > 0) || (tmax > 0))
          { restrictedDomain = ndomain.getName(addTS);
            restrictedRange = nrange.getReifiedName();
            inverseProp = true;
            addRestriction(restrictedDomain, restrictedProperty, restrictedRange, equivRestr,
                                                                    inverseProp, tmin, tmax); }
        }
    }
  }
}

public void addRestriction(String restrictedClass, String propertyName, String propertyClass,
                                    Boolean equivalent, Boolean inverse, Integer min, Integer max)
{ if (listRestrictions == null) listRestrictions = new LinkedList<Restriction>();
  List<String> listClasses = new LinkedList<String>();
  listClasses.add(propertyClass);
  Restriction r = new Restriction(restrictedClass, propertyName,
                                                        listClasses, equivalent, inverse, min, max);
  listRestrictions.add(r);
}

public void addRestriction(String restrictedClass, String propertyName, List<String> listClasses,
                                    Boolean equivalent, Boolean inverse, Integer min, Integer max)
{ if (listRestrictions == null) listRestrictions = new LinkedList<Restriction>();
  Restriction r = new Restriction(restrictedClass, propertyName,
            listClasses, equivalent, inverse, min, max);
  listRestrictions.add(r);
}

public class Restriction {
  public String restrictedClass, propertyName;
  public List<String> listClasses;
  public Boolean equivalent = false;
  public Boolean inverse = false;
  public Integer min, max;

  public Restriction (String restrictedClass, String propertyName,  List<String> listClasses,
                                    Boolean equivalent, Boolean inverse, Integer min, Integer max)
  { this.restrictedClass = restrictedClass;
```

```
      this.propertyName = propertyName;
      this.listClasses = listClasses;
      this.equivalent = equivalent;
      this.inverse = inverse;
      this.min = min;
      this.max = max; }
  }

  public List<String> getDomainList(MappingType mtype)
  { List<String> domainList = new LinkedList<String>();
    NodeClass domain = getDomain();
    if (mtype == MappingType.REIFICATION) domainList.add(domain.getReifiedName());
    else
    { Boolean addTS = ((mtype == MappingType.WORM_VIEW_A0) ||
          ((mtype == MappingType.WORM_VIEW_A1) && !(existDep && invExistDep) ) ||
          ((mtype == MappingType.WORM_VIEW_A2) && !(existDep || invExistDep) ) );
      domainList.add(domain.getName(addTS)); }
    if (!domainList.isEmpty()) return domainList;
    else return null;
  }

  public List<String> getRangeList(MappingType mtype)
  { List<String> rangeList = new LinkedList<String>();
    NodeClass range = getRange();
    if (mtype == MappingType.REIFICATION)
      rangeList.add(range.getReifiedName());
    else
    { Boolean addTS = ((mtype == MappingType.WORM_VIEW_A0) ||
          ((mtype == MappingType.WORM_VIEW_A1) && !(existDep && invExistDep) ) ||
          ((mtype == MappingType.WORM_VIEW_A2) && !(existDep || invExistDep) ) );
      rangeList.add(range.getName(addTS)); }
    if (!rangeList.isEmpty()) return rangeList;
    else return null;
  }

  public List<String> getSuperAssocList(MappingType mtype)
  { if (existDep) addSuperAssociation("existentiallyDependentOf");
    if (invExistDep) addSuperAssociation("invExistentiallyDependentOf");
    if ((existDep && invExistDep && (mtype == MappingType.WORM_VIEW_A1)) ||
        ((existDep || invExistDep) && (mtype == MappingType.WORM_VIEW_A2)))
      addSuperAssociation("objPropertyIC");
    else if ((mtype != null) && (mtype != MappingType.REIFICATION))
      addSuperAssociation("objPropertyTS");
    return listSuperAssociations;
  }
}

////////////////////////////////////////////////////////////////////////////////
package br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.auxiliary;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import RefOntoUML.MaterialAssociation;
import RefOntoUML.Property;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree.NodeBinAssociation;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree.NodeClass;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree.TreeProcessor;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.tree.NodeBinAssociation.Restriction;
import br.ufes.inf.nemo.ontouml.transformation.ontouml2owl.verbose.MainVerbose;

/*****************************************************************
```

```
 * Procedures for dealing with the owl elements to be mapped
 * @param mappingType: indicates the kind of structure that will support the mapping
 * @param OWLClasses: list of classes to be mapped
 * @param OWLObjectProperties: list of object properties to be mapped
 * @param OWLDataTypeProperties: list of data properties to be mapped */

public class OWLStructure
{ public MappingType mappingType = null;
  private List<OWLClass> OWLClasses;
  private List<OWLObjectProperty> OWLObjectProperties;
  private List<OWLDataTypeProperty> OWLDataTypeProperties;

  /*************************************************************
   * Procedures for creating an OWLElements object           */
  public OWLStructure (MappingType mt)
  { this.OWLClasses = new LinkedList<OWLClass>();
    this.OWLObjectProperties = new LinkedList<OWLObjectProperty>();
    this.OWLDataTypeProperties = new LinkedList<OWLDataTypeProperty>();
    this.mappingType = mt;
    this.initiateBasicStructure(); }

  public Boolean is4DView()
  { return ((mappingType == MappingType.WORM_VIEW_A0) || (mappingType == MappingType.WORM_VIEW_A1) ||
        (mappingType == MappingType.WORM_VIEW_A2)); }

  public Boolean isReificationView()
  { return (mappingType == MappingType.REIFICATION); }

  public Boolean isDynamicView()
  { return (is4DView() || isReificationView()) ; }

  public Boolean isStaticView()
  { return (!isDynamicView()); }

  /*************************************************************
   * Procedures for initiating a basic Structure for an OWLElements object   */
  public void initiateBasicStructure ()
  { /*************************************************************
    // STRUCTURAL CLASSES
    /*************************************************************
    // MAIN CLASSES: */
    // INDIVIDUAL
    if (!is4DView())
    { OWLClass ind = addClass("Individual", null);
      String[] s0 = {"Object","Moment"};
      ind.addCompleteClasses(s0);
      if (isReificationView())
        ind.addRestriction("some", "hasTemporalExtent", "TemporalExtent", false, null, null);
    }
    else
    { // INDIVIDUAL CONCEPT
      OWLClass ic = addClass("IndividualConcept", null);
      String[] sic = {"Object","Moment"};
      ic.addCompleteClasses(sic);
      ic.addDisjointClass("TimeSlice");
      ic.addRestriction("equivalentClass", "none", "some", null,
                                                "timeSliceOf", "TimeSlice", true, null, null, null);
      // TIME SLICE
      OWLClass ts = addClass("TimeSlice", null);
      String[] sts={"FunctionalComplexTS","CollectiveTS","QuantityTS","ModeTS","RelatorTS"};
      ts.addCompleteClasses(sts);
```

```
  ts.addDisjointClass("IndividualConcept");
  ts.addRestriction("equivalentClass", "none", "some", null,
                                    "hasTemporalExtent", "TemporalExtent", false, null, null, null);
  ts.addRestriction("equivalentClass", "none", "exactly", null,
                                      "timeSliceOf", "IndividualConcept", false, null, "1", "1");
}

// TEMPORAL EXTENT
if (isDynamicView())
{ OWLClass tem = addClass("TemporalExtent", null);
  if (is4DView())
  { tem.addDisjointClass("IndividualConcept");
    tem.addDisjointClass("TimeSlice"); }
  else tem.addDisjointClass("Individual"); }
/*************************************************************/

/*************************************************************
// SUB-CLASSES OF INDIVIDUAL: */
// OBJECT
String[] s1 = {"FunctionalComplex","Collective","Quantity"};
OWLClass obj;
if (is4DView()) obj = addClass("Object", "IndividualConcept", s1, null);
else obj = addClass("Object", "Individual", s1, null);
obj.addDisjointClass("Moment");

// if it is the reification view, then a restriction can be added for
// addressing the "no-propertyless individual" rule
if (isReificationView())
  obj.addRestriction("some", "inheresIn", "Moment", true, null, null);

// MOMENT
OWLClass mom;
if (!is4DView())
{ String[] s2 = {"IntrinsicMoment","Relator"};
  mom = addClass("Moment", "Individual", s2, null); }
else
{ String[] s2 = {"Mode","Relator"};
  mom = addClass("Moment", "IndividualConcept", s2, null); }
mom.addDisjointClass("Object");
/*************************************************************/

/*************************************************************
// SUB-CLASSES OF OBJECT: */
// FUNCTIONAL COMPLEX
OWLClass fc = addClass("FunctionalComplex", "Object", null, s1);
// COLLECTIVE
OWLClass col = addClass("Collective", "Object", null, s1);
// QUANTITY
OWLClass qt = addClass("Quantity", "Object", null, s1);

if (is4DView())
{ //the corresponding classes are added at the TS-level
  // FUNCTIONAL COMPLEX TS
  OWLClass fcts = addClass("FunctionalComplexTS", "TimeSlice");
  fcts.addRestriction("equivalentClass", "none", "some", null,
                                     "timeSliceOf", "FunctionalComplex", false, null, null, null);
  // COLLECTIVE TS
  OWLClass cts = addClass("CollectiveTS", "TimeSlice");
  cts.addRestriction("equivalentClass", "none", "some", null,
                                         "timeSliceOf", "Collective", false, null, null, null);
  // QUANTITY TS
```

```
  OWLClass qts = addClass("QuantityTS", "TimeSlice");
  qts.addRestriction("equivalentClass", "none", "some", null,
                                                "timeSliceOf", "Quantity", false, null, null, null);
  //Restrictions over the ic-level classes
  fc.addRestriction("subClass", "none", "only", null,
                                     "timeSliceOf", "FunctionalComplexTS", true, null, null, null);
  col.addRestriction("subClass", "none", "only", null,
                                          "timeSliceOf", "CollectiveTS", true, null, null, null);
  qt.addRestriction("subClass", "none", "only", null,
                                           "timeSliceOf", "QuantityTS", true, null, null, null);
}
/*************************************************************/

/*************************************************************
// SUB-CLASSES OF MOMENT: */
// INTRINSIC MOMENT
OWLClass imom = null;
if (!is4DView())
{ imom = addClass("IntrinsicMoment", "Moment");
  imom.addDisjointClass("Relator");
  imom.addRestriction("equivalentClass", "none", "some", null,
                                "inheresIn", "Individual", false, null, null, null); }

// RELATOR
OWLClass rel = addClass("Relator", "Moment");
rel.addDisjointClass("IntrinsicMoment");
if (is4DView())
{ //the corresponding classes are added at the TS-level
  // RELATOR TS
  OWLClass relts = addClass("RelatorTS", "TimeSlice");
  relts.addDisjointClass("ModeTS");
  relts.addRestriction("equivalentClass", "none", "some", null,
                                                 "timeSliceOf", "Relator", false, null, null, null);
  //restrictions over the ic- or ts-level classes depending on the type of 4D view
  if (mappingType == MappingType.WORM_VIEW_A0)
  { relts.addRestriction("equivalentClass", "none", "some", null,
                                                  "mediates", "TimeSlice", false, null, null, null);
    relts.addRestriction("min/max", "mediates", "TimeSlice", false, "2", "-1"); }
  else if (mappingType == MappingType.WORM_VIEW_A2)
  { rel.addRestriction("equivalentClass", "none", "some", null,
                                           "mediates", "IndividualConcept", false, null, null, null);
    rel.addRestriction("min/max", "mediates", "Individual", false, "2", "-1"); }
  rel.addRestriction("subClass", "none", "only", null,
                                                 "timeSliceOf", "RelatorTS", true, null, null, null);
}
else
{ rel.addRestriction("equivalentClass", "none", "some", null,
                                                  "mediates", "Individual", false, null, null, null);
  rel.addRestriction("min/max", "mediates", "Individual", false, "2", "-1");
  if (isReificationView())
    //a restriction concerning the partOfRelator must be added
    rel.addRestriction("min/max", "partOfRelator", "RelationalQuaIndividual", true, "2", "-1");
}
/*************************************************************/

/*************************************************************
// SUB-CLASSES OF INTRINSIC MOMENT: */
// MODE
OWLClass mod;
if (!is4DView()) mod = addClass("Mode", "IntrinsicMoment");
else mod = addClass("Mode", "Moment");
```

```
if (is4DView())
{ //the corresponding classes are added at the TS-level
  // MODE TS
  OWLClass mts = addClass("ModeTS", "TimeSlice");
  mts.addRestriction("equivalentClass", "none", "some", null,
                                                  "timeSliceOf", "Mode", false, null, null, null);
  //restrictions over the ic-level classes
  mod.addRestriction("subClass", "none", "only", null,
                                                 "timeSliceOf", "ModeTS", true, null, null, null);
}
else if (isReificationView())
{ // QUALITY
  OWLClass qual = addClass("Quality", "IntrinsicMoment");
  qual.addDisjointClass("Mode");
  qual.addRestriction("min/max", "hasValue", null, false, "1", "-1");
  // hasValue
  OWLDataTypeProperty hv = addDTProp("hasValue", "Quality", null, null);
  hv.setFunctional();
  // disjoint class of mode
  mod.addDisjointClass("Quality");
  // complete classes of intrinsic moment
  String[] simom = {"Quality","Mode"};
  imom.addCompleteClasses(simom);
}
/*************************************************************/

/***********************************************************
// SUB-CLASSES OF MODE:  */
if (isReificationView())
{ // QUA INDIVIDUAL
  OWLClass qi = addClass("QuaIndividual", "Mode");
  String[] sq = {"PhasedQuaIndividual","RelationalQuaIndividual"};
  qi.addCompleteClasses(sq);

  // RELATIONAL QUA INDIVIDUAL
  OWLClass rqua = addClass("RelationalQuaIndividual", "QuaIndividual");
  rqua.addRestriction("equivalentClass", "none", "some", null,
                                            "partOfRelator", "Relator", false, null, null, null);
  rqua.addDisjointClass("PhasedQuaIndividual");

  // PHASED QUA INDIVIDUAL
  OWLClass pqua = addClass("PhasedQuaIndividual", "QuaIndividual");
  pqua.addDisjointClass("RelationalQuaIndividual");
}
/*************************************************************/

/***********************************************************
//STRUCTURAL OBJECT PROPERTIES
/***********************************************************
// MAIN PROPERTIES:  */
// EXISTENTIALLY DEPENDENT
OWLObjectProperty edo;
if (!is4DView())
  edo = addObjProp("existentiallyDependentOf", "Individual", "Individual",
                                                          null, "invExistentiallyDependentOf");
else
  edo = addObjProp("existentiallyDependentOf", null, null, null, "invExistentiallyDependentOf");
edo.setIrreflexive();

// INVERSE EXISTENTIALLY DEPENDENT
OWLObjectProperty iedo;
```

```
if (!is4DView())
  iedo = addObjProp("invExistentiallyDependentOf", "Individual", "Individual",
                                                                  null, "existentiallyDependentOf");
else
  iedo = addObjProp("invExistentiallyDependentOf", null, null, null, "existentiallyDependentOf");

// PART OF
OWLObjectProperty partOf;
if (!is4DView())
  partOf = addObjProp("partOf", "Individual", "Individual", null, null);
else
{ //this relation holds in different levels according to the type of 4D view
  partOf = addObjProp("partOf", null, null, null, null);
  List<String> ls = new LinkedList<String>();
  if ((mappingType == MappingType.WORM_VIEW_A0) || ((mappingType == MappingType.WORM_VIEW_A1)))
    ls.add("TimeSlice");
  if ((mappingType == MappingType.WORM_VIEW_A2) || ((mappingType == MappingType.WORM_VIEW_A1)))
    ls.add("IndividualConcept");
  partOf.addDomain(ls);
  partOf.addRange(ls);
}
partOf.setAsymmetric(); partOf.setIrreflexive();

if (isReificationView())
{ // HAS TEMPORAL EXTENT
  OWLObjectProperty hte = addObjProp("hasTemporalExtent", "Individual", "TemporalExtent",
                                                                          null, null);
  hte.setFunctional();
}
else if (is4DView())
{ // HAS TEMPORAL EXTENT
  OWLObjectProperty hte = addObjProp("hasTemporalExtent", "TimeSlice", "TemporalExtent",
                                                                          null, null);
  String[] pc = {"objPropertyTS","hasTemporalExtent"};
  hte.addPropChain(pc);
  //the reasoner does not allow a property that is functional and has property chain
  //hte.setFunctional();

  // TIME SLICE OF
  OWLObjectProperty tso = addObjProp("timeSliceOf", "TimeSlice", "IndividualConcept", null, null);
  tso.setFunctional();

  // OBJECT PROPERTY BETWEEN INDIVIDUAL CONCEPTS
  OWLObjectProperty opic = null;
  if (mappingType != MappingType.WORM_VIEW_A0)
  { opic = addObjProp("objPropertyIC", "IndividualConcept", "IndividualConcept", null, null);
    opic.setSymmetric(); }

  // OBJECT PROPERTY BETWEEN TIME SLICES
  OWLObjectProperty opts = addObjProp("objPropertyTS", "TimeSlice", "TimeSlice", null, null);
  opts.setSymmetric();

  if (mappingType == MappingType.WORM_VIEW_A0)
  // all properties are represented at the TS level
  { //all the main properties specializes the obj prop ts
    edo.addSuperProperty("objPropertyTS");
    iedo.addSuperProperty("objPropertyTS");
    partOf.addSuperProperty("objPropertyTS"); }
  else if (mappingType == MappingType.WORM_VIEW_A1)
  // just the mutual existential dependence relations are represented at the IC level
  { //the obj prop between ic's specializes both existencialDependentOf and its inverse properties.
```

```
    opic.addSuperProperty("existentiallyDependentOf");
    opic.addSuperProperty("invExistentiallyDependentOf"); }
  else if (mappingType == MappingType.WORM_VIEW_A2)
  // just the existential dependence relations are represented at the IC level
  { //both existencialDependentOf and its inverse properties specialize the obj prop between ic's
    edo.addSuperProperty("objPropertyIC");
    iedo.addSuperProperty("objPropertyIC"); }
}
/*************************************************************/

/*************************************************************/
// SUB-PROPERTIES OF EXISTENTIALLY DEPENDENT:
// INHERES
OWLObjectProperty i;
if (is4DView())
{ //this relation holds in different levels according to the type of 4D view
  if (mappingType == MappingType.WORM_VIEW_A0) //ts-level
    i = addObjProp("inheresIn","ModeTS","TimeSlice","existentiallyDependentOf", null);
  else if (mappingType == MappingType.WORM_VIEW_A1) //ic- or ts-level
  { i = addObjProp("inheresIn", null, null, "existentiallyDependentOf", null);
    List<String> ld = new LinkedList<String>();
    ld.add("IntrinsicMoment"); ld.add("IntrinsicMomentTS");
    i.addDomain(ld);
    List<String> lr = new LinkedList<String>();
    lr.add("IndividualConcept"); lr.add("TimeSlice");
    i.addRange(lr); }
  else //if (mappingType == mappingType.WORM_VIEW_A2) //ic-level
    i = addObjProp("inheresIn", "IntrinsicMoment", "IndividualConcept", "existentiallyDependentOf",
                                                                                        null);
}
else
  i = addObjProp("inheresIn", "IntrinsicMoment", "Individual", "existentiallyDependentOf", null);
i.setFunctional(); i.setAsymmetric();

// MEDIATES
OWLObjectProperty m;
if (is4DView())
{ //this relation holds in different levels according to the type of 4D view
  if (mappingType == MappingType.WORM_VIEW_A0) //ts-level
    m = addObjProp("mediates", "RelatorTS", "TimeSlice", "existentiallyDependentOf", null);
  else if (mappingType == MappingType.WORM_VIEW_A1) //ic- or ts-level
  { m = addObjProp("mediates", null, null, "existentiallyDependentOf", null);
    List<String> ls = new LinkedList<String>();
    ls.add("Relator");  ls.add("RelatorTS");
    m.addDomain(ls);
    List<String> lr = new LinkedList<String>();
    lr.add("IndividualConcept"); lr.add("TimeSlice");
    m.addRange(lr);
  }
  else //if (mappingType == mappingType.WORM_VIEW_A2)  //ic-level
    m = addObjProp("mediates", "Relator", "IndividualConcept", "existentiallyDependentOf", null);
}
else
  m = addObjProp("mediates", "Relator", "Individual", "existentiallyDependentOf", null);
m.setAsymmetric();


if (isReificationView())
{ // EXISTENTIALLY DEPENDENT OF QUA
  OWLObjectProperty por = addObjProp("existentiallyDependentOfQua", "RelationalQuaIndividual",
                                     "RelationalQuaIndividual", "existentiallyDependentOf", null);
```

```
  por.setSymmetric(); }
/*************************************************************/

/*************************************************************/
// SUB-PROPERTIES OF PART-OF (AND EVENTUALLY OF EXIST-DEP-OF/INV):
// INSEPARABLE PART-OF
OWLObjectProperty ipo = addObjProp("inseparablePartOf", null, null, "partOf", null);
ipo.addSuperProperty("existentiallyDependentOf");

// ESSENTIAL PART-OF
OWLObjectProperty epo = addObjProp("essentialPartOf", null, null, "partOf", null);
epo.addSuperProperty("invExistentiallyDependentOf");

// COMPONENT-OF
List<String> lDomainCOf = new LinkedList<String>();
List<String> lRangeCOf = new LinkedList<String>();
lDomainCOf.add("FunctionalComplex");
lRangeCOf.add("FunctionalComplex");
if (is4DView())
{ //this relation holds also at ts-level
  lDomainCOf.add("FunctionalComplexTS");
  lRangeCOf.add("FunctionalComplexTS"); }
addObjProp("componentOf", lDomainCOf, lRangeCOf, "partOf");

// MEMBER-OF
List<String> lDomainMOf = new LinkedList<String>();
List<String> lRangeMOf = new LinkedList<String>();
lDomainMOf.add("FunctionalComplex");
lDomainMOf.add("Collective");
lRangeMOf.add("Collective");
if (is4DView())
{ //this relation holds also at ts-level
  lDomainMOf.add("FunctionalComplexTS");
  lDomainMOf.add("CollectiveTS");
  lRangeMOf.add("CollectiveTS"); }
addObjProp("memberOf", lDomainMOf, lRangeMOf, "partOf");

// SUB-COLLECTION-OF
List<String> lDomainSCOf = new LinkedList<String>();
List<String> lRangeSCOf = new LinkedList<String>();
lDomainSCOf.add("Collective");
lRangeSCOf.add("Collective");
if (is4DView())
{ //this relation holds also at ts-level
  lDomainSCOf.add("CollectiveTS");
  lRangeSCOf.add("CollectiveTS"); }
addObjProp("subCollectionOf", lDomainSCOf, lRangeSCOf, "partOf");

// SUB-QUANTITY-OF
List<String> lDomainSQOf = new LinkedList<String>();
List<String> lRangeSQOf = new LinkedList<String>();
lDomainSQOf.add("Quantity");
lRangeSQOf.add("Quantity");
if (is4DView())
{ //this relation holds also at ts-level
  lDomainSQOf.add("QuantityTS");
  lRangeSQOf.add("QuantityTS"); }
addObjProp("subQuantityOf", lDomainSQOf, lRangeSQOf, "essentialPartOf");

if (isReificationView())
{ // PART-OF-RELATOR
```

```
    OWLObjectProperty por = addObjProp("partOfRelator", "RelationalQuaIndividual", "Relator",
                                                                    "inseparablePartOf", null);
    por.addSuperProperty("essentialPartOf");
    por.setFunctional(); }
  /*************************************************************/
}

/*************************************************************
  * Procedures for adding a Class to the OWL Elements */
public OWLClass addClass(String name, String superClass)
{ OWLClass n = new OWLClass(name, superClass);
  OWLClasses.add(n);
  return n;
}

public OWLClass addClass(String name, String superClass, String[] completeClassesNames,
                                                                    String[] disjointClassesNames)
{ if (name == null) return null;

  List<String> superClasses = new LinkedList<String>();
  if (superClass != null)
  { superClasses = new LinkedList<String>();
    superClasses.add(superClass); }

  List<List<String>> llcompleteClasses = null;
  if (completeClassesNames != null)
  { llcompleteClasses = new LinkedList<List<String>>();
    llcompleteClasses.add(Arrays.asList(completeClassesNames)); }

  List<String> ldisjointClasses = null;
  if (disjointClassesNames != null)
  { ldisjointClasses = Arrays.asList(disjointClassesNames); }

  OWLClass n = new OWLClass(name, superClasses, llcompleteClasses, ldisjointClasses);
  OWLClasses.add(n);
  return n;
}

public OWLClass addClass(String name, List<String> superClasses, List<List<String>>
                                          completeClassesNames, List<String> disjointClassesNames)
{ OWLClass n = new OWLClass(name, superClasses, completeClassesNames, disjointClassesNames);
  OWLClasses.add(n);
  return n;
}

/*************************************************************
  * Procedures for getting a Class to the OWL Elements */
  public OWLClass getOWLClass(String name)
{ for (OWLClass c : OWLClasses) { if (c.name.equals(name)) return c; }
  return null;
}

/*************************************************************
  * Procedures for adding an Object Property to the OWL Elements */
public OWLObjectProperty addObjProp(String name, String domain, String range, String superProp,
                                                                    String invProp)
{ OWLObjectProperty n = new OWLObjectProperty(name, domain, range, superProp, invProp);
  OWLObjectProperties.add(n);
  return n;
}
```

```
public OWLObjectProperty addObjProp(String name, List<String> disjDomains, List<String> disjRanges,
                                                                        List<String> lSuperProperties)
{ OWLObjectProperty n = new OWLObjectProperty(name, disjDomains, disjRanges, lSuperProperties);
  OWLObjectProperties.add(n);
  return n;
}

public OWLObjectProperty addObjProp(String name, List<String> disjDomains, List<String> disjRanges,
                                                                            String superProperty)
{ List<String> lsp = null;
  if (superProperty != null)
  { lsp = new LinkedList<String>();
    lsp.add(superProperty); }
  OWLObjectProperty n = new OWLObjectProperty(name, disjDomains, disjRanges, lsp);
  OWLObjectProperties.add(n);
  return n;
}

/*************************************************************
  * Procedures for getting an Object Property from the OWL Elements */
public OWLObjectProperty getOWLObjProp(String name)
{
  for (OWLObjectProperty p : OWLObjectProperties)
  { if (p.name.equals(name)) return p; }
  return null;
}

/*************************************************************
  * Procedures for adding an DataType Property to the OWL Elements */
public OWLDataTypeProperty addDTProp(String name, String domain, String range, String superProp)
{
  OWLDataTypeProperty n = new OWLDataTypeProperty(name, domain, range, superProp);
  OWLDataTypeProperties.add(n);
  return n;
}

/*************************************************************
  * Procedures for printing the OWL Elements            */
public String verbose(String modelName)
{ String out = "";
  for (OWLObjectProperty p : OWLObjectProperties) { out += p.verbose(); }
  for (OWLDataTypeProperty p : OWLDataTypeProperties) { out += p.verbose(); }
  for (OWLClass c : OWLClasses) { out += c.verbose(); }
  MainVerbose.setModelId(modelName);
  return MainVerbose.initialVerbose() + out + MainVerbose.finalVerbose();
}

/*************************************************************
  * Procedures for mapping the treeProcessor elements into OWLStructure    */
  public void map(TreeProcessor tp)
  { completeOWLStructuralClasses();
    for (NodeClass nc : tp.nodes)
    { mapNodeClass2OWL(nc);
      mapAttributes2OWL(nc); }
    for (NodeBinAssociation na : tp.assocNodes)
      mapNodeAssociation2OWL(na);
  }

  public void completeOWLStructuralClasses()
  {
     OWLClass oc;
```

```
    oc = getOWLClass("FunctionalComplex");
    oc.addCompleteClasses(TreeProcessor.kindsNames);
    oc = getOWLClass("Collective");
    oc.addCompleteClasses(TreeProcessor.collectivesNames);
    oc = getOWLClass("Quantity");
    oc.addCompleteClasses(TreeProcessor.quantitiesNames);
    oc = getOWLClass("Relator");
    oc.addCompleteClasses(TreeProcessor.relatorsNames);

    if (isReificationView())
    { if (TreeProcessor.modesNames == null)
        TreeProcessor.modesNames = new LinkedList<String>();
      TreeProcessor.modesNames.add("QuaIndividual"); }
    oc = getOWLClass("Mode");
    if (oc != null) oc.addCompleteClasses(TreeProcessor.modesNames);
    oc = getOWLClass("RelationalQuaIndividual");
    if (oc != null) oc.addCompleteClasses(TreeProcessor.relationalquasNames);
    oc = getOWLClass("PhasedQuaIndividual");
    if (oc != null) oc.addCompleteClasses(TreeProcessor.phasedquasNames);
    oc = getOWLClass("Quality");
    if (oc != null) oc.addCompleteClasses(TreeProcessor.qualitiesNames);
}

public void mapNodeClass2OWL(NodeClass n)
{  String className;
    List<String> lparents = null;
    List<List<String>> llchildren = null;
    List<String> lsiblings = null;

    Boolean addTS = is4DView();
    if ((isReificationView()) && (!n.isRigid()))
    { className = n.getReifiedName();
      lparents = new LinkedList<String>();
      lparents.add(n.getStructuralParent()); }
    else
    { //if is4DView, the class is created at the ts level
      className = n.getName(addTS);
      lparents = n.getParentsNames(addTS); }

    llchildren = n.getCompleteChildren(addTS, is4DView(), isReificationView());
    lsiblings = n.getDisjointSiblingsNames(addTS, isReificationView());

    OWLClass oc = addClass(className, lparents, llchildren, lsiblings);
    if (is4DView())
    { if (n.isRigid())
      { addTS = false;
        className = n.getName(addTS);
        lparents = n.getParentsNames(addTS);
        llchildren = n.getCompleteChildren(addTS, is4DView(), isReificationView());
        lsiblings = n.getDisjointSiblingsNames(addTS, isReificationView());
        OWLClass ocic = addClass(className, lparents, llchildren, lsiblings);

        //adding restrictions
        oc.addRestriction("equivalentClass", "none", "some", null, "timeSliceOf", n.getName(),
                                                                    false, null, null, null);
        ocic.addRestriction("subClass", "none", "only", null, "timeSliceOf", n.getTSName(), true,
                                                                  null, null, null);
      }
    }

    if ((isReificationView()) && (!n.isRigid()))
```

```
    {
      // rigid parents are those in which the qua inheres
      List<String> inhParents = n.getRigidParentsNames(false);
      if (inhParents != null)
        oc.addRestriction("subClass", "none", "some", null, "inheresIn", null, false, inhParents,
                                                                                    null, null);
      // reified parents are those in which the qua is existentially dependent of
      List<String> reifiedParents = n.getReifiedParentsNames();
      if (reifiedParents != null)
        oc.addRestriction("subClass", "none", "some", null, "existentiallyDependentOf", null,
                                                          false, reifiedParents, null, null);
    }

  }

  public void mapAttributes2OWL(NodeClass n)
  {List<Property> properties = n.getRelatedClass().getOwnedAttribute();
   if (properties != null)
    for (Property p : properties)
    { String name = p.getName() + n.getName();
      String domain, range = null;
      Integer min = p.lowerBound(), max = p.upperBound();

      // Improvising an way of getting datatype from OntoUML model
      // due to a kind of bug of Eclipse on recognizing Primitive Types
      if (p.getType() != null)
      { String datatype = p.getType().getName().toLowerCase();
        if ( datatype.equals("string") || datatype.equals("integer") ||
           datatype.equals("short")  || datatype.equals("long")    ||
           datatype.equals("double") || datatype.equals("float")   ||
           datatype.equals("date")   || datatype.equals("time")    ||
           datatype.equals("dateTime") )
        range = "&xsd;" + datatype;
      }

      if (isReificationView())
      { OWLClass qlt = addClass(/*NAME:*/ name, /*SUPERCLASS:*/ "Quality");
        qlt.addRestriction("some", "inheresIn", n.getName(), false, null, null);
        qlt.addRestriction("some", "hasValue", range, false, null, null);
        if (TreeProcessor.qualitiesNames == null)
          TreeProcessor.qualitiesNames = new LinkedList<String>();
        TreeProcessor.qualitiesNames.add(name);

        if (!p.isIsReadOnly())  max = -1;
        if (n.isRigid()) domain = n.getName();
        else domain = n.getReifiedName();

        OWLClass oc = getOWLClass(domain);
        oc.addSubClassCardRestriction("inheresIn", name, true, min, max);
      }
      else
      { Boolean addTS = (is4DView()) && ((mappingType == MappingType.WORM_VIEW_A0)
                          || !(p.isIsReadOnly() && n.isRigid() && (min > 0)));
        domain = n.getName(addTS);
        addDTProp(name, domain, range, /*SUPER-CLASS:*/ null);
        if ((min > 0) || (max > 0))
        { OWLClass oc = getOWLClass(domain);
          oc.addSubClassCardRestriction(name, "", false, min, max); }
      }
    }
}
```

```
    public void mapNodeAssociation2OWL(NodeBinAssociation n)
    {
    if ((n.mappingName != null) &&
        !(isReificationView() && (n.myAssociation instanceof MaterialAssociation)) )
        //create the property
        addObjProp(n.mappingName, n.getDomainList(mappingType),
                     n.getRangeList(mappingType),
                     n.getSuperAssocList(mappingType));
      n.setRestrictions(mappingType);
      if (n.listRestrictions != null)
        for(Restriction r : n.listRestrictions)
        {
          OWLClass oc = getOWLClass(r.restrictedClass);
          if (oc != null)
          {
            if (r.equivalent)
              oc.addEquivClassCardRestriction(r.propertyName, r.listClasses, r.inverse, r.min, r.max);
            else
              oc.addSubClassCardRestriction(r.propertyName, r.listClasses, r.inverse, r.min, r.max);
          }
        }
    }
}
```

APÊNDICE II – Ontologias OWL resultantes do mapeamento do modelo de Exemplo

//---------- 4D-A0 EXAMPLE ----------//

Ontology(<example-4DA0.owl>
// Class: http://www.w3.org/2002/07/owl#Thing

// STRUCTURAL CLASSES

// Class: example-4DA0.owl#TemporalExtent
DisjointClasses(TemporalExtent IndividualConcept TimeSlice)
// Class: example-4DA0.owl#IndividualConcept
EquivalentClasses(IndividualConcept **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) TimeSlice))
EquivalentClasses(IndividualConcept **ObjectUnionOf**(Object Moment))
// Class: example-4DA0.owl#Object
EquivalentClasses(Object **ObjectUnionOf**(Collective FunctionalComplex Quantity))
SubClassOf(Object IndividualConcept)
// Class: example-4DA0.owl#FunctionalComplex
EquivalentClasses(FunctionalComplex **ObjectUnionOf**(Cerebro Coracao Pessoa))
SubClassOf(FunctionalComplex Object)
SubClassOf(FunctionalComplex **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) FunctionalComplexTS))
DisjointClasses(FunctionalComplex Quantity Collective)
// Class: example-4DA0.owl#Collective
SubClassOf(Collective Object)
SubClassOf(Collective **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CollectiveTS))
// Class: example-4DA0.owl#Quantity
SubClassOf(Quantity Object)
SubClassOf(Quantity **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) QuantityTS))
// Class: example-4DA0.owl#Moment
EquivalentClasses(Moment **ObjectUnionOf**(Relator Mode))
SubClassOf(Moment IndividualConcept)
// Class: example-4DA0.owl#Mode
SubClassOf(Mode **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) ModeTS))
SubClassOf(Mode Moment)
DisjointClasses(Mode Relator)

// Class: example-4DA0.owl#Relator
EquivalentClasses(Relator Casamento)
SubClassOf(Relator Moment)
SubClassOf(Relator **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) RelatorTS))

// Class: example-4DA0.owl#TimeSlice
EquivalentClasses(TimeSlice **ObjectExactCardinality**(1 timeSliceOf IndividualConcept))
EquivalentClasses(TimeSlice **ObjectUnionOf**(CollectiveTS FunctionalComplexTS ModeTS QuantityTS RelatorTS))
EquivalentClasses(TimeSlice **ObjectSomeValuesFrom**(hasTemporalExtent TemporalExtent))
// Class: example-4DA0.owl#FunctionalComplexTS
EquivalentClasses(FunctionalComplexTS **ObjectSomeValuesFrom**(timeSliceOf FunctionalComplex))
SubClassOf(FunctionalComplexTS TimeSlice)
// Class: example-4DA0.owl#CollectiveTS
EquivalentClasses(CollectiveTS **ObjectSomeValuesFrom**(timeSliceOf Collective))
SubClassOf(CollectiveTS TimeSlice)
// Class: example-4DA0.owl#QuantityTS
EquivalentClasses(QuantityTS **ObjectSomeValuesFrom**(timeSliceOf Quantity))
SubClassOf(QuantityTS TimeSlice)
// Class: example-4DA0.owl#ModeTS
EquivalentClasses(ModeTS **ObjectSomeValuesFrom**(timeSliceOf Mode))
SubClassOf(ModeTS TimeSlice)
DisjointClasses(ModeTS RelatorTS)
// Class: example-4DA0.owl#RelatorTS
EquivalentClasses(RelatorTS **ObjectSomeValuesFrom**(mediates TimeSlice))
EquivalentClasses(RelatorTS **ObjectSomeValuesFrom**(timeSliceOf Relator))
SubClassOf(RelatorTS TimeSlice)
SubClassOf(RelatorTS **ObjectMinCardinality**(2 mediates TimeSlice))

// DOMAIN CLASSES

// Class: example-4DA0.owl#Pessoa
EquivalentClasses(Pessoa ObjectUnionOf(Mulher Homem))
SubClassOf(Pessoa **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) PessoaTS))
SubClassOf(Pessoa FunctionalComplex)
DisjointClasses(Pessoa Cerebro Coracao)
// Class: example-4DA0.owl#Homem
SubClassOf(Homem **ObjectAllValuesFrom**(InverseObjectProperty(timeSliceOf) HomemTS))
SubClassOf(Homem Pessoa)
DisjointClasses(Homem Mulher)
// Class: example-4DA0.owl#Mulher
SubClassOf(Mulher **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) MulherTS))
SubClassOf(Mulher Pessoa)
// Class: example-4DA0.owl#OrgaoHumano
EquivalentClasses(OrgaoHumano **ObjectUnionOf**(Coracao Cerebro))
SubClassOf(OrgaoHumano **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) OrgaoHumanoTS))
// Class: example-4DA0.owl#Coracao
SubClassOf(Coracao **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CoracaoTS))
SubClassOf(Coracao FunctionalComplex)
// Class: example-4DA0.owl#Cerebro
SubClassOf(Cerebro FunctionalComplex)
SubClassOf(Cerebro **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CerebroTS))
// Class: example-4DA0.owl#Casamento
EquivalentClasses(Casamento Relator)
SubClassOf(Casamento **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CasamentoTS))
SubClassOf(Casamento Relator)

// Class: example-4DA0.owl#PessoaTS
EquivalentClasses(PessoaTS **ObjectSomeValuesFrom**(timeSliceOf Pessoa))
EquivalentClasses(PessoaTS **ObjectUnionOf**(MulherTS HomemTS))
SubClassOf(PessoaTS **ObjectExactCardinality**(1 **InverseObjectProperty**(componentOfCoracaoPessoa) CoracaoTS))
SubClassOf(PessoaTS FunctionalComplexTS)
SubClassOf(PessoaTS **ObjectExactCardinality**(1 **InverseObjectProperty**(componentOfCerebroPessoa) CerebroTS))
SubClassOf(PessoaTS **DataExactCardinality**(1 nomePessoa))
SubClassOf(PessoaTS **DataMaxCardinality**(1 cpfPessoa))
SubClassOf(PessoaTS **DataExactCardinality**(1 idadePessoa))
DisjointClasses(PessoaTS CoracaoTS CerebroTS)
// Class: example-4DA0.owl#HomemTS
EquivalentClasses(HomemTS **ObjectSomeValuesFrom**(timeSliceOf Homem))
SubClassOf(HomemTS PessoaTS)
DisjointClasses(HomemTS MulherTS)
// Class: example-4DA0.owl#MulherTS
EquivalentClasses(MulherTS **ObjectSomeValuesFrom**(timeSliceOf Mulher))
SubClassOf(MulherTS PessoaTS)
// Class: example-4DA0.owl#EsposoTS
SubClassOf(EsposoTS ConjugeTS)
SubClassOf(EsposoTS HomemTS)
DisjointClasses(EsposoTS EsposaTS)
// Class: example-4DA0.owl#EsposaTS
SubClassOf(EsposaTS ConjugeTS)
SubClassOf(EsposaTS MulherTS)
// Class: example-4DA0.owl#CerebroTS
EquivalentClasses(CerebroTS **ObjectSomeValuesFrom**(timeSliceOf Cerebro))
SubClassOf(CerebroTS FunctionalComplexTS)
SubClassOf(CerebroTS **ObjectMaxCardinality**(1 componentOf PessoaTS))
SubClassOf(CerebroTS **ObjectExactCardinality**(1 componentOfCerebroPessoa PessoaTS))
// Class: example-4DA0.owl#CoracaoTS
EquivalentClasses(CoracaoTS **ObjectSomeValuesFrom**(timeSliceOf Coracao))
SubClassOf(CoracaoTS **ObjectExactCardinality**(1 componentOfCoracaoPessoa PessoaTS))
SubClassOf(CoracaoTS **ObjectMaxCardinality**(1 componentOf PessoaTS))
SubClassOf(CoracaoTS FunctionalComplexTS)
// Class: example-4DA0.owl#OrgaoHumanoTS
EquivalentClasses(OrgaoHumanoTS **ObjectSomeValuesFrom**(timeSliceOf OrgaoHumano))
EquivalentClasses(OrgaoHumanoTS **ObjectUnionOf**(CoracaoTS CerebroTS))
// Class: example-4DA0.owl#ConjugeTS
EquivalentClasses(ConjugeTS **ObjectUnionOf**(EsposoTS EsposaTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1 casadoComConjugeConjuge ConjugeTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1 **InverseObjectProperty**(casadoComConjugeConjuge) ConjugeTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1 InverseObjectProperty(mediates) CasamentoTS))

SubClassOf(ConjugeTS PessoaTS)
// Class: example-4DA0.owl#CasamentoTS
EquivalentClasses(CasamentoTS **ObjectSomeValuesFrom**(timeSliceOf Casamento))

// STRUCTURAL OBJECT PROPERTIES

// Object property: example-4DA0.owl#timeSliceOf
FunctionalObjectProperty(timeSliceOf)
ObjectPropertyDomain(timeSliceOf TimeSlice)
ObjectPropertyRange(timeSliceOf IndividualConcept)
// Object property: example-4DA0.owl#hasTemporalExtent
ObjectPropertyDomain(hasTemporalExtent TimeSlice)
ObjectPropertyRange(hasTemporalExtent TemporalExtent)
// Object property: example-4DA0.owl#objPropertyTS
SymmetricObjectProperty(objPropertyTS)
ObjectPropertyDomain(objPropertyTS TimeSlice)
ObjectPropertyRange(objPropertyTS TimeSlice)
// Object property: example-4DA0.owl#partOf
SubObjectPropertyOf(partOf objPropertyTS)
AntisymmetricObjectProperty(partOf)
IrreflexiveObjectProperty(partOf)
ObjectPropertyDomain(partOf TimeSlice)
ObjectPropertyRange(partOf TimeSlice)
// Object property: example-4DA0.owl#essentialPartOf
SubObjectPropertyOf(essentialPartOf partOf)
SubObjectPropertyOf(essentialPartOf invExistentiallyDependentOf)
// Object property: example-4DA0.owl#inseparablePartOf
SubObjectPropertyOf(inseparablePartOf partOf)
SubObjectPropertyOf(inseparablePartOf existentiallyDependentOf)
// Object property: example-4DA0.owl#componentOf
SubObjectPropertyOf(componentOf partOf)
ObjectPropertyDomain(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))
ObjectPropertyRange(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))
// Object property: example-4DA0.owl#memberOf
SubObjectPropertyOf(memberOf partOf)
ObjectPropertyDomain(memberOf **ObjectUnionOf**(Collective CollectiveTS FunctionalComplex FunctionalComplexTS))
ObjectPropertyRange(memberOf **ObjectUnionOf**(CollectiveTS Collective))
// Object property: example-4DA0.owl#subCollectionOf
SubObjectPropertyOf(subCollectionOf partOf)
ObjectPropertyDomain(subCollectionOf ObjectUnionOf(CollectiveTS Collective))
ObjectPropertyRange(subCollectionOf ObjectUnionOf(CollectiveTS Collective))
// Object property: example-4DA0.owl#subQuantityOf
SubObjectPropertyOf(subQuantityOf essentialPartOf)
ObjectPropertyDomain(subQuantityOf ObjectUnionOf(QuantityTS Quantity))
ObjectPropertyRange(subQuantityOf ObjectUnionOf(QuantityTS Quantity))
// Object property: example-4DA0.owl#existentiallyDependentOf
SubObjectPropertyOf(existentiallyDependentOf objPropertyTS)
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
IrreflexiveObjectProperty(existentiallyDependentOf)
// Object property: example-4DA0.owl#invExistentiallyDependentOf
SubObjectPropertyOf(invExistentiallyDependentOf objPropertyTS)
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
// Object property: example-4DA0.owl#inheresIn
SubObjectPropertyOf(inheresIn existentiallyDependentOf)
FunctionalObjectProperty(inheresIn)
AntisymmetricObjectProperty(inheresIn)
ObjectPropertyDomain(inheresIn ModeTS)
ObjectPropertyRange(inheresIn TimeSlice)
// Object property: example-4DA0.owl#mediates
SubObjectPropertyOf(mediates existentiallyDependentOf)
AntisymmetricObjectProperty(mediates)
ObjectPropertyDomain(mediates RelatorTS)
ObjectPropertyRange(mediates TimeSlice)

// Sub property chain axiom
SubObjectPropertyOf(**SubObjectPropertyChain**(objPropertyTS hasTemporalExtent) hasTemporalExtent)

SubClassOf(CasamentoTS **ObjectExactCardinality**(2 mediates ConjugeTS))
SubClassOf(CasamentoTS RelatorTS)

// DOMAIN OBJECT PROPERTIES

// Object property: example-4DA0.owl#componentOfCerebroPessoa
SubObjectPropertyOf(componentOfCerebroPessoa invExistentiallyDependentOf)
SubObjectPropertyOf(componentOfCerebroPessoa componentOf)
SubObjectPropertyOf(componentOfCerebroPessoa existentiallyDependentOf)
ObjectPropertyDomain(componentOfCerebroPessoa CerebroTS)
ObjectPropertyRange(componentOfCerebroPessoa PessoaTS)
// Object property: example-4DA0.owl#componentOfCoracaoPessoa
SubObjectPropertyOf(componentOfCoracaoPessoa existentiallyDependentOf)
SubObjectPropertyOf(componentOfCoracaoPessoa componentOf)
ObjectPropertyDomain(componentOfCoracaoPessoa CoracaoTS)
ObjectPropertyRange(componentOfCoracaoPessoa PessoaTS)
// Object property: example-4DA0.owl#casadoCom
// Object property: example-4DA0.owl#casadoComConjugeConjuge
SubObjectPropertyOf(casadoComConjugeConjuge casadoCom)
ObjectPropertyDomain(casadoComConjugeConjuge ConjugeTS)
ObjectPropertyRange(casadoComConjugeConjuge ConjugeTS)

// DOMAIN DATA PROPERTIES

// Data property: example-4DA0.owl#nomePessoa
DataPropertyDomain(nomePessoa PessoaTS)
// Data property: example-4DA0.owl#apelidoPessoa
DataPropertyDomain(apelidoPessoa PessoaTS)
// Data property: example-4DA0.owl#idadePessoa
DataPropertyDomain(idadePessoa PessoaTS)
// Data property: example-4DA0.owl#cpfPessoa
DataPropertyDomain(cpfPessoa PessoaTS)
)

//---------- 4D-A1 EXAMPLE ----------//
Ontology(<example-4DA1.owl>
// Class: http://www.w3.org/2002/07/owl#Thing

// STRUCTURAL CLASSES

// Class: example-4DA1.owl#TemporalExtent
DisjointClasses(TemporalExtent IndividualConcept TimeSlice)
// Class: example-4DA1.owl#IndividualConcept
EquivalentClasses(IndividualConcept **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) TimeSlice))
EquivalentClasses(IndividualConcept **ObjectUnionOf**(Object Moment))
// Class: example-4DA1.owl#Object
EquivalentClasses(Object **ObjectUnionOf**(Collective FunctionalComplex Quantity))
SubClassOf(Object IndividualConcept)
DisjointClasses(Object Moment)
// Class: example-4DA1.owl#FunctionalComplex
EquivalentClasses(FunctionalComplex **ObjectUnionOf**(Cerebro Coracao Pessoa))
SubClassOf(FunctionalComplex Object)
SubClassOf(FunctionalComplex **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) FunctionalComplexTS))
DisjointClasses(FunctionalComplex Quantity Collective)
// Class: example-4DA1.owl#Collective
SubClassOf(Collective Object)
SubClassOf(Collective **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CollectiveTS))
// Class: example-4DA1.owl#Quantity
SubClassOf(Quantity Object)
SubClassOf(Quantity **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) QuantityTS))
// Class: example-4DA1.owl#Moment
EquivalentClasses(Moment **ObjectUnionOf**(Relator Mode))
SubClassOf(Moment IndividualConcept)
// Class: example-4DA1.owl#Mode

SubClassOf(Mode **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) ModeTS))
SubClassOf(Mode Moment)
DisjointClasses(Mode Relator)
// Class: example-4DA1.owl#Relator
EquivalentClasses(Relator Casamento)
SubClassOf(Relator Moment)
SubClassOf(Relator **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) RelatorTS))

// Class: example-4DA1.owl#TimeSlice
EquivalentClasses(TimeSlice **ObjectExactCardinality**(1 timeSliceOf IndividualConcept))
EquivalentClasses(TimeSlice **ObjectUnionOf**(CollectiveTS FunctionalComplexTS ModeTS QuantityTS RelatorTS))
EquivalentClasses(TimeSlice **ObjectSomeValuesFrom**(hasTemporalExtent TemporalExtent))
// Class: example-4DA1.owl#FunctionalComplexTS
EquivalentClasses(FunctionalComplexTS **ObjectSomeValuesFrom**(timeSliceOf FunctionalComplex))

// DOMAIN CLASSES

// Class: example-4DA1.owl#Pessoa
EquivalentClasses(Pessoa **ObjectUnionOf**(Mulher Homem))
SubClassOf(Pessoa **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) PessoaTS))
SubClassOf(Pessoa FunctionalComplex)
SubClassOf(Pessoa **DataExactCardinality**(1 nomePessoa))
SubClassOf(Pessoa **ObjectExactCardinality**(1 **InverseObjectProperty**(componentOfCerebroPessoa) Cerebro))
DisjointClasses(Pessoa Cerebro Coracao)
// Class: example-4DA1.owl#Homem
SubClassOf(Homem **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) HomemTS))
SubClassOf(Homem Pessoa)
DisjointClasses(Homem Mulher)
// Class: example-4DA1.owl#Mulher
SubClassOf(Mulher **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) MulherTS))
SubClassOf(Mulher Pessoa)
// Class: example-4DA1.owl#Casamento
EquivalentClasses(Casamento Relator)
SubClassOf(Casamento **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CasamentoTS))
SubClassOf(Casamento Relator)
// Class: example-4DA1.owl#OrgaoHumano
SubClassOf(FunctionalComplexTS TimeSlice)
// Class: example-4DA1.owl#CollectiveTS
EquivalentClasses(CollectiveTS **ObjectSomeValuesFrom**(timeSliceOf Collective))
SubClassOf(CollectiveTS TimeSlice)
// Class: example-4DA1.owl#QuantityTS
EquivalentClasses(QuantityTS **ObjectSomeValuesFrom**(timeSliceOf Quantity))
SubClassOf(QuantityTS TimeSlice)
// Class: example-4DA1.owl#ModeTS
EquivalentClasses(ModeTS **ObjectSomeValuesFrom**(timeSliceOf Mode))
SubClassOf(ModeTS TimeSlice)
DisjointClasses(RelatorTS ModeTS)
// Class: example-4DA1.owl#RelatorTS
EquivalentClasses(RelatorTS **ObjectSomeValuesFrom**(timeSliceOf Relator))
SubClassOf(RelatorTS TimeSlice)

EquivalentClasses(OrgaoHumano **ObjectUnionOf**(Coracao Cerebro))
SubClassOf(OrgaoHumano **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) OrgaoHumanoTS))
// Class: example-4DA1.owl#Coracao
SubClassOf(Coracao **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CoracaoTS))
SubClassOf(Coracao FunctionalComplex)
// Class: example-4DA1.owl#Cerebro
SubClassOf(Cerebro **ObjectExactCardinality**(1 componentOfCerebroPessoa Pessoa))
SubClassOf(Cerebro **ObjectMaxCardinality**(1 componentOf Pessoa))
SubClassOf(Cerebro FunctionalComplex)
SubClassOf(Cerebro **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CerebroTS))

// Class: example-4DA1.owl#PessoaTS
EquivalentClasses(PessoaTS **ObjectSomeValuesFrom**(timeSliceOf Pessoa))
EquivalentClasses(PessoaTS **ObjectUnionOf**(MulherTS HomemTS))
SubClassOf(PessoaTS **ObjectExactCardinality**(1 **InverseObjectProperty**(componentOfCoracaoPessoa) CoracaoTS))
SubClassOf(PessoaTS FunctionalComplexTS)
SubClassOf(PessoaTS **DataMaxCardinality**(1 cpfPessoa))
SubClassOf(PessoaTS **DataExactCardinality**(1 idadePessoa))

DisjointClasses(PessoaTS CoracaoTS CerebroTS)
// Class: example-4DA1.owl#HomemTS
EquivalentClasses(HomemTS **ObjectSomeValuesFrom**(timeSliceOf Homem))
SubClassOf(HomemTS PessoaTS)
DisjointClasses(HomemTS MulherTS)
// Class: example-4DA1.owl#MulherTS
EquivalentClasses(MulherTS **ObjectSomeValuesFrom**(timeSliceOf Mulher))
SubClassOf(MulherTS PessoaTS)
// Class: example-4DA1.owl#ConjugeTS
EquivalentClasses(ConjugeTS **ObjectUnionOf**(EsposoTS EsposaTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1 casadoComConjugeConjuge ConjugeTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1
InverseObjectProperty(casadoComConjugeConjuge) ConjugeTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1 **InverseObjectProperty**(mediates) CasamentoTS))
SubClassOf(ConjugeTS PessoaTS)
// Class: example-4DA1.owl#EsposoTS
SubClassOf(EsposoTS ConjugeTS)
SubClassOf(EsposoTS HomemTS)
DisjointClasses(EsposoTS EsposaTS)

// STRUCTURAL OBJECT PROPERTIES

// Object property: example-4DA1.owl#timeSliceOf
FunctionalObjectProperty(timeSliceOf)
ObjectPropertyDomain(timeSliceOf TimeSlice)
ObjectPropertyRange(timeSliceOf IndividualConcept)
// Object property: example-4DA1.owl#objPropertyIC
SubObjectPropertyOf(objPropertyIC invExistentiallyDependentOf)
SubObjectPropertyOf(objPropertyIC existentiallyDependentOf)
SymmetricObjectProperty(objPropertyIC)
ObjectPropertyDomain(objPropertyIC IndividualConcept)
ObjectPropertyRange(objPropertyIC IndividualConcept)
// Object property: example-4DA1.owl#objPropertyTS
SymmetricObjectProperty(objPropertyTS)
ObjectPropertyDomain(objPropertyTS TimeSlice)
ObjectPropertyRange(objPropertyTS TimeSlice)
// Object property: example-4DA1.owl#hasTemporalExtent
ObjectPropertyDomain(hasTemporalExtent TimeSlice)

// Class: example-4DA1.owl#EsposaTS
SubClassOf(EsposaTS ConjugeTS)
SubClassOf(EsposaTS MulherTS)
// Class: example-4DA1.owl#OrgaoHumanoTS
EquivalentClasses(OrgaoHumanoTS **ObjectSomeValuesFrom**(timeSliceOf OrgaoHumano))
EquivalentClasses(OrgaoHumanoTS **ObjectUnionOf**(CoracaoTS CerebroTS))
// Class: example-4DA1.owl#CerebroTS
EquivalentClasses(CerebroTS **ObjectSomeValuesFrom**(timeSliceOf Cerebro))
SubClassOf(CerebroTS FunctionalComplexTS)
// Class: example-4DA1.owl#CoracaoTS
EquivalentClasses(CoracaoTS **ObjectSomeValuesFrom**(timeSliceOf Coracao))
SubClassOf(CoracaoTS **ObjectExactCardinality**(1 componentOfCoracaoPessoa PessoaTS))
SubClassOf(CoracaoTS **ObjectMaxCardinality**(1 componentOf PessoaTS))
SubClassOf(CoracaoTS FunctionalComplexTS)
// Class: example-4DA1.owl#CasamentoTS
EquivalentClasses(CasamentoTS **ObjectSomeValuesFrom**(timeSliceOf Casamento))
SubClassOf(CasamentoTS **ObjectExactCardinality**(2 mediates ConjugeTS))
SubClassOf(CasamentoTS RelatorTS)

ObjectPropertyRange(hasTemporalExtent TemporalExtent)
// Object property: example-4DA1.owl#partOf
AntisymmetricObjectProperty(partOf)
IrreflexiveObjectProperty(partOf)
ObjectPropertyDomain(partOf **ObjectUnionOf**(TimeSlice IndividualConcept))
ObjectPropertyRange(partOf **ObjectUnionOf**(TimeSlice IndividualConcept))
// Object property: example-4DA1.owl#essentialPartOf
SubObjectPropertyOf(essentialPartOf partOf)
SubObjectPropertyOf(essentialPartOf invExistentiallyDependentOf)
// Object property: example-4DA1.owl#inseparablePartOf
SubObjectPropertyOf(inseparablePartOf partOf)
SubObjectPropertyOf(inseparablePartOf existentiallyDependentOf)
// Object property: example-4DA1.owl#componentOf
SubObjectPropertyOf(componentOf partOf)
ObjectPropertyDomain(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))
ObjectPropertyRange(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))

// Object property: example-4DA1.owl#subCollectionOf
SubObjectPropertyOf(subCollectionOf partOf)
ObjectPropertyDomain(subCollectionOf **ObjectUnionOf**(CollectiveTS Collective))
ObjectPropertyRange(subCollectionOf **ObjectUnionOf**(CollectiveTS Collective))
// Object property: example-4DA1.owl#memberOf
SubObjectPropertyOf(memberOf partOf)
ObjectPropertyDomain(memberOf **ObjectUnionOf**(Collective CollectiveTS FunctionalComplex FunctionalComplexTS))
ObjectPropertyRange(memberOf **ObjectUnionOf**(CollectiveTS Collective))
// Object property: example-4DA1.owl#subQuantityOf
SubObjectPropertyOf(subQuantityOf essentialPartOf)
ObjectPropertyDomain(subQuantityOf **ObjectUnionOf**(QuantityTS Quantity))
ObjectPropertyRange(subQuantityOf **ObjectUnionOf**(QuantityTS Quantity))
// Object property: example-4DA1.owl#existentiallyDependentOf
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
IrreflexiveObjectProperty(existentiallyDependentOf)
// Object property: example-4DA1.owl#invExistentiallyDependentOf
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
// Object property: example-4DA1.owl#inheresIn
SubObjectPropertyOf(inheresIn existentiallyDependentOf)
FunctionalObjectProperty(inheresIn)
AntisymmetricObjectProperty(inheresIn)
ObjectPropertyDomain(inheresIn ObjectUnionOf(ModeTS Mode))
ObjectPropertyRange(inheresIn ObjectUnionOf(TimeSlice IndividualConcept))
// Object property: example-4DA1.owl#mediates
SubObjectPropertyOf(mediates existentiallyDependentOf)
AntisymmetricObjectProperty(mediates)
ObjectPropertyDomain(mediates ObjectUnionOf(RelatorTS Relator))
ObjectPropertyRange(mediates ObjectUnionOf(TimeSlice IndividualConcept))

// Sub property chain axiom
SubObjectPropertyOf(**SubObjectPropertyChain**(objPropertyTS hasTemporalExtent) hasTemporalExtent)

// DOMAIN OBJECT PROPERTIES

// Object property: example-4DA1.owl#componentOfCerebroPessoa
SubObjectPropertyOf(componentOfCerebroPessoa invExistentiallyDependentOf)
SubObjectPropertyOf(componentOfCerebroPessoa componentOf)
SubObjectPropertyOf(componentOfCerebroPessoa existentiallyDependentOf)
SubObjectPropertyOf(componentOfCerebroPessoa objPropertyIC)
ObjectPropertyDomain(componentOfCerebroPessoa Cerebro)
ObjectPropertyRange(componentOfCerebroPessoa Pessoa)
// Object property: example-4DA1.owl#componentOfCoracaoPessoa
SubObjectPropertyOf(componentOfCoracaoPessoa existentiallyDependentOf)
SubObjectPropertyOf(componentOfCoracaoPessoa componentOf)
ObjectPropertyDomain(componentOfCoracaoPessoa CoracaoTS)
ObjectPropertyRange(componentOfCoracaoPessoa PessoaTS)
// Object property: example-4DA1.owl#casadoCom
// Object property: example-4DA1.owl#casadoComConjugeConjuge
SubObjectPropertyOf(casadoComConjugeConjuge casadoCom)
ObjectPropertyDomain(casadoComConjugeConjuge ConjugeTS)
ObjectPropertyRange(casadoComConjugeConjuge ConjugeTS)

// DOMAIN DATA PROPERTIES

// Data property: example-4DA1.owl#nomePessoa
DataPropertyDomain(nomePessoa Pessoa)
// Data property: example-4DA1.owl#apelidoPessoa
DataPropertyDomain(apelidoPessoa PessoaTS)
)
// Data property: example-4DA1.owl#idadePessoa
DataPropertyDomain(idadePessoa PessoaTS)
// Data property: example-4DA1.owl#cpfPessoa
DataPropertyDomain(cpfPessoa PessoaTS)

//---------- 4D-A2 EXAMPLE ----------//

Ontology(<example-4DA2.owl>
// Class: http://www.w3.org/2002/07/owl#Thing

// STRUCTURAL CLASSES

// Class: example-4DA2.owl#TemporalExtent
DisjointClasses(TemporalExtent IndividualConcept TimeSlice)
// Class: example-4DA2.owl#IndividualConcept
EquivalentClasses(IndividualConcept **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) TimeSlice))
EquivalentClasses(IndividualConcept **ObjectUnionOf**(Object Moment))
// Class: example-4DA2.owl#Object
EquivalentClasses(Object **ObjectUnionOf**(Collective FunctionalComplex Quantity))
SubClassOf(Object IndividualConcept)
DisjointClasses(Object Moment)
// Class: example-4DA2.owl#FunctionalComplex
EquivalentClasses(FunctionalComplex **ObjectUnionOf**(Cerebro Coracao Pessoa))
SubClassOf(FunctionalComplex Object)
SubClassOf(FunctionalComplex **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) FunctionalComplexTS))
DisjointClasses(FunctionalComplex Quantity Collective)
// Class: example-4DA2.owl#Collective
SubClassOf(Collective Object)
SubClassOf(Collective **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CollectiveTS))
// Class: example-4DA2.owl#Quantity
SubClassOf(Quantity Object)
SubClassOf(Quantity **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) QuantityTS))
// Class: example-4DA2.owl#Moment
EquivalentClasses(Moment **ObjectUnionOf**(Relator Mode))
SubClassOf(Moment IndividualConcept)
// Class: example-4DA2.owl#Mode
SubClassOf(Mode **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) ModeTS))
SubClassOf(Mode Moment)
DisjointClasses(Mode Relator)

// Class: example-4DA2.owl#Relator
EquivalentClasses(Relator **ObjectSomeValuesFrom**(mediates IndividualConcept))
EquivalentClasses(Relator Casamento)
SubClassOf(Relator Moment)
SubClassOf(Relator **ObjectMinCardinality**(2 mediates IndividualConcept))
SubClassOf(Relator **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) RelatorTS))

// Class: example-4DA2.owl#TimeSlice
EquivalentClasses(TimeSlice **ObjectExactCardinality**(1 timeSliceOf IndividualConcept))
EquivalentClasses(TimeSlice **ObjectUnionOf**(CollectiveTS FunctionalComplexTS ModeTS QuantityTS RelatorTS))
EquivalentClasses(TimeSlice **ObjectSomeValuesFrom**(hasTemporalExtent TemporalExtent))
// Class: example-4DA2.owl#FunctionalComplexTS
EquivalentClasses(FunctionalComplexTS **ObjectSomeValuesFrom**(timeSliceOf FunctionalComplex))
SubClassOf(FunctionalComplexTS TimeSlice)
// Class: example-4DA2.owl#CollectiveTS
EquivalentClasses(CollectiveTS **ObjectSomeValuesFrom**(timeSliceOf Collective))
SubClassOf(CollectiveTS TimeSlice)
// Class: example-4DA2.owl#QuantityTS
EquivalentClasses(QuantityTS **ObjectSomeValuesFrom**(timeSliceOf Quantity))
SubClassOf(QuantityTS TimeSlice)
// Class: example-4DA2.owl#ModeTS
EquivalentClasses(ModeTS **ObjectSomeValuesFrom**(timeSliceOf Mode))
SubClassOf(ModeTS TimeSlice)
DisjointClasses(ModeTS RelatorTS)
// Class: example-4DA2.owl#RelatorTS
EquivalentClasses(RelatorTS **ObjectSomeValuesFrom**(timeSliceOf Relator))
SubClassOf(RelatorTS TimeSlice)

// DOMAIN CLASSES

// Class: example-4DA2.owl#Pessoa
EquivalentClasses(Pessoa **ObjectUnionOf**(Mulher Homem))
SubClassOf(Pessoa **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) PessoaTS))
SubClassOf(Pessoa FunctionalComplex)
SubClassOf(Pessoa **ObjectMinCardinality**(1 **InverseObjectProperty**(componentOfCoracaoPessoa) Coracao))
SubClassOf(Pessoa **DataExactCardinality**(1 nomePessoa))
SubClassOf(Pessoa **ObjectExactCardinality**(1 **InverseObjectProperty**(componentOfCerebroPessoa) Cerebro))
DisjointClasses(Pessoa Cerebro Coracao)
// Class: example-4DA2.owl#Homem
SubClassOf(Homem **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) HomemTS))
SubClassOf(Homem Pessoa)
DisjointClasses(Homem Mulher)
// Class: example-4DA2.owl#Mulher
SubClassOf(Mulher **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) MulherTS))
SubClassOf(Mulher Pessoa)
// Class: example-4DA2.owl#OrgaoHumano
EquivalentClasses(OrgaoHumano **ObjectUnionOf**(Coracao Cerebro))
SubClassOf(OrgaoHumano **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) OrgaoHumanoTS))
// Class: example-4DA2.owl#Coracao
SubClassOf(Coracao **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CoracaoTS))
SubClassOf(Coracao **ObjectMaxCardinality**(1 componentOf Pessoa))
SubClassOf(Coracao FunctionalComplex)
SubClassOf(Coracao **ObjectExactCardinality**(1 componentOfCoracaoPessoa Pessoa))
// Class: example-4DA2.owl#Cerebro
SubClassOf(Cerebro **ObjectExactCardinality**(1 componentOfCerebroPessoa Pessoa))
SubClassOf(Cerebro **ObjectMaxCardinality**(1 componentOf Pessoa))
SubClassOf(Cerebro FunctionalComplex)
SubClassOf(Cerebro **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CerebroTS))
// Class: example-4DA2.owl#Casamento
EquivalentClasses(Casamento Relator)
SubClassOf(Casamento **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CasamentoTS))
SubClassOf(Casamento Relator)
SubClassOf(Casamento **ObjectExactCardinality**(2 mediates Pessoa))

// Class: example-4DA2.owl#PessoaTS
EquivalentClasses(PessoaTS **ObjectSomeValuesFrom**(timeSliceOf Pessoa))
EquivalentClasses(PessoaTS **ObjectUnionOf**(MulherTS HomemTS))
SubClassOf(PessoaTS FunctionalComplexTS)
SubClassOf(PessoaTS **DataMaxCardinality**(1 cpfPessoa))
SubClassOf(PessoaTS **DataExactCardinality**(1 idadePessoa))
DisjointClasses(PessoaTS CoracaoTS CerebroTS)
// Class: example-4DA2.owl#HomemTS
EquivalentClasses(HomemTS **ObjectSomeValuesFrom**(timeSliceOf Homem))
SubClassOf(HomemTS PessoaTS) **DisjointClasses**(HomemTS MulherTS)
// Class: example-4DA2.owl#MulherTS
EquivalentClasses(MulherTS **ObjectSomeValuesFrom**(timeSliceOf Mulher))
SubClassOf(MulherTS PessoaTS)
// Class: example-4DA2.owl#ConjugeTS
EquivalentClasses(ConjugeTS **ObjectUnionOf**(EsposoTS EsposaTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1 casadoComConjugeConjuge ConjugeTS))
SubClassOf(ConjugeTS **ObjectExactCardinality**(1 **InverseObjectProperty**(casadoComConjugeConjuge) ConjugeTS))
SubClassOf(ConjugeTS PessoaTS)
// Class: example-4DA2.owl#EsposoTS
SubClassOf(EsposoTS ConjugeTS) **SubClassOf**(EsposoTS HomemTS)
DisjointClasses(EsposoTS EsposaTS)
// Class: example-4DA2.owl#EsposaTS
SubClassOf(EsposaTS ConjugeTS) **SubClassOf**(EsposaTS MulherTS)
// Class: example-4DA2.owl#OrgaoHumanoTS
EquivalentClasses(OrgaoHumanoTS **ObjectSomeValuesFrom**(timeSliceOf OrgaoHumano))
EquivalentClasses(OrgaoHumanoTS **ObjectUnionOf**(CoracaoTS CerebroTS))
// Class: example-4DA2.owl#CerebroTS
EquivalentClasses(CerebroTS **ObjectSomeValuesFrom**(timeSliceOf Cerebro))
SubClassOf(CerebroTS FunctionalComplexTS)
// Class: example-4DA2.owl#CoracaoTS
EquivalentClasses(CoracaoTS **ObjectSomeValuesFrom**(timeSliceOf Coracao))
SubClassOf(CoracaoTS FunctionalComplexTS)
// Class: example-4DA2.owl#CasamentoTS
EquivalentClasses(CasamentoTS **ObjectSomeValuesFrom**(timeSliceOf Casamento))
SubClassOf(CasamentoTS RelatorTS)

// STRUCTURAL OBJECT PROPERTIES

// Object property: example-4DA2.owl#timeSliceOf
FunctionalObjectProperty(timeSliceOf)
ObjectPropertyDomain(timeSliceOf TimeSlice)
ObjectPropertyRange(timeSliceOf IndividualConcept)
// Object property: example-4DA2.owl#objPropertyIC
SymmetricObjectProperty(objPropertyIC)
ObjectPropertyDomain(objPropertyIC IndividualConcept)
ObjectPropertyRange(objPropertyIC IndividualConcept)
// Object property: example-4DA2.owl#objPropertyTS
SymmetricObjectProperty(objPropertyTS)
ObjectPropertyDomain(objPropertyTS TimeSlice)
ObjectPropertyRange(objPropertyTS TimeSlice)
// Object property: example-4DA2.owl#hasTemporalExtent
ObjectPropertyDomain(hasTemporalExtent TimeSlice)
ObjectPropertyRange(hasTemporalExtent TemporalExtent)
// Object property: example-4DA2.owl#partOf
AntisymmetricObjectProperty(partOf)
IrreflexiveObjectProperty(partOf)
ObjectPropertyDomain(partOf IndividualConcept)
ObjectPropertyRange(partOf IndividualConcept)
// Object property: example-4DA2.owl#essentialPartOf
SubObjectPropertyOf(essentialPartOf partOf)
SubObjectPropertyOf(essentialPartOf invExistentiallyDependentOf)
// Object property: example-4DA2.owl#inseparablePartOf
SubObjectPropertyOf(inseparablePartOf partOf)
SubObjectPropertyOf(inseparablePartOf existentiallyDependentOf)
// Object property: example-4DA2.owl#componentOf
SubObjectPropertyOf(componentOf partOf)
ObjectPropertyDomain(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))
ObjectPropertyRange(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))
// Object property: example-4DA2.owl#subCollectionOf
SubObjectPropertyOf(subCollectionOf partOf)
ObjectPropertyDomain(subCollectionOf **ObjectUnionOf**(CollectiveTS Collective))
ObjectPropertyRange(subCollectionOf **ObjectUnionOf**(CollectiveTS Collective))
// Object property: example-4DA2.owl#memberOf
SubObjectPropertyOf(memberOf partOf)
ObjectPropertyDomain(memberOf **ObjectUnionOf**(Collective CollectiveTS FunctionalComplex FunctionalComplexTS))
ObjectPropertyRange(memberOf **ObjectUnionOf**(CollectiveTS Collective))
// Object property: example-4DA2.owl#subQuantityOf
SubObjectPropertyOf(subQuantityOf essentialPartOf)
ObjectPropertyDomain(subQuantityOf **ObjectUnionOf**(QuantityTS Quantity))
ObjectPropertyRange(subQuantityOf **ObjectUnionOf**(QuantityTS Quantity))
// Object property: example-4DA2.owl#existentiallyDependentOf
SubObjectPropertyOf(existentiallyDependentOf objPropertyIC)
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
IrreflexiveObjectProperty(existentiallyDependentOf)
// Object property: example-4DA2.owl#invExistentiallyDependentOf
SubObjectPropertyOf(invExistentiallyDependentOf objPropertyIC)
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
// Object property: example-4DA2.owl#inheresIn
SubObjectPropertyOf(inheresIn existentiallyDependentOf)
FunctionalObjectProperty(inheresIn)
AntisymmetricObjectProperty(inheresIn)
ObjectPropertyDomain(inheresIn Mode)
ObjectPropertyRange(inheresIn IndividualConcept)
// Object property: example-4DA2.owl#mediates
SubObjectPropertyOf(mediates existentiallyDependentOf)
AntisymmetricObjectProperty(mediates)
ObjectPropertyDomain(mediates Relator)
ObjectPropertyRange(mediates IndividualConcept)

// Sub property chain axiom
SubObjectPropertyOf(SubObjectPropertyChain(objPropertyTS hasTemporalExtent) hasTemporalExtent)

// DOMAIN OBJECT PROPERTIES

// Object property: example-4DA2.owl#componentOfCerebroPessoa
SubObjectPropertyOf(componentOfCerebroPessoa invExistentiallyDependentOf)
SubObjectPropertyOf(componentOfCerebroPessoa componentOf)
SubObjectPropertyOf(componentOfCerebroPessoa existentiallyDependentOf)
SubObjectPropertyOf(componentOfCerebroPessoa objPropertyIC)
ObjectPropertyDomain(componentOfCerebroPessoa Cerebro)
ObjectPropertyRange(componentOfCerebroPessoa Pessoa)
// Object property: example-4DA2.owl#casadoCom
// Object property: example-4DA2.owl#casadoComConjugeConjuge
SubObjectPropertyOf(casadoComConjugeConjuge casadoCom)
ObjectPropertyDomain(casadoComConjugeConjuge ConjugeTS)
ObjectPropertyRange(casadoComConjugeConjuge ConjugeTS)
// Object property: example-4DA2.owl#componentOfCoracaoPessoa
SubObjectPropertyOf(componentOfCoracaoPessoa existentiallyDependentOf)
SubObjectPropertyOf(componentOfCoracaoPessoa componentOf)
SubObjectPropertyOf(componentOfCoracaoPessoa objPropertyIC)
ObjectPropertyDomain(componentOfCoracaoPessoa Coracao)
ObjectPropertyRange(componentOfCoracaoPessoa Pessoa)

// DOMAIN DATA PROPERTIES

// Data property: example-4DA2.owl#nomePessoa
DataPropertyDomain(nomePessoa Pessoa)
// Data property: example-4DA2.owl#apelidoPessoa
DataPropertyDomain(apelidoPessoa PessoaTS)
// Data property: example-4DA2.owl#idadePessoa
DataPropertyDomain(idadePessoa PessoaTS)
// Data property: example-4DA2.owl#cpfPessoa
DataPropertyDomain(cpfPessoa PessoaTS)
)

//---------- REIFICATION EXAMPLE ----------//
Ontology(<example-Reification.owl>
// Class: http://www.w3.org/2002/07/owl#Thing

// STRUCTURAL CLASSES

// Class: example-Reification.owl#Individual
EquivalentClasses(Individual **ObjectUnionOf**(Object Moment))
SubClassOf(Individual **ObjectSomeValuesFrom**(hasTemporalExtent TemporalExtent))
DisjointClasses(Individual TemporalExtent)
// Class: example-Reification.owl#TemporalExtent
DisjointClasses(TemporalExtent Individual)
// Class: example-Reification.owl#Object
EquivalentClasses(Object **ObjectUnionOf**(Collective FunctionalComplex Quantity))
SubClassOf(Object **ObjectSomeValuesFrom**(**InverseObjectProperty**(inheresIn) Moment))
SubClassOf(Object Individual)
DisjointClasses(Object Moment)
// Class: example-Reification.owl#FunctionalComplex
EquivalentClasses(FunctionalComplex **ObjectUnionOf**(Cerebro Coracao Pessoa))
SubClassOf(FunctionalComplex Object)
DisjointClasses(FunctionalComplex Quantity Collective)
// Class: example-Reification.owl#Collective
SubClassOf(Collective Object)
// Class: example-Reification.owl#Quantity
SubClassOf(Quantity Object)
// Class: example-Reification.owl#Moment
EquivalentClasses(Moment **ObjectUnionOf**(Relator IntrinsicMoment))
SubClassOf(Moment Individual)
// Class: example-Reification.owl#IntrinsicMoment
EquivalentClasses(IntrinsicMoment **ObjectSomeValuesFrom**(inheresIn Individual))

EquivalentClasses(IntrinsicMoment **ObjectUnionOf**(Quality Mode))
SubClassOf(IntrinsicMoment Moment)
DisjointClasses(IntrinsicMoment Relator)
// Class: example-Reification.owl#Mode
EquivalentClasses(Mode QuaIndividual)
SubClassOf(Mode IntrinsicMoment)
DisjointClasses(Mode Quality)
// Class: example-Reification.owl#QuaIndividual
EquivalentClasses(QuaIndividual Mode)
EquivalentClasses(QuaIndividual **ObjectUnionOf**(RelationalQuaIndividual PhasedQuaIndividual))
SubClassOf(QuaIndividual Mode)
// Class: example-Reification.owl#PhasedQuaIndividual
SubClassOf(PhasedQuaIndividual QuaIndividual)
DisjointClasses(PhasedQuaIndividual RelationalQuaIndividual)

// DOMAIN CLASSES

// Class: example-Reification.owl#Pessoa
EquivalentClasses(Pessoa **ObjectUnionOf**(Mulher Homem))
SubClassOf(Pessoa **ObjectMinCardinality**(1 **InverseObjectProperty**(inheresIn) idadePessoa))
SubClassOf(Pessoa **ObjectExactCardinality**(1 **InverseObjectProperty**(inheresIn) nomePessoa))
SubClassOf(Pessoa FunctionalComplex)
SubClassOf(Pessoa **ObjectMinCardinality**(1 **InverseObjectProperty**(componentOfCoracaoPessoa) Coracao))
SubClassOf(Pessoa **ObjectMaxCardinality**(1 **InverseObjectProperty**(inheresIn) cpfPessoa))
SubClassOf(Pessoa **ObjectExactCardinality**(1 **InverseObjectProperty**(componentOfCerebroPessoa) Cerebro))
DisjointClasses(Pessoa Cerebro Coracao)
// Class: example-Reification.owl#Homem
SubClassOf(Homem Pessoa)
DisjointClasses(Homem Mulher)
// Class: example-Reification.owl#Mulher
SubClassOf(Mulher Pessoa)
// Class: example-Reification.owl#OrgaoHumano
EquivalentClasses(OrgaoHumano **ObjectUnionOf**(Coracao Cerebro))
// Class: example-Reification.owl#Coracao
SubClassOf(Coracao **ObjectMaxCardinality**(1 componentOf Pessoa))
SubClassOf(Coracao FunctionalComplex)
SubClassOf(Coracao **ObjectExactCardinality**(1 componentOfCoracaoPessoa Pessoa))
// Class: example-Reification.owl#Cerebro
SubClassOf(Cerebro **ObjectExactCardinality**(1 componentOfCerebroPessoa Pessoa))
SubClassOf(Cerebro **ObjectMaxCardinality**(1 componentOf Pessoa))
SubClassOf(Cerebro FunctionalComplex)
// Class: example-Reification.owl#Casamento
EquivalentClasses(Casamento Relator)
SubClassOf(Casamento **ObjectExactCardinality**(2 **InverseObjectProperty**(partOfRelator) QuaConjuge))
SubClassOf(Casamento Relator)
SubClassOf(Casamento **ObjectExactCardinality**(2 mediates Pessoa))
// Class: example-Reification.owl#QuaConjuge
SubClassOf(QuaConjuge **ObjectMinCardinality**(1 partOfRelator Casamento))
SubClassOf(QuaConjuge **RelationalQuaIndividual**)
SubClassOf(QuaConjuge **ObjectExactCardinality**(1 existentiallyDependentOfQua QuaConjuge))
SubClassOf(QuaConjuge **ObjectSomeValuesFrom**(inheresIn Pessoa))
// Class: example-Reification.owl#QuaEsposo
SubClassOf(QuaEsposo **ObjectSomeValuesFrom**(existentiallyDependentOf QuaConjuge))
SubClassOf(QuaEsposo **ObjectSomeValuesFrom**(inheresIn Homem))
SubClassOf(QuaEsposo RelationalQuaIndividual)
DisjointClasses(QuaEsposo QuaEsposa)
// Class: example-Reification.owl#RelationalQuaIndividual
EquivalentClasses(RelationalQuaIndividual **ObjectSomeValuesFrom**(partOfRelator Relator))
SubClassOf(RelationalQuaIndividual QuaIndividual)
// Class: example-Reification.owl#Quality
SubClassOf(Quality IntrinsicMoment)
DisjointClasses(Quality Mode)
// Class: example-Reification.owl#Relator
EquivalentClasses(Relator Casamento)
EquivalentClasses(Relator **ObjectSomeValuesFrom**(mediates Individual))
SubClassOf(Relator Moment)
SubClassOf(Relator **ObjectMinCardinality**(2 mediates Individual))
SubClassOf(Relator **ObjectMinCardinality**(2 **InverseObjectProperty**(partOfRelator) RelationalQuaIndividual))
DisjointClasses(Relator IntrinsicMoment)

// Class: example-Reification.owl#QuaEsposa
SubClassOf(QuaEsposa **ObjectSomeValuesFrom**(existentiallyDependentOf QuaConjuge))
SubClassOf(QuaEsposa RelationalQuaIndividual)
SubClassOf(QuaEsposa **ObjectSomeValuesFrom**(inheresIn Mulher))

// Class: example-Reification.owl#nomePessoa
SubClassOf(nomePessoa Quality)
SubClassOf(nomePessoa **ObjectSomeValuesFrom**(inheresIn Pessoa))
// Class: example-Reification.owl#cpfPessoa
SubClassOf(cpfPessoa **ObjectSomeValuesFrom**(inheresIn Pessoa))
SubClassOf(cpfPessoa Quality)
// Class: example-Reification.owl#idadePessoa
SubClassOf(idadePessoa **ObjectSomeValuesFrom**(inheresIn Pessoa))
SubClassOf(idadePessoa Quality)
// Class: example-Reification.owl#apelidoPessoa
SubClassOf(apelidoPessoa Quality)
SubClassOf(apelidoPessoa **ObjectSomeValuesFrom**(inheresIn Pessoa))

// STRUCTURAL OBJECT PROPERTIES

// Object property: example-Reification.owl#hasTemporalExtent
FunctionalObjectProperty(hasTemporalExtent)
ObjectPropertyDomain(hasTemporalExtent Individual)
ObjectPropertyRange(hasTemporalExtent TemporalExtent)
// Object property: example-Reification.owl#partOf
AntisymmetricObjectProperty(partOf)
IrreflexiveObjectProperty(partOf)
ObjectPropertyDomain(partOf Individual)
ObjectPropertyRange(partOf Individual)
// Object property: example-Reification.owl#essentialPartOf
SubObjectPropertyOf(essentialPartOf partOf)
SubObjectPropertyOf(essentialPartOf invExistentiallyDependentOf)
// Object property: example-Reification.owl#inseparablePartOf
SubObjectPropertyOf(inseparablePartOf partOf)
SubObjectPropertyOf(inseparablePartOf existentiallyDependentOf)
// Object property: example-Reification.owl#componentOf
SubObjectPropertyOf(componentOf partOf)
ObjectPropertyDomain(componentOf FunctionalComplex)
ObjectPropertyRange(componentOf FunctionalComplex)
// Object property: example-Reification.owl#subCollectionOf
SubObjectPropertyOf(subCollectionOf partOf)
ObjectPropertyDomain(subCollectionOf Collective)
ObjectPropertyRange(subCollectionOf Collective)
// Object property: example-Reification.owl#memberOf
SubObjectPropertyOf(memberOf partOf)
ObjectPropertyDomain(memberOf **ObjectUnionOf**(FunctionalComplex Collective))
ObjectPropertyRange(memberOf Collective)
// Object property: example-Reification.owl#subQuantityOf
SubObjectPropertyOf(subQuantityOf essentialPartOf)
ObjectPropertyDomain(subQuantityOf Quantity)
ObjectPropertyRange(subQuantityOf Quantity)
// Object property: example-Reification.owl#partOfRelator
SubObjectPropertyOf(partOfRelator essentialPartOf)
SubObjectPropertyOf(partOfRelator inseparablePartOf)
FunctionalObjectProperty(partOfRelator)
ObjectPropertyDomain(partOfRelator RelationalQuaIndividual)
ObjectPropertyRange(partOfRelator Relator)
// Object property: example-Reification.owl#existentiallyDependentOf
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
IrreflexiveObjectProperty(existentiallyDependentOf)
ObjectPropertyDomain(existentiallyDependentOf Individual)
ObjectPropertyRange(existentiallyDependentOf Individual)
// Object property: example-Reification.owl#invExistentiallyDependentOf
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
ObjectPropertyDomain(invExistentiallyDependentOf Individual)
ObjectPropertyRange(invExistentiallyDependentOf Individual)
// Object property: example-Reification.owl#inheresIn
SubObjectPropertyOf(inheresIn existentiallyDependentOf)
FunctionalObjectProperty(inheresIn)
AntisymmetricObjectProperty(inheresIn)
ObjectPropertyDomain(inheresIn IntrinsicMoment)
ObjectPropertyRange(inheresIn Individual)

// Object property: example-Reification.owl#mediates
SubObjectPropertyOf(mediates existentiallyDependentOf)
AntisymmetricObjectProperty(mediates)
ObjectPropertyDomain(mediates Relator)
ObjectPropertyRange(mediates Individual)

// DOMAIN OBJECT PROPERTIES

// Object property: example-Reification.owl#componentOfCerebroPessoa
SubObjectPropertyOf(componentOfCerebroPessoa invExistentiallyDependentOf)
SubObjectPropertyOf(componentOfCerebroPessoa componentOf)
SubObjectPropertyOf(componentOfCerebroPessoa existentiallyDependentOf)
ObjectPropertyDomain(componentOfCerebroPessoa Cerebro)
ObjectPropertyRange(componentOfCerebroPessoa Pessoa)

// STRUCTURAL DATA PROPERTIES

// Data property: example-Reification.owl#hasValue
FunctionalDataProperty(hasValue)
DataPropertyDomain(hasValue Quality)
)

// Object property: example-Reification.owl#existentiallyDependentOfQua
SubObjectPropertyOf(existentiallyDependentOfQua existentiallyDependentOf)
SymmetricObjectProperty(existentiallyDependentOfQua)
ObjectPropertyDomain(existentiallyDependentOfQua RelationalQuaIndividual)
ObjectPropertyRange(existentiallyDependentOfQua RelationalQuaIndividual)

// Object property: example-Reification.owl#componentOfCoracaoPessoa
SubObjectPropertyOf(componentOfCoracaoPessoa existentiallyDependentOf)
SubObjectPropertyOf(componentOfCoracaoPessoa componentOf)
ObjectPropertyDomain(componentOfCoracaoPessoa Coracao)
ObjectPropertyRange(componentOfCoracaoPessoa Pessoa)

APÊNDICE III – Ontologia OWL de ECG segundo a abordagem 4D-AO

Ontology(<ECG.owl>
// Class: http://www.w3.org/2002/07/owl#Thing

// STRUCTURAL CLASSES

// Class: ECG.owl#TemporalExtent
DisjointClasses(IndividualConcept TimeSlice TemporalExtent)
// Class: ECG.owl#IndividualConcept
EquivalentClasses(IndividualConcept **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) TimeSlice))
EquivalentClasses(IndividualConcept **ObjectUnionOf**(Object Moment))
// Class: ECG.owl#FunctionalComplex
EquivalentClasses(FunctionalComplex **ObjectUnionOf**(Brain CardiacElectricalImpulse ECGRecord Heart Person))
EquivalentClasses(FunctionalComplex **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) FunctionalComplexTS))
SubClassOf(FunctionalComplex Object)
SubClassOf(FunctionalComplex **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) FunctionalComplexTS))
DisjointClasses(FunctionalComplex Collective Quantity)
// Class: ECG.owl#Object
EquivalentClasses(Object **ObjectUnionOf**(Collective FunctionalComplex Quantity))
SubClassOf(Object IndividualConcept)
DisjointClasses(Object Moment)
// Class: ECG.owl#Collective
SubClassOf(Collective Object)
SubClassOf(Collective **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) CollectiveTS))
// Class: ECG.owl#Quantity
SubClassOf(Quantity **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) QuantityTS))
SubClassOf(Quantity Object)
// Class: ECG.owl#Mode
EquivalentClasses(Mode **ObjectUnionOf**(CompleteCycle Cycle PartialCycle))
SubClassOf(Mode **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) ModeTS))
SubClassOf(Mode Moment)
DisjointClasses(Mode Relator)
// Class: ECG.owl#Relator
EquivalentClasses(Relator **ObjectUnionOf**(PartialHeartContraction CompleteHeartContraction))
SubClassOf(Relator **ObjectAllValuesFrom**(**InverseObjectProperty**(timeSliceOf) RelatorTS))
SubClassOf(Relator Moment)

// Class: ECG.owl#TimeSlice
EquivalentClasses(TimeSlice **ObjectExactCardinality**(1 timeSliceOf IndividualConcept))
EquivalentClasses(TimeSlice **ObjectSomeValuesFrom**(hasTemporalExtent TemporalExtent))
EquivalentClasses(TimeSlice **ObjectUnionOf**(CollectiveTS FunctionalComplexTS ModeTS QuantityTS RelatorTS))
// Class: ECG.owl#FunctionalComplexTS
EquivalentClasses(FunctionalComplexTS **ObjectSomeValuesFrom**(timeSliceOf FunctionalComplex))
SubClassOf(FunctionalComplexTS TimeSlice)
// Class: ECG.owl#CollectiveTS
EquivalentClasses(CollectiveTS **ObjectSomeValuesFrom**(timeSliceOf Collective))
SubClassOf(CollectiveTS TimeSlice)
// Class: ECG.owl#QuantityTS
EquivalentClasses(QuantityTS **ObjectSomeValuesFrom**(timeSliceOf Quantity))
SubClassOf(QuantityTS TimeSlice)
// Class: ECG.owl#ModeTS
EquivalentClasses(ModeTS **ObjectSomeValuesFrom**(timeSliceOf Mode))
SubClassOf(ModeTS IntrinsicMomentTS)
DisjointClasses(ModeTS RelatorTS)
// Class: ECG.owl#RelatorTS
EquivalentClasses(RelatorTS **ObjectSomeValuesFrom**(mediates TimeSlice))
EquivalentClasses(RelatorTS **ObjectSomeValuesFrom**(timeSliceOf Relator))
SubClassOf(RelatorTS TimeSlice)
SubClassOf(RelatorTS **ObjectMinCardinality**(2 mediates TimeSlice))

// DOMAIN CLASSES

// Class: ECG.owl#Brain
EquivalentClasses(Brain **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) BrainTS))
SubClassOf(Brain FunctionalComplex)
DisjointClasses(Brain Heart ECGRecord CardiacElectricalImpulse Person)
// Class: ECG.owl#Heart
EquivalentClasses(Heart **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) HeartTS))
SubClassOf(Heart FunctionalComplex)
// Class: ECG.owl#ECGRecord
EquivalentClasses(ECGRecord **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) ECGRecordTS))
SubClassOf(ECGRecord FunctionalComplex)
// Class: ECG.owl#Person
EquivalentClasses(Person **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) PersonTS))
SubClassOf(Person FunctionalComplex)
// Class: ECG.owl#CardiacElectricalImpulse
EquivalentClasses(CardiacElectricalImpulse
ObjectSomeValuesFrom(**InverseObjectProperty**(timeSliceOf) CardiacElectricalImpulseTS))
SubClassOf(CardiacElectricalImpulse FunctionalComplex)
// Class: ECG.owl#CompleteCardiacElectricalImpulse
EquivalentClasses(CompleteCardiacElectricalImpulse
ObjectSomeValuesFrom(**InverseObjectProperty**(timeSliceOf) CompleteCardiacElectricalImpulseTS))
SubClassOf(CompleteCardiacElectricalImpulse CardiacElectricalImpulse)
DisjointClasses(CompleteCardiacElectricalImpulse PartialCardiacElectricalImpulse)
// Class: ECG.owl#PartialCardiacElectricalImpulse
EquivalentClasses(PartialCardiacElectricalImpulse
ObjectSomeValuesFrom(**InverseObjectProperty**(timeSliceOf) PartialCardiacElectricalImpulseTS))
SubClassOf(PartialCardiacElectricalImpulse CardiacElectricalImpulse)
// Class: ECG.owl#Cycle
EquivalentClasses(Cycle **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) CycleTS))
SubClassOf(Cycle Mode)
// Class: ECG.owl#CompleteCycle
EquivalentClasses(CompleteCycle **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) CompleteCycleTS))
SubClassOf(CompleteCycle Mode)
SubClassOf(CompleteCycle Cycle)
DisjointClasses(CompleteCycle PartialCycle)

// Class: ECG.owl#PartialCycle
EquivalentClasses(PartialCycle **ObjectSomeValuesFrom**(**InverseObjectProperty**(timeSliceOf) PartialCycleTS))
SubClassOf(PartialCycle Mode)
SubClassOf(PartialCycle Cycle)
// Class: ECG.owl#CompleteHeartContraction
EquivalentClasses(CompleteHeartContraction
ObjectSomeValuesFrom(**InverseObjectProperty**(timeSliceOf) CompleteHeartContractionTS))
SubClassOf(CompleteHeartContraction Relator)
// Class: ECG.owl#PartialHeartContraction
EquivalentClasses(PartialHeartContraction
ObjectSomeValuesFrom(**InverseObjectProperty**(timeSliceOf) PartialHeartContractionTS))
SubClassOf(PartialHeartContraction Relator)

// Class: ECG.owl#BrainTS
EquivalentClasses(BrainTS **ObjectExactCardinality**(1 timeSliceOf Brain))
SubClassOf(BrainTS FunctionalComplexTS)
SubClassOf(BrainTS **ObjectExactCardinality**(1 componentOfBrainPerson PersonTS))
DisjointClasses(BrainTS PersonTS HeartTS ECGRecordTS CardiacElectricalImpulseTS)
// Class: ECG.owl#HeartTS
EquivalentClasses(HeartTS **ObjectExactCardinality**(1 timeSliceOf Heart))
SubClassOf(HeartTS **ObjectExactCardinality**(1 componentOfHeartPerson PersonTS))
SubClassOf(HeartTS FunctionalComplexTS)
// Class: ECG.owl#CompletelyFunctioningHeartTS
EquivalentClasses(CompletelyFunctioningHeartTS **ObjectSomeValuesFrom**(componentOfHeartPerson
ObjectIntersectionOf(**ObjectSomeValuesFrom**(**InverseObjectProperty**(belongsTo)
ObjectIntersectionOf(**ObjectSomeValuesFrom**(**InverseObjectProperty**(inheresIn) CompleteCycleTS)
ECGRecordTS)) PersonTS)))
SubClassOf(CompletelyFunctioningHeartTS **ObjectExactCardinality**(1
InverseObjectProperty(mediates) CompleteHeartContractionTS))
SubClassOf(CompletelyFunctioningHeartTS ObjectExactCardinality(1
InverseObjectProperty(contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart)
CompleteCardiacElectricalImpulseTS))
SubClassOf(CompletelyFunctioningHeartTS HeartTS)
// Class: ECG.owl#PartiallyFunctioningHeartTS

EquivalentClasses(PartiallyFunctioningHeartTS
ObjectIntersectionOf(**ObjectSomeValuesFrom**(**InverseObjectProperty**(contracts)
ObjectIntersectionOf(**ObjectSomeValuesFrom**(**InverseObjectProperty**(maps) PartialCycleTS)
PartialCardiacElectricalImpulseTS)) HeartTS))
SubClassOf(PartiallyFunctioningHeartTS HeartTS)
SubClassOf(PartiallyFunctioningHeartTS **ObjectExactCardinality**(1
InverseObjectProperty(contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart)
PartialCardiacElectricalImpulseTS))
SubClassOf(PartiallyFunctioningHeartTS **ObjectExactCardinality**(1 **InverseObjectProperty**(mediates)
PartialHeartContractionTS))
// Class: ECG.owl#ECGRecordTS
EquivalentClasses(ECGRecordTS **ObjectExactCardinality**(1 timeSliceOf ECGRecord))
SubClassOf(ECGRecordTS FunctionalComplexTS)
SubClassOf(ECGRecordTS **ObjectExactCardinality**(1 **InverseObjectProperty**(inheresIn) CycleTS))
SubClassOf(ECGRecordTS **ObjectExactCardinality**(1 belongsToECGRecordPerson PersonTS))
// Class: ECG.owl#PersonTS
EquivalentClasses(PersonTS **ObjectExactCardinality**(1 timeSliceOf Person))
SubClassOf(PersonTS FunctionalComplexTS)
SubClassOf(PersonTS ObjectExactCardinality(1 **InverseObjectProperty**(componentOfHeartPerson)
HeartTS))
SubClassOf(PersonTS ObjectExactCardinality(1 **InverseObjectProperty**(componentOfBrainPerson)
BrainTS))
SubClassOf(PersonTS ObjectExactCardinality(1 **InverseObjectProperty**(belongsToECGRecordPerson)
ECGRecordTS))
// Class: ECG.owl#CardiacElectricalImpulseTS
EquivalentClasses(CardiacElectricalImpulseTS **ObjectExactCardinality**(1 timeSliceOf
CardiacElectricalImpulse))
SubClassOf(CardiacElectricalImpulseTS **ObjectExactCardinality**(1
InverseObjectProperty(mapsCycleCardiacElectricalImpulse) CycleTS))
SubClassOf(CardiacElectricalImpulseTS FunctionalComplexTS)
// Class: ECG.owl#PartialCardiacElectricalImpulseTS
EquivalentClasses(PartialCardiacElectricalImpulseTS **ObjectExactCardinality**(1 timeSliceOf
PartialCardiacElectricalImpulse))
SubClassOf(PartialCardiacElectricalImpulseTS **ObjectExactCardinality**(1
InverseObjectProperty(mapsPartialCyclePartialCardiacElectricalImpulse) PartialCycleTS))
SubClassOf(PartialCardiacElectricalImpulseTS CardiacElectricalImpulseTS)
SubClassOf(PartialCardiacElectricalImpulseTS **ObjectExactCardinality**(1
contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart PartiallyFunctioningHeartTS))
SubClassOf(PartialCardiacElectricalImpulseTS **ObjectExactCardinality**(1
InverseObjectProperty(mediates) PartialHeartContractionTS))
DisjointClasses(PartialCardiacElectricalImpulseTS CompleteCardiacElectricalImpulseTS)
// Class: ECG.owl#CompleteCardiacElectricalImpulseTS
EquivalentClasses(CompleteCardiacElectricalImpulseTS **ObjectExactCardinality**(1 timeSliceOf
CompleteCardiacElectricalImpulse))
SubClassOf(CompleteCardiacElectricalImpulseTS **ObjectExactCardinality**(1
InverseObjectProperty(mediates) CompleteHeartContractionTS))
SubClassOf(CompleteCardiacElectricalImpulseTS **ObjectExactCardinality**(1
InverseObjectProperty(mapsCompleteCycleCompleteCardiacElectricalImpulse) CompleteCycleTS))
SubClassOf(CompleteCardiacElectricalImpulseTS **ObjectExactCardinality**(1
contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart CompletelyFunctioningHeartTS))
SubClassOf(CompleteCardiacElectricalImpulseTS CardiacElectricalImpulseTS)
// Class: ECG.owl#CycleTS
EquivalentClasses(CycleTS **ObjectExactCardinality**(1 timeSliceOf Cycle))
SubClassOf(CycleTS **ObjectExactCardinality**(1 mapsCycleCardiacElectricalImpulse
CardiacElectricalImpulseTS))
SubClassOf(CycleTS ModeTS)
SubClassOf(CycleTS ObjectExactCardinality(1 inheresIn ECGRecordTS))
// Class: ECG.owl#PartialCycleTS
EquivalentClasses(PartialCycleTS **ObjectIntersectionOf**(**ObjectSomeValuesFrom**(maps
ObjectIntersectionOf(**ObjectSomeValuesFrom**(contracts HeartTS) PartialCardiacElectricalImpulseTS))
CycleTS))
EquivalentClasses(PartialCycleTS **ObjectExactCardinality**(1 timeSliceOf PartialCycle))
SubClassOf(PartialCycleTS ModeTS)
SubClassOf(PartialCycleTS CycleTS)
SubClassOf(PartialCycleTS **ObjectExactCardinality**(1 mapsPartialCyclePartialCardiacElectricalImpulse
PartialCardiacElectricalImpulseTS))
DisjointClasses(PartialCycleTS CompleteCycleTS)
// Class: ECG.owl#CompleteCycleTS
EquivalentClasses(CompleteCycleTS **ObjectExactCardinality**(1 timeSliceOf CompleteCycle))
SubClassOf(CompleteCycleTS CycleTS)
SubClassOf(CompleteCycleTS ModeTS)
SubClassOf(CompleteCycleTS **ObjectExactCardinality**(1
mapsCompleteCycleCompleteCardiacElectricalImpulse CompleteCardiacElectricalImpulseTS))
// Class: ECG.owl#PartialHeartContractionTS
EquivalentClasses(PartialHeartContractionTS **ObjectExactCardinality**(1 timeSliceOf
PartialHeartContraction))

SubClassOf(PartialHeartContractionTS RelatorTS)
SubClassOf(PartialHeartContractionTS **ObjectExactCardinality**(1 mediates PartialCardiacElectricalImpulseTS))

// STRUCTURAL OBJECT PROPERTIES

// Object property: example-4DA0.owl#timeSliceOf
FunctionalObjectProperty(timeSliceOf)
ObjectPropertyDomain(timeSliceOf TimeSlice)
ObjectPropertyRange(timeSliceOf IndividualConcept)
// Object property: example-4DA0.owl#hasTemporalExtent
ObjectPropertyDomain(hasTemporalExtent TimeSlice)
ObjectPropertyRange(hasTemporalExtent TemporalExtent)
// Object property: example-4DA0.owl#objPropertyTS
SymmetricObjectProperty(objPropertyTS)
ObjectPropertyDomain(objPropertyTS TimeSlice)
ObjectPropertyRange(objPropertyTS TimeSlice)
// Object property: example-4DA0.owl#partOf
SubObjectPropertyOf(partOf objPropertyTS)
AntisymmetricObjectProperty(partOf)
IrreflexiveObjectProperty(partOf)
ObjectPropertyDomain(partOf TimeSlice)
ObjectPropertyRange(partOf TimeSlice)
// Object property: example-4DA0.owl#essentialPartOf
SubObjectPropertyOf(essentialPartOf partOf)
SubObjectPropertyOf(essentialPartOf invExistentiallyDependentOf)
// Object property: example-4DA0.owl#inseparablePartOf
SubObjectPropertyOf(inseparablePartOf partOf)
SubObjectPropertyOf(inseparablePartOf existentiallyDependentOf)
// Object property: example-4DA0.owl#componentOf
SubObjectPropertyOf(componentOf partOf)
ObjectPropertyDomain(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))
ObjectPropertyRange(componentOf **ObjectUnionOf**(FunctionalComplexTS FunctionalComplex))
// Object property: example-4DA0.owl#memberOf
SubObjectPropertyOf(memberOf partOf)
ObjectPropertyDomain(memberOf **ObjectUnionOf**(Collective CollectiveTS FunctionalComplex FunctionalComplexTS))
SubClassOf(PartialHeartContractionTS **ObjectExactCardinality**(1 mediates PartiallyFunctioningHeartTS))

ObjectPropertyRange(memberOf **ObjectUnionOf**(CollectiveTS Collective))
// Object property: example-4DA0.owl#subCollectionOf
SubObjectPropertyOf(subCollectionOf partOf)
ObjectPropertyDomain(subCollectionOf ObjectUnionOf(CollectiveTS Collective))
ObjectPropertyRange(subCollectionOf ObjectUnionOf(CollectiveTS Collective))
// Object property: example-4DA0.owl#subQuantityOf
SubObjectPropertyOf(subQuantityOf essentialPartOf)
ObjectPropertyDomain(subQuantityOf ObjectUnionOf(QuantityTS Quantity))
ObjectPropertyRange(subQuantityOf ObjectUnionOf(QuantityTS Quantity))
// Object property: example-4DA0.owl#existentiallyDependentOf
SubObjectPropertyOf(existentiallyDependentOf objPropertyTS)
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
IrreflexiveObjectProperty(existentiallyDependentOf)
// Object property: example-4DA0.owl#invExistentiallyDependentOf
SubObjectPropertyOf(invExistentiallyDependentOf objPropertyTS)
InverseObjectProperties(existentiallyDependentOf invExistentiallyDependentOf)
// Object property: example-4DA0.owl#inheresIn
SubObjectPropertyOf(inheresIn existentiallyDependentOf)
FunctionalObjectProperty(inheresIn)
AntisymmetricObjectProperty(inheresIn)
ObjectPropertyDomain(inheresIn ModeTS)
ObjectPropertyRange(inheresIn TimeSlice)
// Object property: example-4DA0.owl#mediates
SubObjectPropertyOf(mediates existentiallyDependentOf)
AntisymmetricObjectProperty(mediates)
ObjectPropertyDomain(mediates RelatorTS)
ObjectPropertyRange(mediates TimeSlice)

// Sub property chain axiom
SubObjectPropertyOf(**SubObjectPropertyChain**(objPropertyTS hasTemporalExtent) hasTemporalExtent)

// DOMAIN OBJECT PROPERTIES

// Object property: ECG.owl#componentOfBrainPerson
SubObjectPropertyOf(componentOfBrainPerson objPropertyTS)
SubObjectPropertyOf(componentOfBrainPerson componentOf)
SubObjectPropertyOf(componentOfBrainPerson existentiallyDependentOf)
SubObjectPropertyOf(componentOfBrainPerson invExistentiallyDependentOf)
ObjectPropertyDomain(componentOfBrainPerson BrainTS)
ObjectPropertyRange(componentOfBrainPerson PersonTS)
// Object property: ECG.owl#componentOfHeartPerson
SubObjectPropertyOf(componentOfHeartPerson invExistentiallyDependentOf)
SubObjectPropertyOf(componentOfHeartPerson componentOf)
SubObjectPropertyOf(componentOfHeartPerson objPropertyTS)
SubObjectPropertyOf(componentOfHeartPerson existentiallyDependentOf)
ObjectPropertyDomain(componentOfHeartPerson HeartTS)
ObjectPropertyRange(componentOfHeartPerson PersonTS)
// Object property: ECG.owl#belongsTo
// Object property: ECG.owl#belongsToECGRecordPerson
SubObjectPropertyOf(belongsToECGRecordPerson objPropertyTS)
SubObjectPropertyOf(belongsToECGRecordPerson belongsTo)
SubObjectPropertyOf(belongsToECGRecordPerson existentiallyDependentOf)
ObjectPropertyDomain(belongsToECGRecordPerson ECGRecordTS)
ObjectPropertyRange(belongsToECGRecordPerson PersonTS)
// Object property: ECG.owl#maps
// Object property: ECG.owl#mapsPartialCyclePartialCardiacElectricalImpulse
SubObjectPropertyOf(mapsPartialCyclePartialCardiacElectricalImpulse existentiallyDependentOf)
SubObjectPropertyOf(mapsPartialCyclePartialCardiacElectricalImpulse invExistentiallyDependentOf)
SubObjectPropertyOf(mapsPartialCyclePartialCardiacElectricalImpulse objPropertyTS)
SubObjectPropertyOf(mapsPartialCyclePartialCardiacElectricalImpulse maps)
ObjectPropertyDomain(mapsPartialCyclePartialCardiacElectricalImpulse PartialCycleTS)
ObjectPropertyRange(mapsPartialCyclePartialCardiacElectricalImpulse PartialCardiacElectricalImpulseTS)
// Object property: ECG.owl#mapsCompleteCycleCompleteCardiacElectricalImpulse
SubObjectPropertyOf(mapsCompleteCycleCompleteCardiacElectricalImpulse existentiallyDependentOf)
SubObjectPropertyOf(mapsCompleteCycleCompleteCardiacElectricalImpulse objPropertyTS)
SubObjectPropertyOf(mapsCompleteCycleCompleteCardiacElectricalImpulse maps)
SubObjectPropertyOf(mapsCompleteCycleCompleteCardiacElectricalImpulse invExistentiallyDependentOf)
ObjectPropertyDomain(mapsCompleteCycleCompleteCardiacElectricalImpulse CompleteCycleTS)
ObjectPropertyRange(mapsCompleteCycleCompleteCardiacElectricalImpulse CompleteCardiacElectricalImpulseTS)
// Object property: ECG.owl#mapsCycleCardiacElectricalImpulse
SubObjectPropertyOf(mapsCycleCardiacElectricalImpulse invExistentiallyDependentOf)
SubObjectPropertyOf(mapsCycleCardiacElectricalImpulse objPropertyTS)
SubObjectPropertyOf(mapsCycleCardiacElectricalImpulse maps)
SubObjectPropertyOf(mapsCycleCardiacElectricalImpulse existentiallyDependentOf)
ObjectPropertyDomain(mapsCycleCardiacElectricalImpulse CycleTS)
ObjectPropertyRange(mapsCycleCardiacElectricalImpulse CardiacElectricalImpulseTS)
// Object property: ECG.owl#contracts
// Object property: ECG.owl#contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart
SubObjectPropertyOf(contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart contracts)
SubObjectPropertyOf(contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart objPropertyTS)
SubObjectPropertyOf(contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart existentiallyDependentOf)
ObjectPropertyDomain(contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart CompleteCardiacElectricalImpulseTS)
ObjectPropertyRange(contractsCompleteCardiacElectricalImpulseCompletelyFunctioningHeart CompletelyFunctioningHeartTS)
// Object property: ECG.owl#contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart
SubObjectPropertyOf(contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart objPropertyTS)
SubObjectPropertyOf(contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart existentiallyDependentOf)
SubObjectPropertyOf(contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart contracts)
ObjectPropertyDomain(contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart PartialCardiacElectricalImpulseTS)
ObjectPropertyRange(contractsPartialCardiacElectricalImpulsePartiallyFunctioningHeart PartiallyFunctioningHeartTS)

// INDIVIDUALS

// Individual: ECG.owl#ECGRecJohn_t2
ClassAssertion(ECGRecJohn_t2 owl:Thing)
ObjectPropertyAssertion(timeSliceOf ECGRecJohn_t2 ECGRecJohn)
ObjectPropertyAssertion(belongsToECGRecordPerson ECGRecJohn_t2 John_t2)
// Individual: ECG.owl#Cycle1_t2
ClassAssertion(Cycle1_t2 owl:Thing)
ObjectPropertyAssertion(inheresIn Cycle1_t2 ECGRecJohn_t2)
ObjectPropertyAssertion(timeSliceOf Cycle1_t2 Cycle1)
// Individual: ECG.owl#John_t4
ClassAssertion(John_t4 owl:Thing)
ObjectPropertyAssertion(timeSliceOf John_t4 John)
// Individual: ECG.owl#JohnHeart_t1
ClassAssertion(JohnHeart_t1 owl:Thing)
ObjectPropertyAssertion(timeSliceOf JohnHeart_t1 JohnHeart)
ObjectPropertyAssertion(componentOfHeartPerson JohnHeart_t1 John_t1)
// Individual: ECG.owl#John_t3
ClassAssertion(John_t3 owl:Thing)
ObjectPropertyAssertion(timeSliceOf John_t3 John)
// Individual: ECG.owl#Cycle2
ClassAssertion(Cycle2 PartialCycle)
ClassAssertion(Cycle2 owl:Thing)
// Individual: ECG.owl#ECGRecJohn
ClassAssertion(ECGRecJohn owl:Thing)
// Individual: ECG.owl#Cycle3
ClassAssertion(Cycle3 owl:Thing)
// Individual: ECG.owl#John_t2
ClassAssertion(John_t2 owl:Thing)
ObjectPropertyAssertion(timeSliceOf John_t2 John)
// Individual: ECG.owl#John
ClassAssertion(John owl:Thing)
// Individual: ECG.owl#Cycle1
ClassAssertion(Cycle1 CompleteCycle)
ClassAssertion(Cycle1 owl:Thing)
// Individual: ECG.owl#John_t1
ClassAssertion(John_t1 owl:Thing)
)

ObjectPropertyAssertion(timeSliceOf John_t1 John)
// Individual: ECG.owl#ECGRecJohn_t4
ClassAssertion(ECGRecJohn_t4 owl:Thing)
ObjectPropertyAssertion(belongsToECGRecordPerson ECGRecJohn_t4 John_t4)
ObjectPropertyAssertion(timeSliceOf ECGRecJohn_t4 ECGRecJohn)
// Individual: ECG.owl#JohnHeart_t2
ClassAssertion(JohnHeart_t2 owl:Thing)
ObjectPropertyAssertion(componentOfHeartPerson JohnHeart_t2 John_t2)
ObjectPropertyAssertion(timeSliceOf JohnHeart_t2 JohnHeart)
// Individual: ECG.owl#JohnHeart_t3
ClassAssertion(JohnHeart_t3 owl:Thing)
ObjectPropertyAssertion(timeSliceOf JohnHeart_t3 JohnHeart)
ObjectPropertyAssertion(componentOfHeartPerson JohnHeart_t3 John_t3)
// Individual: ECG.owl#JohnHeart
ClassAssertion(JohnHeart owl:Thing)
// Individual: ECG.owl#ECGRecJohn_t3
ClassAssertion(ECGRecJohn_t3 owl:Thing)
ObjectPropertyAssertion(timeSliceOf ECGRecJohn_t3 ECGRecJohn)
ObjectPropertyAssertion(belongsToECGRecordPerson ECGRecJohn_t3 John_t3)
// Individual: ECG.owl#JohnHeart_t4
ClassAssertion(JohnHeart_t4 owl:Thing)
ObjectPropertyAssertion(componentOfHeartPerson JohnHeart_t4 John_t4)
ObjectPropertyAssertion(timeSliceOf JohnHeart_t4 JohnHeart)
// Individual: ECG.owl#Cycle2_t3
ClassAssertion(Cycle2_t3 owl:Thing)
ObjectPropertyAssertion(inheresIn Cycle2_t3 ECGRecJohn_t3)
ObjectPropertyAssertion(timeSliceOf Cycle2_t3 Cycle2)
// Individual: ECG.owl#Cycle3_t4
ClassAssertion(Cycle3_t4 owl:Thing)
ObjectPropertyAssertion(inheresIn Cycle3_t4 ECGRecJohn_t4)
ObjectPropertyAssertion(timeSliceOf Cycle3_t4 Cycle3)
// Sub property chain axiom
SubObjectPropertyOf(SubObjectPropertyChain(objPropertyTS hasTemporalExtent)
hasTemporalExtent)