Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Departamento de Informática

Uma Linguagem de Domínio Específico para AORE

Nº 30226 Ana Rita Duarte Oliveira

Orientador Prof. Doutor João Baptista da Silva Araújo Júnior

Co-Orientador Prof. Doutor Vasco Miguel Moreira do Amaral

Dissertação apresentada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa para obtenção do grau de Mestre em Engenharia Informática.

2º Semestre de 2009/2010

28 de Julho de 2010

Nº do aluno: 30226
Nome: Ana Rita Duarte Oliveira

Título da dissertação:
Uma Linguagem de Domínio Específico para AORE

Palavras-Chave:

- Engenharia de Requisitos Orientada a Aspectos
- Desenvolvimento Orientado a Modelos
- Linguagem de Domínio Específico
- Metamodelação

Keywords:

- Aspect-Oriented Requirements Engineering
- Model Driven Development
- Domain-Specific Language
- Metamodeling

Agradecimentos

Gostaria de agradecer a todas as pessoas que durante este último ano de trabalho me acompanharam e tornaram possível a realização desta dissertação de mestrado.

Especialmente, gostaria de agradecer ao meu orientador João Araújo pela proposta de trabalho sugerida e por todo o apoio, disponibilidade e boa disposição que me ofereceu em todas as fases desta dissertação. Em seguida, mas não menos importante, queria agradecer ao meu co-orientador Vasco Amaral, por todo o auxílio prestado ao longo deste trabalho.

Agradeço ao Vasco Sousa pela disponibilidade e ajuda que me forneceu, e a todos os meus colegas que participaram na fase de avaliação da ferramenta proposta neste trabalho, assim como aos que me ofereceram qualquer outro tipo de ajuda.

Agradeço aos meus colegas que trabalharam comigo quase diariamente, tais como a Sara e outros, pois tornaram a elaboração desta dissertação mais leve e bem-disposta.

Finalmente agradeço à minha família, em especial aos meus pais e irmãos, por todo o apoio, dedicação, disponibilidade e bons momentos que me proporcionaram, e que assim tornaram este trabalho não só possível, como muito agradável.

Resumo

A Engenharia de Requisitos Orientada a Aspectos (EROA) consiste em identificar, modularizar, especificar e compor assuntos transversais (*crosscutting concerns*), conhecidos como aspectos. A abordagem AORE (*Aspect - Oriented Requirements Engineering*) foi uma das abordagens pioneiras de EROA. É uma abordagem sistemática usada para descobrir e estruturar requisitos com base em *viewpoints* e aspectos.

No entanto, há muito por fazer em termos de investigação sobre a AORE, como: uma representação diagramática de seus modelos (já que a representação actual é puramente textual), tornando a abordagem mais fácil de utilizar por engenheiros de software e mais fácil de se integrar em ambientes de desenvolvimento orientado a modelos; a especificação completa do seu metamodelo; e a especificação e implementação rigorosa de uma ferramenta de suporte à abordagem.

Para atingir estes objectivos, este trabalho propõe a especificação de uma Linguagem de Domínio Específico (LDE) para AORE. Uma LDE tem o propósito de especificar e modelar conceitos num determinado domínio, tendo vantagens em relação às linguagens de domínio geral, tais como permitir expressar um problema na linguagem e no nível de abstracção desejados, delegando a especificação de como resolver o problema para os níveis de abstracção inferiores. Para se criar uma LDE, é necessário começar por especificar a sua sintaxe abstracta, recorrendo a um modelo denominado metamodelo. Este metamodelo expressa como se formam os modelos dessa mesma LDE (no caso, o metamodelo da AORE) e será dado como *input* para os *workbenches* da linguagem que vão gerar o editor correspondente. Com um editor apropriado para a linguagem podemos especificar modelos diagramáticos com a notação definida. Concluindo, esta dissertação pretende desenhar e desenvolver uma LDE para a abordagem AORE.

Abstract

The Requirements Engineering Aspect-Oriented area consists of identifying, modularizing, specifing, and composing crosscutting issues (crosscutting concerns), also known as aspects. The approach AORE (Aspect-Oriented Requirements Engineering) was a pioneering approach to EROA. It is a systematic approach used to discover and structure requirements based on viewpoints and aspects.

However, there are lots of work that remain to be done in terms of research on AORE, such as a diagrammatic representation of their models (as the current representation is purely textual), making the approach easier to use for software engineers and easier to integrate into model-driven development environments; the complete specification of its metamodel; and specification and rigorous implementation of a tool support for the approach.

To achieve this goal this work proposes to specify a Domain-Specific Language (DSL). A DSL aims to specify and model concepts in a given field, with several advantages over the general area of languages, such as allowing expressing a problem in the desired language and level of abstraction, delegating the specificationof how to solve for the lower levels of abstraction. In order to create a DSL successfully, it is usually necessary to start by specifying its abstract syntax using a model called metamodel. This model express how to form the models of the same DSL (in this case, the AORE metamodel) and it will be given as an input to the workbenches of the language that will generate the appropriate editor. With a proper editor for the language we can specify models with the defined diagrammatic notation. In conclusion, this dissertation aims to design and develop a DSL for the AORE approach.

Lista de Acrónimos

ALPH	***A**spect **L**anguage for **P**ervasive **H**ealthcare*
AMPLE	***A**spect-Oriented, **M**odel-Driven **P**roduct **L**ine **E**ngineering*
AOP	***A**spect-**O**riented **P**rogramming*
AOSD	***A**spect-**O**riented **S**oftware **D**esign*
AORE	***A**spect-**O**riented **R**equirements **E**ngineering*
API	***A**pplication **P**rogramming **I**nterface*
ASF	***A**lgebraic **S**pecification **F**ormalism*
ATMOL	***AT**mospheric **MO**deling **L**anguage*
CASE	***C**omputer-**A**ided **S**oftware **E**ngineering*
CODA	***C**ontext-**O**riented **D**omain **A**nalysis*
CTADEL	***C**ode-generation **T**ool for **A**pplications based on **D**ifferential **E**quations using high-level **L**anguage*
DARE	***D**omain Specific **A**nalysis and **R**euse **E**nvironment*
DFD	***D**ata **F**low **D**iagram*
DSAL	***D**omain **S**pecific **A**spect **L**anguage*
DSL	***D**omain **S**pecific **L**anguage*
DSOA	**D**esenvolvimento de **S**oftware **O**rientado a **A**spectos
DSPL	***D**inamic **S**oftware **P**roduct **L**ine*
DSSA	***D**omain **S**pecific **S**oftware **A**rchitectures*
EBNF	***E**xtended **B**ackus **N**aur **F**orm*
EMF	***E**clipse **M**odeling **F**ramework*
ER	**E**ngenharia de **R**equisitos
ER	***E**ntity **R**elashionship*
EROA	**E**ngenharia de **R**equisitos **O**rientada a **A**spectos
EROO	**E**ngenharia de **R**equisitos **O**rientada a **O**bjectivos
FAST	***F**amily-Oriented **A**bstractions, **S**pecification, and **T**ranslation*
FODA	***F**eature-**O**riented **D**omain **A**nalysis*

GAL	***G**raphics **A**daptor **L**anguage*
GME	***G**eneric **M**odeling **E**nvironment*
GPL	***G**eneral **P**urpose **L**anguage*
GUI	***G**raphical **U**ser **I**nterface*
IDE	***I**ntegrated **D**evelopment **E**nvironment*
JPPAL	***J**ava **P**arallel **P**rogramming **A**nnotation **L**ibrary*
KAOS	***K**nowledge **A**cquisition in Aut**O**mated **S**pecification / **K**eep **A**ll **O**bjectives **S**atisfied*
LDE	**L**inguagem de **D**omínio **E**specífico
LHS	***L**eft **H**and **S**ide*
MATA	***M**odeling **A**spects Using a **T**ransformation **A**pproach*
MDD	***M**odel-**D**riven **D**evelopment*
MDA	***M**odel-**D**riven **A**rchitecture*
MOF	***M**eta-**O**bject **F**acility*
OCL	***O**bject **C**onstraint **L**anguage*
ODE	***O**ntology-based **D**omain **E**ngineering*
ODM	***O**rganization **D**omain **M**odeling*
OME	***O**rganization **M**odelinh **E**nvironment*
OMG	***O**bject **M**anagement **G**roup*
OMT	***O**bject **M**odeling **T**echnique*
PADS	***P**rocessing **A**d hoc **D**ata **S**ources*
PDE	***P**artial **D**ifferential **E**quation*
RHS	***R**ight **H**and **S**ide*
RNF	**R**equisito **N**ão **F**uncional
SDF	***S**yntax **D**efinition **F**ormalism*
TCL	***T**ool **C**ommand **L**anguage*
TK	***T**ool**K**it*
UML	***U**nified **M**odeling **L**anguage*
VAODA	***V**iewpoints and **A**spect-**O**riented **D**omain **A**nalysis*
XML	*e**X**tensible **M**arkup **L**anguage*

Índice de Conteúdos

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1. Contexto e Motivação

A Engenharia de Requisitos (ER) é a área da Engenharia de Software que trata da definição e especificação dos requisitos de software. Um requisito é uma funcionalidade, propriedade ou atributo do sistema, uma descrição de como o sistema se deve comportar ou dos serviços disponibilizados. É ainda a definição de restrições existentes no sistema e capazes de condicionar o seu comportamento e desenvolvimento (Sommerville, 2007). Os requisitos são classificados como requisitos do utilizador e requisitos do sistema, sendo que estes últimos se dividem em requisitos funcionais, não funcionais e de domínio. Os requisitos funcionais são os serviços que o sistema deve produzir, assim como o modo como deve reagir a determinados eventos. Os requisitos não funcionais são compostos pelos requisitos de produto, organizacionais e externos, e finalmente, os requisitos de domínio surgem do domínio da aplicação do sistema, reflectindo as características do seu domínio (Sommerville, 2007). O objectivo da ER é conseguir obter um conjunto de metodologias capazes de lidar cada vez melhor com a problemática da identificação e tratamento destes requisitos. Abaixo encontram-se descritos alguns tipos de abordagens actualmente existentes.

- **ER Orientada a Objectivos** (Lamsweerde, 2001) - oferece mecanismos de levantamento de requisitos baseados na identificação de objectivos para melhor traduzir o que um requisito contempla. Fornece um elevado nível de abstracção e os seus objectivos podem ser funcionais ou não funcionais. Como exemplo destacam-se os métodos i* (Yu, 1995) e KAOS (Lamsweerde *et* al., 1991);
- **ER Orientada a Objectos** – nesta abordagem, um objecto representa um requisito que contém informação acerca do processo, produto e da sua funcionalidade (Rumbaugh *et* al., 1991). Como exemplo de abordagens tradicionais foco a OOSE (Jacobson, 1991), e a OMT (Rumbaugh *et* al., 1999). Actualmente, as abordagens são baseadas em UML (Rumbaugh *et* al., 1999);

- **ER Orientada a *Viewpoints*** – abordagem em que os *viewpoints* encapsulam a informação de cada perspectiva do sistema através de requisitos que contêm a sua especificação (e.g. PREview (Sawyer *et* al., 1996);
- **ER Orientada a Aspectos** – esta abordagem efectua a identificação, modularização e composição de assuntos transversais (*crosscutting concerns*) ou aspectos. Isto facilita a sua identificação e tratamento, dando origem a processos de desenvolvimento de software mais modularizados. Como exemplo, realço à abordagem AORE (Rashid *et* al., 2003).

O foco deste trabalho é na Engenharia de Requisitos Orientada a Aspectos (EROA). A EROA tem o aspecto como conceito base, e possui a vantagem de suportar a separação de assuntos. Um aspecto é um *crosscutting*, ou seja, um assunto cuja funcionalidade se encontra dispersa por múltiplos módulos do sistema. Deste modo, os assuntos podem ser modularizados, tratados e compreendidos separadamente, o que facilita a evolução dos requisitos. Este tema é abrangido pelo Desenvolvimento de Software Orientado a Aspectos (DSOA), discutido no Capítulo 3, e possui diversas abordagens que o suportam, como é possível constatar adiante.

Algumas das abordagens que o DSOA contempla, não possuem uma representação visual, não podendo ser esquematizadas nem definidas com elevada precisão. Uma das abordagens mais conhecidas é a AORE – *Aspect-Oriented Requirements Engineering*, orientada a aspectos com *viewpoints*. No entanto, a AORE é um exemplo de abordagem que não possui representação visual dos seus modelos e não especifica os meta-dados através de um metamodelo. Dado que se trata de uma abordagem cujos modelos são textuais e, portanto, eventualmente memos atractivos para a sua utilização, seria conveniente a especificação de uma ferramenta construída através de uma linguagem visual e mais abstracta e que suporte os conceitos específicos, ou seja, uma LDE (Linguagem de Domínio Específico).

Uma LDE é uma linguagem com um elevado nível de abstracção, cujo objectivo é modelar e especificar os conceitos de um dado domínio. Usando a abordagem orientada a modelos, através da especificação do modelo que descreve como devem ser os modelos da própria LDE (ou metamodelo), é possível usar ferramentas gerativas para obter um editor. Este editor permite desenhar modelos que obedecem à especificação do metamodelo tanto ao nível da forma textual como diagramática. É com base no metamodelo que se obtém uma validação dos modelos desenhados.

A técnica AORE enquadra-se na área da Engenharia de Requisitos Orientada a Aspectos, como tal, é fundamental ter em conta as necessidades básicas e as preferências do engenheiro

de software comum. Deste modo, ir-se-à proceder à realização de uma LDE visual/diagramática, ao invés de uma LDE textual. De notal que a única ferramenta que fornece suporte à técnica AORE é a ferramenta ARCaDe, introduzida no Capítulo 2, que é uma ferramenta textual e que utiliza templates XML para especificar os elementos da técnica. Nas áreas de desenho de software, dá-se preferência à utilização de modelos visuais para obter as especificações dos sistemas, pois há um aumento da abstracção dos problemas a resolver, e consequentemente uma maior percepção do sistema como um todo.

Pelos motivos focados, esta dissertação adopta a metodologia AORE na sua vertente orientada a *viewpoints*, com o objectivo de obter a representação da mesma através de uma Linguagem de Domínio Específico visual/diagramática.

1.2. Objectivos

O principal objectivo desta dissertação consiste em elaborar uma LDE para a metodologia AORE. Para tal, é necessário especificar um metamodelo para esta abordagem.

A especificação da LDE com o auxílio de uma ferramenta gerativa, neste caso o Eclipse, dá origem a um editor de suporte à linguagem que permite elaborar modelos visuais AORE de acordo com o metamodelo especificado, tornando a abordagem mais fácil de utilizar por engenheiros de software e mais fácil de se integrar em ambientes de desenvolvimento orientado a modelos.

A abordagem AORE ficará mais precisa e com possibilidade de oferecer uma notação gráfica, através de uma ferramenta que suporte a LDE. Isto facilita o processo de modelação dos requisitos ao Engenheiros de Software comum.

1.3. Organização do documento

Os restantes Capítulos desta dissertação estão organizados como é descrito em seguida.

- **Capítulo 2**: Neste Capítulo é introduzido o conceito Desenvolvimento de Software Orientado a Aspectos com destaque na abordagem Engenharia de Requisitos Orientada a Aspectos. É aplicada esta abordagem ao caso de estudo *Smart Home*, e são posteriormente introduzidas outras abordagens da EROA menos significativas para esta dissertação.
- **Capítulo 3**: Neste Capítulo são abordados os temas Linguagem de Domínio Específico e Desenvolvimento Orientado a Modelos. São posteriormente apresentadas algumas vantagens e desvantagens de uma LDE, assim como uma

breve comparação com as linguagens GPL. Aborda-se a Sintaxe e a Semântica de uma linguagem, e introduz-se a linguagem OCL. Em seguida é dada uma explicação acerca das fases que o desenvolvimento de uma LDE abrange, e por fim são focadas algumas ferramentas úteis ao seu desenvolvimento.

- **Capítulo 4**: É dado o estado da arte através de breves resumos explicando alguns trabalhos relacionados com o trabalho que se está a desenvolver desta dissertação.
- **Capítulo 5**: Neste capítulo é descrito o *plug-in* VisualAORE. Inicialmente é introduzido o metamodelo da linguagem em pormenor, e em seguida é explicado como se obteve o editor através do metamodelo. Finalmente, com o objectivo de introduzir a linguagem e o editor, é aplicada a linguagem ao caso de estudo Via Verde.
- **Capítulo 6**: O capítulo 6 aborda a avaliação da linguagem e do editor. Na secção 6.1 é efectuada a avaliação recorrendo ao caso de estudo *Smart Home*, e na secção 6.2 é exposto o resultado da avaliação feita através de testes realizados a um grupo de utilizadores.
- **Capítulo 7**: No capítulo 7 são abordadas as conclusões do trabalho efectuado, as suas limitações e possíveis trabalhos futuros.
- **Anexo A**: Este anexo contém as tabelas que possuem os construtores da linguagem de composição, ou seja, a tabela da descrição das *Constraint Actions*, a tabela de descrição dos *Constraint Operators* e por fim, a tabela de descrição das *Outcome Actions*.
- **Anexo B**: O anexo B contém o modelo Emfatic que deu origem ao editor base da ferramenta VisualAORE.
- **Anexo C**: O anexo C contém o modelo Emfatic do sub-editor que acenta no elemento *ConcernsAggregationModule* da ferramenta VisualAORE.
- **Anexo D**: Neste anexo pode observar-se o modelo Emfatic do sub-editor presente no elemento *CompositionRulesAggregationModule* da ferramenta desenvolvida.
- **Anexo E**: Este anexo contém o modelo Emfatic do sub-editor presente no elemento *ViewpointsAggregationModule* da ferramenta desenvolvida.
- **Anexo F**: No anexo F está contida parte da especificação textual caso de estudo *Smart Home*.
- **Anexo G**: Este anexo contém um exemplar das intruções de teste que foram fornecidas aos utilizadores, aquando da fase de avaliação da ferramenta desenvolvida nesta dissertação.

- **Anexo H**: No anexo H pode observar-se um exemplar do questionário fornecido aos utilizadores que testaram a ferramenta VisualAORE.
- **Anexo I**: Finalmente, o Anexo I possui o manual de utilização da ferramenta proposta neste trabalho.

2. Engenharia de Requisitos Orientada a Aspectos

2.1. DSOA

O Desenvolvimento de Software Orientado a Aspectos (do inglês *Aspect-Oriented Software Development*, AOSD) (Filman *et* al., 2005) é um dos métodos mais recente no desenvolvimento de software. Geralmente, foca-se na fase de implementação do ciclo de vida do software, onde os peritos identificam aspectos a partir de código já realizado. No entanto, os aspectos surgem muito mais cedo no ciclo de vida do software, pelo que é necessário tratá-los anteriormente, mais precisamente na fase de tratamento e análise de requisitos.

Na maioria dos sistemas de grande porte, pode tornar-se complicado manter uma relação coerente entre os requisitos do sistema, na medida em que um simples requisito pode ser implementado por várias componentes do sistema, ou seja, pode estar disperso pelos vários módulos que o constituem.

O Desenvolvimento de Software Orientado a Aspectos (DSOA) fornece uma metodologia para separar os assuntos (*concerns*) do sistema, a fim de obter sistemas mais modularizados e, portanto, mais preparados para a sua evolução (Rashid *et* al., 2003). Um assunto pode ser visto como uma funcionalidade do sistema, propriedade que o sistema deve fornecer, ou algo que tem que ser considerado (Filman *et* al., 2005).

A necessidade de obter soluções para a separação dos assuntos na fase de Engenharia de Requisitos, inexistentes nos métodos tradicionais fez com que surgisse este novo método, que tem como base, o aspecto.

O DSOA surge para solucionar o problema da modularização, incidindo em especificações de software que possuem assuntos que são difíceis de separar, os chamados assuntos transversais ou *crosscutting concerns* (Rashid *et* al., 2002).

Por *crosscutting concern* entende-se um assunto cuja especificação se encontra dispersa entre vários outros assuntos, atravessando vários conjuntos de requisitos ou unidades da especificação do sistema. Tem assim um efeito em vários requisitos, causando consequências em fases finais do desenvolvimento do sistema, fase em que são geralmente identificados (nível da implementação). Por este motivo, os *crosscutting concerns* devem ser encapsulados em módulos separados, chamados aspectos: "*An aspect is a concern whose functionality is triggered by other concerns, and in multiple situations.*" (Sommerville, 2007). Um aspecto é portanto, um mecanismo de abstracção.

As consequências que os *crosscutting concerns* originam são, essencialmente, a falta de localização dos assuntos independentemente, num só módulo, o que impossibilita a sua composição e estudo da influência na especificação do sistema, originando casos difíceis de compreender e manter.

Desta forma, surge o efeito *scattering* (dispersão), ou seja, a inclusão dos assuntos transversais em várias especificações de outros assuntos. Este facto é indesejado, pois dá origem ao efeito *tangling* (emaranhamento) nas especificações. Especificações emaranhadas são especificações que contêm propriedades diferentes de assuntos diferentes.

A separação de *concerns* é o princípio básico do desenho de software, pois este deve ser desenhado para que cada unidade ou componente realize uma e uma só tarefa.

Na programação orientada a aspectos, os aspectos são implementados dentro do programa e é escolhido onde é que o aspecto deve ser associado. Estes são os chamados *join points*. Dado que os aspectos são tratados em separado, então, numa fase de pré-compilação, eles são ligados a estes pontos de ligação (Clarke e Baniassad, 2005). Este facto favorece a programação, pois torna-a não invasiva. Desta forma, para adicionar ou remover funcionalidades a um sistema, não é necessário remodelar nem analisar o código todo, mas apenas o módulo que se deseja alterar, ou proceder à realização de um novo módulo que se queira adicionar.

As técnicas de DSOA englobam as actividades de Decomposição, Representação e Composição. Na Decomposição decompõe-se o problema e identificam-se os aspectos. Na Representação especifica-se e implementa-se cada aspecto num módulo separado. Por fim, na Composição, compõe-se ou integram-se os aspectos, com os restantes módulos do sistema.

O DSOA tem as seguintes vantagens (Clarke e Baniassad, 2005):

- Aumento da capacidade de pensar sobre um domínio de um problema e respectiva solução;
- Redução do tamanho de código da aplicação, dos custos de desenvolvimento e do tempo de manutenção;
- Aumento da reutilização de código;
- Reutilização dos requisitos, arquitectura e nível de desenho;
- Aumento da capacidade para Linhas de Produto de Engenharias;
- Disponibilização de uma adaptação a aplicações sensíveis ao contexto;
- Aumento dos métodos de modelação.

O DSOA tem as seguintes desvantagens (Clarke e Baniassad, 2005):

- Não é ainda uma técnica usada intensivamente na indústria de software;
- Os processos de verificação e validação tornam-se difíceis, obrigando a um melhor conhecimento da relação entre os aspectos e as propriedades não funcionais do sistema.

Na próxima secção sumarizam-se os princípios básicos da Engenharia de Requisitos Orientada a Aspectos e, seguidamente, descrevem-se algumas abordagens EROA, com ênfase na abordagem AORE.

2.2. Engenharia de Requisitos Orientada a Aspectos

A Engenharia de Requisitos Orientada a Aspectos é constituída pelas várias técnicas (AORE com *viewpoints*, AORA, MATA, Theme, *use-cases* com Aspectos, entre outras) que tratam os assuntos transversais dos requisitos funcionais e não funcionais, numa primeira fase do desenvolvimento de software.

O seu objectivo é fornecer uma melhor separação dos assuntos transversais. Dado que estes estão presentes em várias funcionalidades de um sistema, é fundamental dividir o tratamento que lhes é dado, em duas fases:

- Identificação de cada assunto em separado, ou seja, modulação dos assuntos transversais em aspectos;
- Reflexão sobre cada aspecto, estudo do impacto entre eles, encontrando as contribuições negativas e positivas que causam entre si, e estudo do impacto que originam nos requisitos onde estão presentes.

A identificação de aspectos numa fase inicial do desenvolvimento de software é importante, e dá origem ao conceito *Early Aspect* (Baniassad *et* al., 2006). Estes aspectos estão dispersos pelos modelos de requisitos, e surgem antes da fase de implementação do software, isto é, nas fases de análise de domínio, análise de requisitos e desenho arquitectural. A sua identificação e gestão nestas fases do desenvolvimento do software contribuem para (Baniassad *et* al., 2006):

- Aumentar a modularidade;
- Aumentar a consistência entre requisitos e os modelos arquitecturais, e posteriormente com a implementação;
- Fornecer uma base racional de rastreabilidade para os aspectos através das actividades do seu ciclo de vida;
- Assegurar que os assuntos transversais do sistema são capturados como aspectos na implementação.

As actividades que compõem o tratamento dos aspectos são (Baniassad *et* al., 2006):

- **Identificação**: Nesta fase existe uma procura de termos, geralmente atributos de qualidade, que apontam para a existência de aspectos. Deve procurar-se o impacto dos requisitos com vista a descobrir sobreposições, e por fim, deve procurar-se os assuntos *scattered*, ou seja, os termos, conceitos ou comportamentos que aparecem em diversos requisitos.
- **Captura**: Fase em que existe uma reorganização dos requisitos para que cada modelo de requisitos seja relativo a apenas um assunto.
- **Composição:** Esta fase consiste em declarar formalmente o impacto dos requisitos para especificar como os assuntos devem ser compostos. Várias técnicas podem ser usadas, tal como a AORE com *viewpoints*.
- **Análise**: É feita uma análise dos aspectos modularizados para se proceder à identificação e compreensão dos conflitos com outros requisitos e possíveis inconsistências.

A importância dos *Early Aspects* é também notada em fases posteriores do desenvolvimento de software, para que se consiga ter uma visão mais ampla do sistema, melhorar a rastreabilidade, assim como a forma de tratar e negociar os conflitos.

De uma forma geral, a falta de tratamento dos assuntos transversais pode originar a inibição, adaptabilidade e fraca evolução do software.

Dentro da Engenharia de Requisitos Orientada a Aspectos existem várias abordagens, pelo que o tema final deste Capítulo será abordar algumas delas. Uma abordagem AORE é caracterizada pelas quatro características seguintes (Araújo *et* al., 2005):

- Existência de meios eficazes para identificar as propriedades transversais numa especificação de requisitos;
- Existência de habilidade para modularizar as propriedades transversais;
- Existência de meios adequados para representar os aspectos do nível dos requisitos;
- Capacidade de compor os requisitos aspectuais e não aspectuais para compreender o efeito cumulativo dos aspectos do nível dos requisitos noutros requisitos do sistema.

2.3. A Abordagem AORE

Na abordagem *Aspect-Oriented Requirement Engineering* (AORE), o modelo usado para tratar os assuntos transversais é baseado em PREview (*Process and Requirements Engineering viewpoints*) (Sawyer *et* al., 1996), e portanto utiliza um modelo de *viewpoints* para tratar os requisitos. O método PREview encapsula informação parcial do sistema em *viewpoints*, mas não toda a informação acerca do sistema. A estrutura deste método é a seguinte:

- Identificar os assuntos que afectam o sistema;
- Derivar um conjunto de questões que assegurem que a informação obtida para satisfazer os assuntos é suficiente;
- Realizar o levantamento e negociar requisitos, o que assegura que o sistema satisfaz os assuntos identificados.

O novo modelo proposto em AORE adiciona o processo de composição ao já existente PREview, processo que é realizado numa base XML (Rashid *et* al., 2003).

No modelo abaixo (Figura 2.1) focam-se as seis fases da técnica AORE e as suas sub-fases. Na fase de Identificação e Especificação são identificados e especificados os assuntos (essencialmente os requisitos não funcionais) e os requisitos funcionais com base nas perspectivas (*viewpoints*) dos *stakeholders*, através da análise dos requisitos iniciais. Após esta fase, são identificadas as relações entre os assuntos e os *stakeholders* para uma posterior identificação dos assuntos candidatos a aspectos, ou seja, aqueles que afectam mais do que um *stakeholder*.

Dada a identificação dos aspectos, segue-se a fase da Composição, em que são definidas detalhadamente as regras de composição que relacionam os requisitos dos *stakeholders* e dos assuntos. Estas regras operam ao nível do requisito individualmente, e não do módulo que os envolve.

De seguida dá-se a fase do Tratamento de Conflitos, que engloba a construção de uma tabela de contribuições entre os assuntos, a atribuição de pesos aos aspectos que entram em conflito, e finalmente a resolução dos conflitos. Nesta fase, é possível haver uma revisão da especificação dos requisitos. A tabela de contribuições contém apenas os assuntos e as contribuições entre eles, em que através do símbolo '+' se exprime que existe uma contribuição positiva para os outros aspectos, e através do símbolo '-' se representa uma contribuição negativa para com eles. A atribuição de pesos é feita através de um número real no intervalo [0.. 1], e representa a prioridade do aspecto em relação aos requisitos do *stakeholder*. É de notar a recente existência de uma formulação matemática (Sardinha *et* al., 2010) capaz de resolver a escalabilidade e a tendência a erro presentes na resolução de conflitos mostrada pelos métodos existentes da Engenharia de Requisitos Orientada a Aspectos. Esta formulação matemática pode ser implementada através de uma técnica baseadas em pesquisa denominada Algoritmo Genético.

Para concluir o processo, são especificadas as dimensões do aspecto, ou seja, descreve-se o seu impacto em termos de mapeamento e influência. Esta fase consiste em determinar a influência que um aspecto possui em fases mais avançadas do desenvolvimento do software, e identificar o seu mapeamento numa função, decisão ou aspecto do sistema.

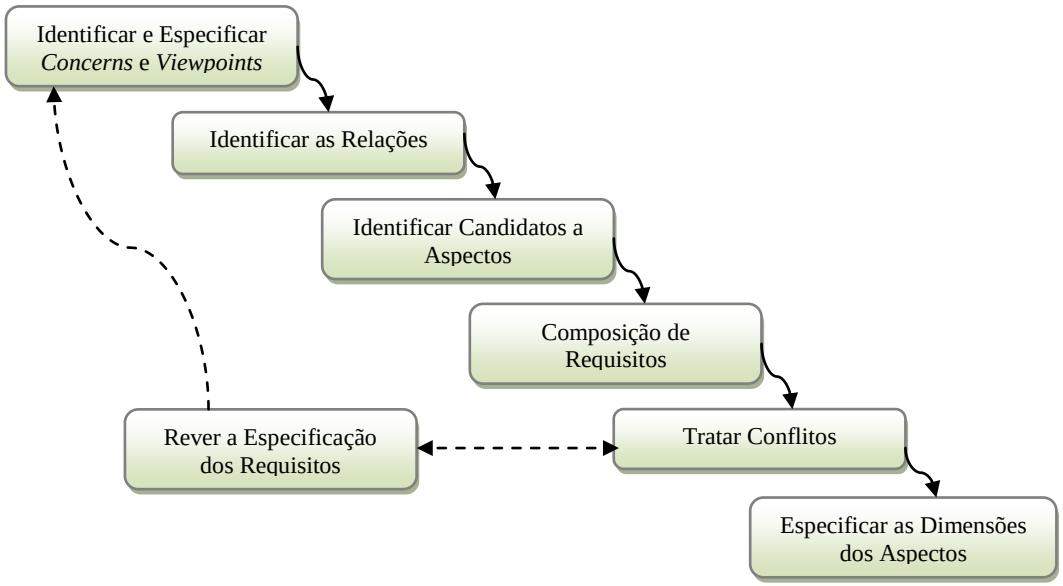

Figura 2.1. Modelo AORE

Na realidade, a técnica AORE é genérica, sendo que nesta dissertação, é usada a abordagem orientada a *viewpoints* (Rashid *et* al., 2003), estendendo a abordagem previamente existente, PREview.

Um *viewpoint* é um encapsulamento parcial de informação do sistema, visto apenas de uma perspectiva. Segundo (Sommerville e Sawyer, 1997), um *viewpoint* é composto pelas componentes *name*, *focus*, *concerns*, *sources*, *requirements* e *history*. Esta abordagem é importante na medida em que permite que o sistema seja analisado por todas as perspectivas dos humanos ou outros sistemas que com ele interagem, os *stakeholders*. Um *stakeholder é* portanto qualquer pessoa ou sistema que possui um contacto ou influência directa ou indirecta com o sistema (Sommerville, 2007).

O primeiro passo a realizar na utilização do método AORE orientado a *Viewpoints* é a identificação e especificação dos assuntos e dos requisitos dos *stakeholders* em XML. A Figura 2.2 exemplifica uma especificação realizada na ferramenta ARCaDe (ferramenta introduzida em seguida) de um *viewpoint* para o sistema Via Verde (Rashid *et* al., 2003), ilustrando o *viewpoint* Veículo.

```
<?xml version="1.0" ?>
- <Viewpoint name="Veículo">
        <Requirement id="1">O veículo entra no sistema quando está a dez metros de
        distância da portagem.</Requirement>
       <Requirement id="2">O veículo entra na portagem.</Requirement>
       <Requirement id="3">O veículo sai da portagem.</Requirement>
       <Requirement id="4">O veículo sai do sistema quando se encontra a vinte metros
        de distância da portagem.</Requirement>
        -<Viewpoint name="VeículoDesautorizado">
           <Requirement id="1">A matrícula do veículo é fotografada.</Requirement>
                    </Viewpoint>
  </Viewpoint>
```

Figura 2.2. *Viewpoint* Veículo em XML

Na Figura 2.3 apresenta-se a especificação de um *concern* Tempo de Resposta.

```
<?xml version="1.0" ?>
- <Concern name="Tempo de Resposta">
  - <Requirement id="1">
      O sistema necessita de reagir atempadamente com o objective de:
        <Requirement id="1.1">ler o identificador ;</Requirement>
        <Requirement id="1.2">ligar a luz (de forma a ficar verde ou
        amarela);</Requirement>
        <Requirement id="1.3">mostrar a quantia a pagar ;</Requirement>
        <Requirement id="1.4">fotografar a matrícula traseira do
        veículo;</Requirement>
        <Requirement id="1.5">disparar o alarme;</Requirement>
        <Requirement id="1.6">responder á activação e reativação do
        identificador.</Requirement>
    </Requirement>
</Concern>
```

Figura 2.3. C*oncern* Tempo de Resposta em XML

O segundo passo do método AORE consite em relacionar os *viewpoints* e os *concerns*. As relações entre os *viewpoints* e *concerns* são obtidas por análise dos requisitos de cada um. A representação das relações é feita através de uma tabela semelhante à Tabela 2.1, onde é feita a relação entre os assuntos e os requisitos, observando-se os assuntos que estão presentes em cada *viewpoint*. P.e., na Tabela 2.1, o assunto Tempo de Resposta está presente nos *viewpoints* Veículo e Cliente. Esta fase consiste também em relacionar os *concerns* entre sí, através da atribuição de contribuições, para encontrar situações de confito. Na Tabela 2.2 observa-se que o assunto Tempo de Resposta contribui negativamente para o assunto Segurança, e vice-versa, o que dá origem a um coflito.

A fase de identificação e especificação de requisitos permite começar a haver uma noção dos assuntos que podem ser transversais (Rashid *et* al., 2003).

Tabela 2.1. Relação entre os *concerns* e os *viewpoints*.

Viewpoints / ***Concerns***	**Veículo**	**Cliente**	…
Tempo de Resposta	✓	✓	…
Segurança		✓	…
…	…	…	…

Tabela 2.2. Relação de contribuição entre os *concerns*.

Concerns / ***Concerns***	**Tempo de Resposta**	**Segurança**	…
Tempo de Resposta		-	…
Segurança	-		…
…	…	…	…

A Figura 2.4 ilustra a especificação da composição do assunto Tempo de Resposta. Aqui, o requisito 1 de Tempo de Resposta deve ser "forçado" entre (*Constraint action="**enforce**" operator="**between**")* os requisitos 1 e 2 de Veículo, e o resultado *(Outcome action = **"satisfied"***) é a satisfação do requisito 1 de Gizmo, incluindo os seus filhos (*children="**include**"*).

```
<?xml version="1.0" ?>
- <Composition>
  - <Requirement aspect="Tempo de Resposta" id="1.1">
    - <Constraint action="enforce" operator="between">
        <Requirement viewpoint="Veículo" id="1" />
        <Requirement viewpoint="Veículo" id="2" />
      </Constraint>
    - <Outcome action="satisfied">
        <Requirement viewpoint="Gizmo" id="1" children="include" />
      </Outcome>
  </Requirement>
- </Composition>
```

Figura 2.4. Composição de Tempo de Resposta com *veículo*

Se forem detectados conflitos entre dois *concerns* que afectam o mesmo *viewpoint*, pesos (com valores entre zero e um) devem ser dados a cada *concern* num mesmo *viewpoint*, como é ilustrado na Tabela 2.3. No exemplo há um conflito entre Tempo de Resposta e Segurança (como é visível na Tabela 2.2) o qual afecta o *viewpoint* Cliente. Da Tabela 2.3 pode afirmar-se que o *viewpoint* Cliente possui um maior impacto do *concern* Segurança do que do *concern* Tempo de Resposta.

Tabela 2.3. Relação dos *concerns* com os *viewpoints*.

Viewpoints / ***Concerns***	**Veículo**	**Cliente**	…
Tempo de Resposta	✓	0.8	…
Segurança		1.0	…
…	…	…	…

No Anexo A apresentam-se os construtores existentes para a linguagem de composição. É uma linguagem que pode ser difícil de usar, em parte por ser puramente textual mas também por ter muitos operadores. Nesta dissertação pretende-se definir uma linguagem mais visual e mais simples, implementada com uma Linguagem de Domínio Específico. Linguagens de Domínio Específico são descritas no Capítulo 3.

2.3.1. Ferramenta ARCaDe

A ferramenta ARCaDe (*Aspectual Requirements Composition and Decision*) (Rashid *et* al., 2003) é uma ferramenta de suporte à abordagem AORE, usada para definir os requisitos dos *viewpoints* e dos aspectos, assim como as regras de composição, através de templates pré-definidos em XML. Estes módulos de encapsulamento de informação são posteriormente guardados num sistema de base de dados XML.

A ferramenta facilita a verificação e validação das regras de composição, certificando-se de que as regras se referem a *viewpoints*, aspectos e requisitos que existem na base de dados. Permite a modularização dos assuntos, *viewpoints* e regras de composição e é ainda capaz de identificar possíveis casos de conflito para sua posterior resolução.

Esta ferramenta utiliza XML, pois é evidenciada uma necessidade de definir acções específicas para assuntos e operadores de composição, para conseguir obter as regras de composição.

No mecanismo de composição desta ferramenta, os requisitos e os aspectos são referenciados através de um nome e de um identificador único dentro do seu âmbito. Cada *viewpoint* e aspecto possuem um nome único e engloba um conjunto de requisitos e sub-requisitos. Cada requisito possui um número de identificação único no âmbito do *viewpoint* ou aspecto que o contém. No entanto, os requisitos são inicialmente preparados para a composição, para que a qualquer requisito constante que seja de interesse para esta, lhe seja atribuído um identificador separadamente. Esta propriedade leva à existência de fragilidades neste processo, pois a adição ou remoção de requisitos pode alterar os seus identificadores, dando origem a requisitos que coincidem de forma indesejada. Simultaneamente, ao assumir que os requisitos são imutáveis ou não reutilizáveis, o uso de quantificadores pode fazer coincidir requisitos adicionados recentemente ou continuar a tentar fazer a correspondência com requisitos já removidos, comprometendo a integridade da análise de domínio (Rashid *et* al., 2003).

A ferramenta usa apenas especificações em XML que não fáceis de ser entendidas rapidamente por analistas de sistemas, onde a utilização de diagramas é mais aceite.

2.4. Outras abordagens

2.4.1. AORA

A abordagem AORA (*Aspect-Oriented Requirements Analysis*) (Brito e Moreira, 2003), tem como objectivo oferecer métodos para identificar, modularizar, representar e compor

assuntos. É uma das primeiras abordagens de requisitos com aspectos, e possui essencial destaque pela integração de requisitos funcionais com requisitos não funcionais.

Trata-se de um método simétrico, uma vez que aborda os assuntos transversais e não transversais da mesma forma. A AORA introduz o conceito *match point*, que representa os pontos onde são introduzidas as composições: um *match point* é um ponto num sistema onde um ou mais assuntos podem necessitar de ser compostos. É composto por um conjunto de assuntos que necessitam de estar juntos. Um desses assuntos possui o papel de assunto base com o qual o comportamento dos outros assuntos deve estar relacionado (Brito, 2008). A relação entre os assuntos é expressa através de uma tabela bidimensional de "Assuntos vs Assuntos Necessários".

Este é um modelo iterativo e incremental, na medida em que os assuntos são especificados nas fases seguintes e, a qualquer momento que surja um novo assunto, o ciclo volta ao início.

A fase inicial da AORA é a identificação de assuntos, em que se elicitam os requisitos e se reutilizam os catálogos. A fase seguinte consiste em especificar os assuntos, em que se identificam as responsabilidades, as contribuições entre assuntos, as prioridades, os assuntos requeridos e se constróem os modelos de assuntos. Na terceira fase, identificam-se os *match points* e os assuntos transversais.

2.4.2. Theme

Para antecipar a identificação de aspectos, surge a abordagem Theme, que visualiza as relações entre os comportamentos num documento de requisitos (Clarke e Baniassad, 2005).

A abordagem Theme (Clarke e Baniassad, 2005) identifica aspectos através de verbos - acções. É utilizada ao nível da Engenharia de Requisitos, quando é possível efectuar uma análise sintáctica da documentação inicial dos requisitos do sistema. Trata-se de uma abordagem que resolve essencialmente os assuntos transversais funcionais. Os assuntos não funcionais, dado que por vezes não se encontram na forma de verbo, podem ser reescritos, para ser possível a identificação das acções.

Um *Theme* é uma acção, ou seja, um elemento de desenho: um aglomerado de estruturas e comportamentos representativos de uma funcionalidade do sistema (Clarke e Baniassad, 2005). Quando se fala em *Theme,* não se deve considerar um sinónimo de aspecto. Os *Themes* são mais gerais e abrangem de forma mais estreita os assuntos, tratando-se de encapsulamento dos mesmos (Sommerville, 2007).

O processo Theme contempla três fases: Análise, Desenho e Composição. Na primeira fase, é feita uma análise com via a identificar os temas. A fase de Desenho consiste em desenhar *Themes* usando UML. Na última fase é especificado o modo como os *Themes* são combinados ou relacionados.

2.4.3. MATA

O MATA é uma notação e ferramenta que utiliza essencialmente diagramas de classe, sequência e de estados. Torna-se diferente de algumas abordagens já referidas, na medida em que não utiliza *joinpoints* explícitos. Nesta abordagem, cada modelo pode ser um *joinpoint* e a composição é um caso especial de transformação de modelos (Whittle e Jayaraman, 2007). A vantagem desta abordagem é conseguir lidar com a identificação precoce de erros, inconsistências ou ambiguidades.

Na abordagem MATA existe sempre uma composição entre um aspecto e uma base. Estes são antes da composição, individualmente e geralmente representados por um diagrama de sequência UML, e compostos por mecanismos de transformações de grafos. Dado que a problemática é a base ser atravessada por um aspecto, aplicando as regras de composição MATA, a base e o aspecto formam uma modelo composto. As regras r: LHS -> RHS definem o padrão esquerdo, indicando os *pointcuts*, ou seja, os pontos onde o novo modelo deve ser adicionado (Whittle e Jayaraman, 2007). Assim, o aspecto deve identificar os pontos na base onde o comportamento deve ser mudado e deve especificar o comportamento a ser inserido. O modelo direito define os novos elementos a serem adicionados e como o deve ser feito no modelo base (Whittle e Jayaraman, 2007).

A Figura 2.5 ilustra a aplicação da abordagem MATA através de diagramas de sequência, dado ser a forma mais comum. Nesta imagem, (a) representa a regra R1, (b) representa a regra R2 e (c) representa a composição final. A base é o diagrama (d) e o aspecto é o diagrama (e). Através da aplicação da regra R1 obtém-se a composição (e). A segunda composição tem como base o diagrama (d) que através da regra R2 e R1 vai dar origem à composição (c).

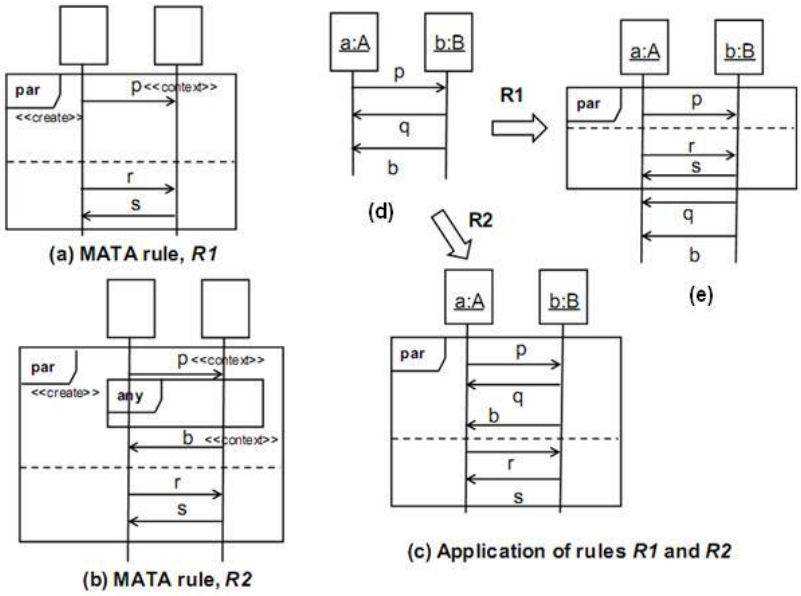

Figura 2.5. Modelo MATA (Whittle e Jayaraman, 2007)

2.4.4. Abordagens de *use-cases* com Aspectos

Por *use-cases* (Jacobson, 1987) entende-se a especificação de requisitos de um sistema através de modelos, usando actores, relações entre eles e relações entre os próprios *use-cases*. Tal como nas abordagens referidas anteriormente, também eles podem atravessar outros *use-cases* e criar casos de conflitos. Desta forma, pode ser incluído novamente o conceito de aspecto com vista a resolver os casos de atravessamento.

Em (Jacobson e NG, 2004), é proposta uma relação entre aspectos e *use-cases*, em que um *use-case* representa um aspecto, e é utilizado um mecanismo de separação e composição de assuntos. Nesta abordagem, os aspectos são separados em *peers* e extensões. Os *peers* são diferentes entre si, pelo que nenhum deles é mais importante que um outro, nem necessita dele para existir. Por outro lado, as extensões são serviços ou características adicionais que são definidas no topo de uma base.

Com a finalidade de obter os assuntos separados, deve-se modelar e estruturar os assuntos, recorrendo-se à utilização de *use-cases* durante a fase de requisitos e análise. No entanto, a dada altura do desenvimento de software, é necessário compor os assuntos separados. Para tal, é usada a unidade modular *use-case slice* que contém os elementos que são específicos à realização de um *use-case*. Assim, para cada *use-case* existirá um *use-case slice* ortogonal ao modelo de desenho, que pode ser adicionado ou retirado, consoante se pretende adicionar ou remover o seu conteúdo do topo do modelo.

Na abordagem *use-cases* com aspectos (Araújo e Moreira, 2003), existem 5 fases fundamentais. Nestas fases são identificados os requisitos funcionais, tal como se fazia nos modelos de *use- cases* e RNFs. A fase final consiste em identificar os assuntos candidatos a aspectos. Isto acontece em casos onde um *use-case* esteja relacionado com mais do que um caso de uso que seja um candidato a aspecto. Um *use-case* é um aspecto se restringe, estende ou é incluído por mais que um *use-case* (Araújo e Moreira, 2003). A problemática desta abordagem é a quantidade de casos de uso identificados que pode ser excessiva, complicando o diagrama de casos de uso e assim a compreensão do sistema.

2.5. Sumário

Neste Capítulo descreve-se o Desenvolvimento de Software Orientado a Aspectos, e os conceitos que este envolve, tal como *concerns*, *crosscuting concerns*, *scattering*, *tangling* e *join points*. Com o objectivo de contextualizar e dar a conhecer melhor o DSOA, são colocadas as suas vantagens e desvantagens em tópicos.

Posteriormente, aborda-se o tema Engenharia de Requisitos Orientada a Aspectos, onde é explicada a abordagem AORE, a sua origem, as suas fases, e a ferramenta principal que lhe dá auxílio (ARCaDe). Para uma maior compreensão da metodologia, é aplicada a técnica AORE ao caso de estudo Smart Home.

Pretendendo fornecer um conhecimento mais abrangedor da EROA, foram brevemente introduzidas outras abordagens orientadas a aspectos.

3. LDE

Uma Linguagem de Domínio Específico (LDE), também designada de forma não consensual de "micro linguagem e linguagem pequena", é uma linguagem focada em domínios específicos, geralmente pequena e declarativa, que apresenta um elevado poder de expressão (Deusen *et* al., 2000). Este tipo de linguagem permite especificar software e/ou outros sistemas, modelando e exprimindo de um modo formal os conceitos chave de um determinado domínio.

Um domínio é uma área de interesse restrita, um ponto de vista concreto ou uma família de produtos relacionados, caracterizada por um conjunto de terminologias e conceitos (Thibault, 1998).

O principal objectivo de uma LDE é oferecer soluções que, aquando da especificação do problema, reduzem a sua abstracção (Thibault, 1998), permitindo aos peritos uma fácil compreensão, validação, modificação da linguagem e até mesmo desenvolver outras LDEs. A abstracção é portanto a característica principal de uma LDE, pois permite uma compactação de conceitos e uma redução à essência do problema, contrariamente a outras linguagens como Java, entre outras.

As LDEs são essencialmente usadas em programas de pequeno porte, que pretendem ter uma proximidade a uma determinada área, reduzindo assim a distância entre o espaço do problema e a linguagem utilizada para exprimir a solução (geralmente linguagens de propósito geral). As LDEs têm sido bastante usadas, como foco essencial juntamente com as Transformações de Modelos, no Desenvolvimento Orientado a Modelos (*Model-Driven Development* (MDD)) (Stahl e Volter, 2006).

3.1. Desenvolvimento Orientado a Modelos

Ao longo das últimas décadas, vários têm sido os esforços para tornar as técnicas orientadas a modelos numa forma de melhorar o desenvolvimento de software. Estas técnicas, de forma simples e clara, pretendem que o sistema seja desenhado através de modelos, de forma a se obter o seu aspecto ou estrutura global. Este diagrama ou modelo é posteriormente utilizado para implementar o sistema.

Um modelo (Bézivin, 2005), de uma forma mais concreta, é a representação simples e abstracta da estrutura de um sistema, funcionalidade ou comportamento, devendo ser visual ou gráfico e declarativo. Trata-se de informação a um particular nível de abstracção da realidade, capaz de expressar a qualidade ou as características de um objecto ou instância sem no entanto se focar em nenhum deles. Um modelo possui elementos e ligações entre os elementos e cada elemento ignora os detalhes que não são relevantes para o ponto de vista em questão e encapsula e descreve os detalhes relevantes. Estes elementos são relacionados posteriormente através de ligações, consoante a relação que possuem entre si. Estas relações geralmente possuem uma identificação, e a cada elemento é dado um nome distinto.

A abstracção é portanto a característica fundamental de um modelo, pois vem resolver o problema da elevada complexidade e duplicação (Bézivin, 2005).

Uma LDE suporta um desenvolvimento orientado a modelos pois numa das primeiras fases do seu desenvolvimento é criado um modelo de forma rigorosa que é processado por máquina para produzir modelos de baixo nível de abstracção, sendo assim utilizados no processo de automação do desenvolvimento de software.

O modelo para representar uma LDE é denominado metamodelo e consiste na descrição dos conceitos existentes no domínio, que podem ser usados na linguagem. É usado para se referir a um tipo de metadados. Trata-se de uma definição das regras e das construções necessárias à criação de modelos sintácticos, utilizados também para compreender e analisar sistemas. São assim fundamentais para impelir, induzir, guiar e restringir o comportamento da LDE (Bézivin, 2005). Este tipo de modelo é equiparado ao código de um programa executado numa GPL (*General Purpose Language)* no sentido em que a implementação final é gerada a partir dele.

3.2. Vantagens e Desvantagens de uma LDE

Uma linguagem LDE permite soluções que podem ser expressas no idioma e ao nível de abstracção do domínio do problema. Deste modo os peritos conseguem compreender, validar, ou até mesmo desenvolver LDEs. Permite assim a validação e a optimização ao nível do domínio. As alterações ao nível dos requisitos também são rapidamente feitas pelo perito, pois basta proceder à alteração da parte do modelo afectada pela mudança (Deusen *et* al., 2000).

Os programas elaborados com LDEs são concisos, documentam-se e podem ser reutilizados para diferentes propósitos. Isto facilita o seu desenvolvimento ao nível de perda de tempo na realização da documentação e permite a compreensão imediata do programa.

Devido à elevada facilidade de uso e à programação considerada simples e reduzida, as LDEs aumentam a produtividade, confiança, manutenção e portabilidade. Tornam-se também facilmente construídas por alguém que não seja necessariamente um programador (*end-users*). Deste modo, aumentam a capacidade de teste seguindo determinadas abordagens (Deusen *et* al., 2000).

As LDEs dão forma a um domínio de conhecimento, e disponibilizam a conservação e a reutilização deste conhecimento (Deusen *et* al., 2000).

Apesar de ser uma linguagem que possui todas as vantagens apontadas anteriormente, existem também algumas desvantagens no seu uso e desenvolvimento.

Uma das áreas mais afectadas é a área financeira. Os custos associados ao desenvolvimento de uma LDE são geralmente elevados, pois o desenho, a implementação, a manutenção e a instrução dos utilizadores das LDEs geralmente ultrapassam os custos inicialmente previstos. Daí que apenas 16% dos trabalhos planeados consigam chegar ao fim, cumprindo o calendário proposto e com o orçamento previsto, 31% dos projectos são cancelados e 53% possuem custos maiores que os planeados (Deusen *et* al., 2000).

A reduzida informação disponível acerca de como e quando desenvolver uma LDE é também uma desvantagem deste tipo de linguagem, e várias são as questões que nessa área continuam por obter resposta (Deusen *et* al., 2000).

Uma LDE acaba por ser uma linguagem de disponibilidade limitada, dado que se foca apenas no domínio pretendido, domínio este que por vezes se torna difícil de definir.

As linguagens LDEs possuem ainda dificuldade em encontrar um equilíbrio com as GPLs, pois são linguagens bastante diferentes, geralmente não *Turing Complete*. Assim, pode dizer-se que em alguns casos, existe uma potencial perda de eficiência, quando comparando software feito com LDEs com software feito com GPLs (Deusen *et* al., 2000).

3.2.1. LDE versus GPL

Quando comparamos uma Linguagem de Domínio Específico com uma Linguagem de Propósito Geral, várias são os aspectos que se podem discutir.

Uma LDE reduz o domínio e a experiência necessária para programar, oferecendo uma maior facilidade de uso e expressividade. Contrariamente às GPLs, as LDEs podem não ser executáveis e as suas técnicas de desenvolvimento são mais variadas. De uma forma geral, as LDEs não são eficientes fora do ambiente para o qual foram especificadas, no entanto, as GPLs, por serem linguagens genéricas, fornecem soluções gerais para um vasto âmbito de problemas, ainda que apenas uma delas seja considerada a melhor solução (Deusen *et* al., 2000).

3.3. Sintaxe e Semântica

3.3.1. Sintaxe (abstracta e concreta)

Uma linguagem é definida através de um conjunto de sequências de caracteres de um determinado alfabeto. A sua sintaxe (Harel e Rumpe, 2000) especifica quais as sequências que a linguagem contém, através da definição dos seus conceitos ou palavras. A sintaxe é dividida em sintaxe concreta e abstracta. A sintaxe abstracta define os conceitos da linguagem, as relações, os atributos e as regras de boa formação. Consiste no metamodelo da linguagem, que pode ser obtido a partir da sintaxe concreta. A sintaxe concreta é definida através de um conjunto de regras representadas por uma gramática livre de contexto, geralmente EBNF, tratando-se de um documento informal. Pode ser textual, utilizando-se expressões regulares e gramáticas, ou visual/diagramática, sendo representada através de meta-modelação com, por exemplo, modelos UML, regras OCL, e gramáticas de grafos. Pode afirmar-se que uma sintaxe possui apenas uma semântica.

3.3.2. Semântica

A semântica é dividida no domínio semântico e no mapeamento semântico. O conceito mapeamento semântico relaciona os conceitos sintácticos com os conceitos do domínio

semântico, mapeando cada criação sintáctica num elemento semântico (Harel e Rumpe, 2000).

A semântica estática relaciona-se apenas com o significado de uma linguagem durante a sua execução, ou seja, descreve as características de um programa válido ou sintacticamente correcto. Geralmente, a semântica estática é definida através de um sistema de inferência de tipos.

3.4. OCL

O OCL (*Object Constraint Language*) (Warmer e Kleppe, 1998) é uma linguagem tipada, cujas expressões dependem de tipos definidos nos esquemas, classes, interfaces, entre outros. O seu objectivo é estabelecer restrições em metamodelos ou diagramas de classes que de outra forma não se conseguem impor. Permite a definição de consultas, valores de referência e estado das condições e regras. Geralmente utiliza-se esta linguagem, pois é uma forma de estabelecer as restrições que não se conseguem impor através de modelos. É usado para especificar invariâncias entre classes e tipos num diagrama de classes, especificar invariâncias de tipo para esteriótipos, descrever pré- e pós-condições sobre operações e métodos, especificar restrições sobre operações e guardas de transições. O uso do OCL dá origem a modelos mais consistentes e fiáveis.

Um exemplo de uma restrição OCL sobre uma classe *Cliente* que possui o atributo *idade* pode ser:

context Cliente

inv: idade >= 18 and **self**.idade < 100

Neste exemplo o contexto é a classe Cliente, e a restrição incide sobre o atributo idade, sendo que um cliente tem que possuir no mínimo 18 anos, e no máximo 99 anos.

3.5. Desenvolvimento de uma LDE

O desenvolvimento de uma LDE pode ser feito através de um sistema de desenvolvimento de linguagens ou *toolkit*. Estes sistemas possuem os mesmos princípios de funcionamento, mas revelam diferentes graus de capacidade. Contudo, todos eles possuem a mesma finalidade, sendo esta gerar uma ferramenta para uma linguagem descritiva. As ferramentas podem ser um verificador e interpretador, um ambiente de desenvolvimento integrado (IDE), entre outros.

Abaixo encontram-se as quatro fases que compõem o desenvolvimento de uma LDE, sendo elas a Análise de Domínio, o Desenho da Linguagem, e a Implementação.

3.5.1. Análise do domínio

Uma LDE inicia-se efectuando-se uma análise de domínio. Desta fase, faz parte a análise da similaridade, das variações e das combinações. É habitual fazer esta análise de uma forma informal, no entanto é mais correcto efectuá-la formalmente, como é explicado mais abaixo.

Existem diversos métodos para representar formalmente a análise de domínio. Exemplos destes métodos são DARE (Frakes *et* al., 1998), DSSA (Taylor *et* al., 1995), FAST (Weiss e Lay 1999), ODE (Falbo *et* al., 2002) e ODM (Simos, 1996).

O método mais utilizado é o FODA (*Feature-Oriented Domain Analysis)* (Kang *et* al., 1990). Este método consiste em criar um modelo de características (*feature model*) constituído pelas características (*features*), ou propriedades do sistema que são relevantes. Estas características podem ser obrigatórias ou variáveis.

Um modelo de características é constituído por um diagrama de características, pela sua definição semântica, regras de composição das características e alguma lógica na sua escolha. A Figura 3.1 é um exemplo de um modelo de *features*.

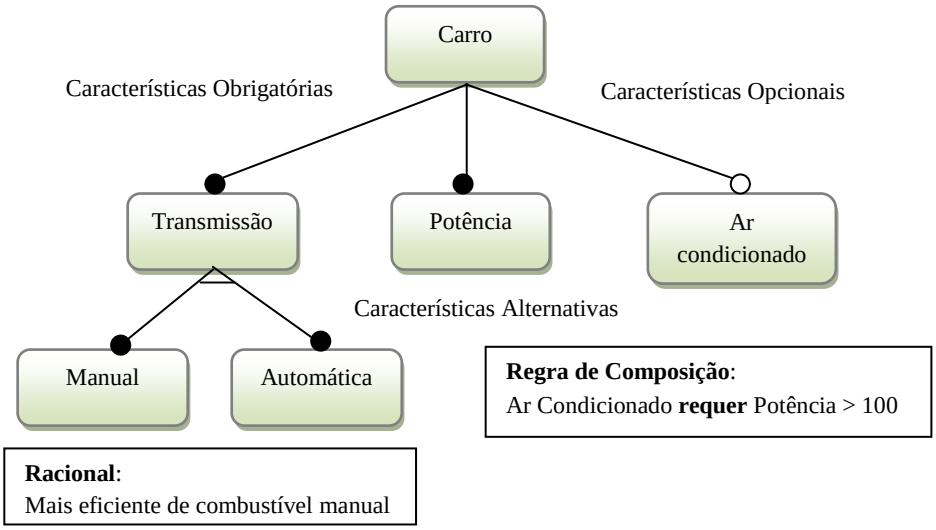

Figura 3.1. Modelo de *features* de Carro

O *input* da análise de domínio está focada na Figura 3.2, e o seu *output* é toda a terminologia e conceitos do domínio em estudo, uma lista de variação indicando a informação necessária para a especificação de uma instância do sistema, e a definição das características em comum, ou seja os modelos e primitivas da linguagem (Figura 3.3).

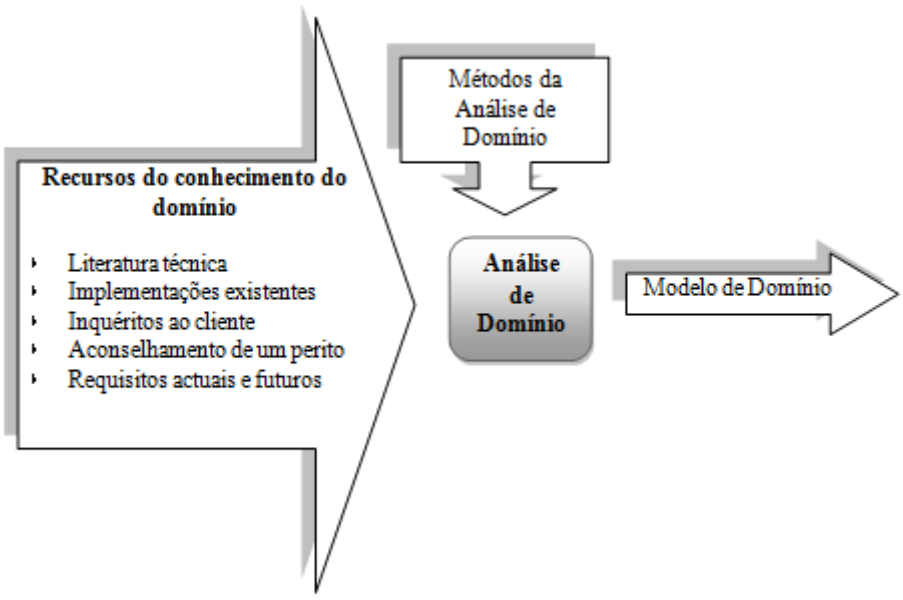

Figura 3.2. Análise de Domínio

3.5.2. Desenho da Linguagem

A forma mais simples de desenhar uma LDE consiste em baseá-la numa notação formal ou informal previamente existente no domínio e amplamente usada pelos seus *experts*. Deste modo, a implementação fica mais facilitada, e a familiaridade dos utilizadores, em alguns casos, aumenta. No entanto, nem sempre existem LDE's no domínio desejado, e os utilizadores da nova linguagem podem não ser os mesmos que realizaram a linguagem base (Mernik *et* al., 2005).

O desenho de uma linguagem (Figura 3.3) pode ser formal ou informal. O desenho informal é feito geralmente em linguagem natural, enquanto o desenho formal é feito através de métodos de definição da semântica. Estes métodos incluem a definição de expressões regulares e gramáticas para a especificação sintáctica da LDE. Existem ainda gramáticas de atributos, sistemas de reescrita e máquinas de estados abstractas para se obter a especificação semântica da linguagem.

De salientar que uma LDE pode ser visual, textual ou ambas as opções.

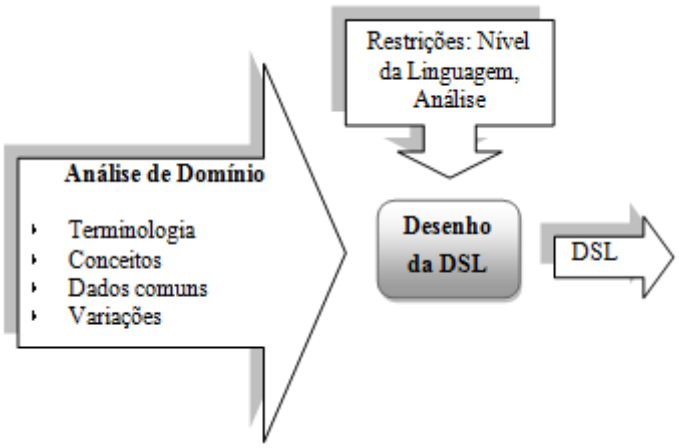

Figura 3.3. Desenho da Linguagem

3.5.3. Implementação

A implementação de uma LDE pode ser feita de três formas:

- Através de uma gramática que implemente a semântica pretendida, e um *parser* capaz de efectuar a tradução;
- Utilizando uma linguagem base e definir as suas expensões (como no caso do TCL/TK ou Perl);
- Utilizando XML através da utilização de *tags* que representam os elementos do domínio.

3.6. Avaliação de uma LDE

A avaliação de uma LDE consiste num processo composto por fases (Reisner, 1981), como ilustra a Figura 3.4. O processo é iniciado na fase Recrutamento de Assuntos. Nesta fase, os assuntos são agrupados em categorias claras, e são construídos grupos de dez utilizadores. A fase Preparação das Tarefas consiste na organização da avaliação, através da elaboração de tarefas e testes capazes de obter os resultados esperados. A tarefa Sessão Piloto é como que uma simulação de um exame. É obtido o material de treino, confirmada a organização dos processos de avaliação e são controladas as restrições de tempo e outras variáveis externas, tais como equipamento apropriado, para que não haja interferências com os resultados e para que as condições simulem um ambiente real. A Sessão de Treino é a primeira fase da avaliação propriamente dita. Nesta fase, é dada uma sessão de formação na qual se apresenta a LDE. Testes de compreensão e revisão podem ser introduzidos nesta fase, e algum material (slides ou exercícios) deve ser realizado e distribuído de forma a auxiliar os utilizadores. Segue-se a fase de exame. Nesta fase, são realizadas questões envolvendo actividades de

escrita da linguagem. Deve haver uma observação do questionário e registo das actividades em curso por parte do supervisor, anotando os tempos de execução e as taxas de erros obtidas. O objectivo é obter o grau de facilidade em aprender e utilizar a linguagem em avaliação. Deve também ser feito um questionário de auto-avaliação com via a captar o sentimento de correcção das respostas dos próprios utilizadores.

Após cada grupo ter feito a sessão de formação e o exame, os utilizadores que participaram são convidados a fazer uma reunião de balanço sob a forma de questionário, para classificar preferências e obter comentários. O objectivo deste questionário é obter uma reacção global à linguagem, taxa de facilidade do uso de aspectos específicos e outros possíveis comentários informais que possam ser dados com a finalidade de conseguir melhorias.

Finalmente efectua-se uma análise qualitativa e quantitativa dos resultados. Nesta fase é considerada a eficácia e a precisão do utilizador através das observações dos erros produzidos. É também feita uma medida da eficiência através da medição dos tempos necessários para a realização das tarefas.

Por fim é avaliada a satisfação do utilizador através das suas respostas aos questionários e é inferida a adequação ou necessidade de melhoria da LDE.

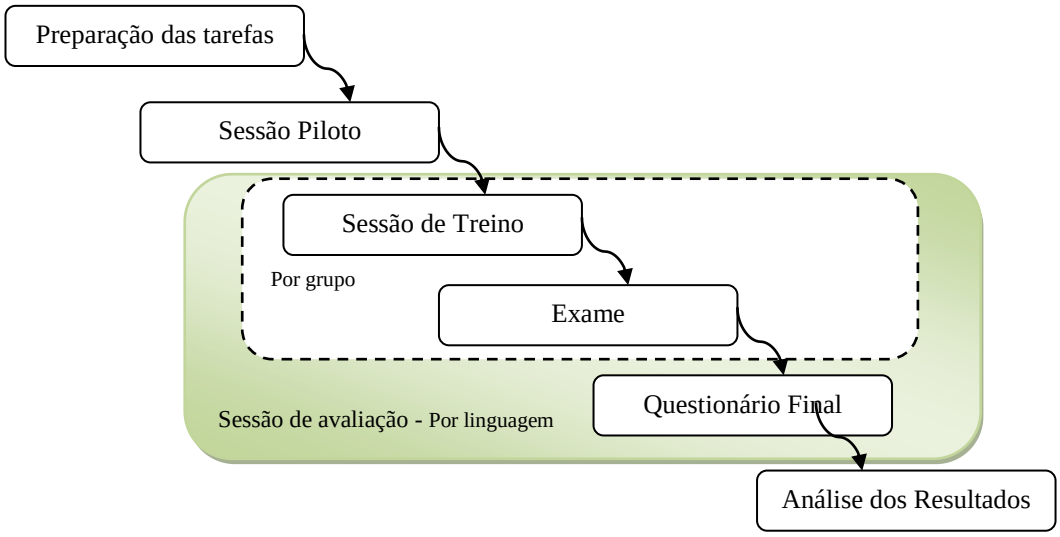

Figura 3.4. Fases da avaliação de uma LDE

O processo descrito é o processo formal utilizado para avaliar uma LDE de uma forma empírica (ou também dita experimental). Apesar de se tratar de um processo essencial para validar a linguagem, normalmente estas avaliações não são feitas com tanto rigor (Gabriel *et* al., 2010).

3.7. Ferramentas de suporte para construir LDEs

Existem diversas ferramentas para proceder ao desenvolvimento de uma LDE. Algumas são mais antigas tal como a Draco (Neighbors, 1984), ASF+SDF (Brand *et* al., 2001), Kephera (Faith *et* al., 1997), Kodiyak (Herndon e Berzins, 1988), IrffoWiz (Nakatani e Jones, 1997) entre outras, encontrando-se desactualizadas. Presentemente são utilizadas as ferramentas abaixo enumeradas.

3.7.1. GME

O GME (*Generic Modeling Environment*) (Ledeczi *et* al., 2001) é um pacote de ferramentas configurável para modelação de domínios específicos através de um ambiente visual. Suporta várias configurações baseadas num conjunto de conceitos implementados pelo próprio GME, e cada uma delas é acompanhada por um metamodelo que especifica o paradigma do domínio da aplicação a modelar. O paradigma engloba a sintáctica, a semântica, e toda a informação sobre o domínio (conceitos, relações entre eles, e outros).

Os principais conceitos usados para definir um paradigma de modelação são Pastas, FCOs (Modelos, Átomos, Conjuntos, Referências e Relações), Regras, Restrições e Aspectos. As linguagens de modelação são compostas por instâncias destes conceitos.

O metamodelo é construído no próprio ambiente da ferramenta, baseado em UML. As definições sintácticas são modeladas por diagramas de classes, e a semântica estática é definida com restrições, recorrendo ao uso de OCL. O metamodelo é usado para gerar automaticamente o ambiente de domínio específico desejado. Este novo ambiente será utilizado para construir modelos, e obter as aplicações desejadas, pelo processo *model interpretation*.

O GME possui uma base de dados onde guarda os modelos, sendo que quando um modelo é criado, é guardado e visto como um novo tipo ou classe, que pode posteriormente ser instanciado tantas vezes quantas as que o utilizador desejar.

Este pacote de ferramentas possui uma estrutura modular, baseada numa arquitectura de componentes. A componente principal é a *Core* que implementa duas outras componentes fundamentais, Meta e MGA. A Meta define os paradigmas de modelação e a MGA implementa os conceitos para o paradigma dado.

3.7.2. DSL Tools

O pacote de ferramentas Microsoft DSL Tools (Cook *et* al., 2007) é uma solução que permite definir LDEs gráficas e geradores de código. Trata-se de um *workbenck* que faz parte do *Visual Studio SDK 2005* e que permite uma definição diagramática através de uma formatação XML, que sem qualquer programação manual, dá origem ao código necessário à implementação dos modelos gráficos da LDE.

O utilizador define os conceitos da linguagem, o modo usado para processar esses conceitos no ambiente de edição diagramático e os componentes auxiliares usados para carregar e salvar o modelo para integrar a nova ferramenta no *Visual Studio* e gerar o código e os outros artefactos dos modelos criados com a ferramenta.

No DSL Tools, as LDEs definidas são usadas para criar modelos que servem de *input* aos geradores de código. A geração de código nesta ferramenta é facilitada em relação a outras ferramentas utilizadas para o mesmo efeito, pois existem recursos de abstracção oferecidos pelas linguagens actuais, como a disponibilização e verificação de tipo estático, herança, funções virtuais e classes parciais. A linguagem C# é o exemplo de maior realce, pois permite a compilação e junção de uma classe definida em vários pacotes, numa única classe. O pacote DSL Tools permite uma evolução inicial facilitada, uma implementação simples, oferece a possibilidade de utilizar mecanismos de compilação, vinculação e depuração de código. Também a personalização do código é facilitada pois é directa. Com o DSL Tools, a interpretação directa dos modelos dispensa a fase de compilação, o que torna a troca dos modelos antigos pelos novos, uma tarefa directa. Isto acontece sempre, mesmo quando o sistema se encontra em execução (Pelechano *et* al., 2006).

3.7.3. EMF/GMF

A plataforma EMF[1] (*Eclipse Modeling Framewok*) (Budinsky *et* al., 2003**)** é uma ferramenta de modelação e geração de código que permite a construção de ferramentas e outras aplicações baseadas em modelos de dados estruturados. O seu objectivo é fornecer um editor, e para tal, é necessário fornecer-lhe como input, uma especificação de um modelo em XMI.

Um padrão XMI ou XML *Metadata Interchange* é um standard OMG (*Object Management Group*) para troca de informaçã de metadados baseada em XML (*eXtensible Markup*

[1] Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/?project=emf,,2004.

Language). Este padrão é essencialmente utilizado para fazer a troca de metadados entre as ferramentas baseadas em UML (*Unified Modeling Language*) e MOF (*Meta-Object Facility*). Assim torna-se facilmente possível salvar modelos UML em formato XML e fornecer formas para mover modelos UML entre ferramentas.

O EMF fornece ferramentas e tempo de execução para produzir um conjunto de classes Java para o modelo, um conjunto de classes adaptadoras que permitem a visualização do modelo, um editor para o modelo baseado em comandos, e um editor básico. Esta ferramenta tem também a capacidade de fornecer a base para a interoperabilidade com outras ferramentas baseadas no EMF.

A ferramenta em foco possui três componentes fundamentais. Uma delas é a *framework* EMF CORE que inclui o metamodelo (ecore) e fornece tempo de execução e notificações das alterações do modelo assim como suporte com serialização XMI por defeito. Disponibiliza ainda uma API eficiente e reflexiva para manipular de forma genérica os objectos EMF.

Outro elemento fundamental do EMF é o *EMF.Edit***.** que possui classes genéricas reutilizáveis, que originam os editores correspondentes aos modelos fornecidos como input. Fornece vistas do modelo, ajuda a integrar os modelos e disponibiliza uma interface bastante atractiva ao utilizador (oferecendo, por exemplo, comandos para avançar e retroceder). Contudo, a componente fundamental do EMF, é o *EMF.Codegen*, pois é responsável pela geração do código necessário para o *core* e para a construção do editor a partir de um modelo dado.

O EMF suporta três níveis de geração de código:

- **Nível Modelo** - fornece classes de interface e de implementação para todos os elementos do modelo, e um pacote para os meta-dados;
- **Nível Adaptadores** - gera as classes de implementação capazes de adaptar as classes do modelo para serem editadas e visualizadas;
- **Nivel Editor** - responsável por produzir a estrutura básica do editor do EMF, que pode depois ser personalizada.

O GMF[2] (*Graphical Modeling Framework)* (Budinsky *et* al., 2003) é um *plug-in* do Eclipse (Eclipse, 2009), baseado no EMF (*Eclipse Modeling Framework*) e no GEF (*Graphical*

[2] GMF, Graphical Modeling Framework, http://www.eclipse.org/modeling/gmf/, 2004.

Editing Framework), e suporta uma infra-estrutura que fornece uma componente de geração e tempo de execução para desenvolver editores gráficos.

O GMF usa, como input, um metamodelo, e o seu processo de execução está descrito na Figura 3.5.

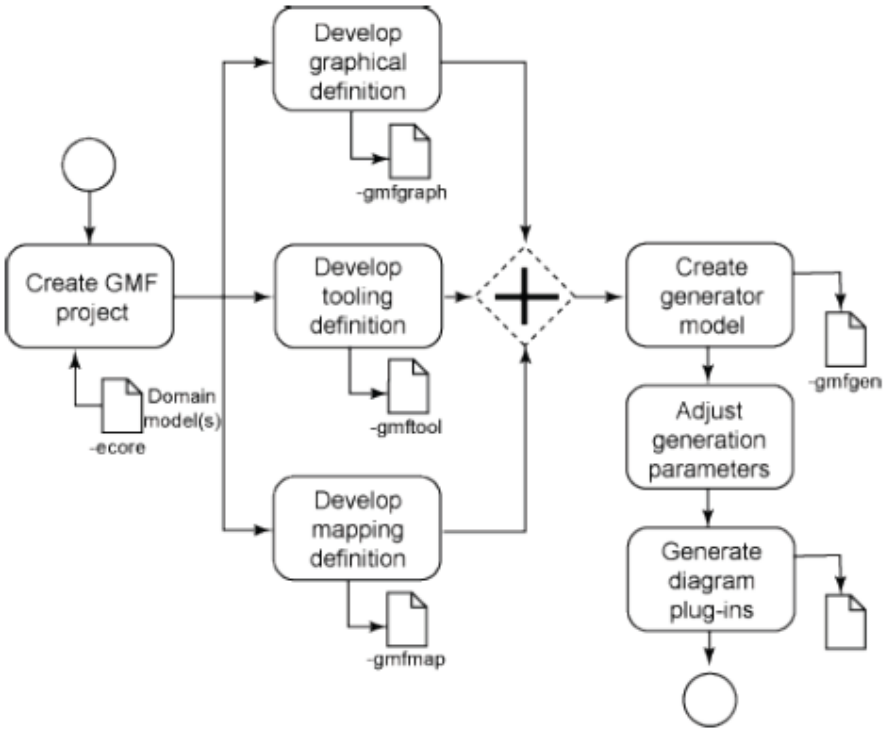

Figura 3.5. Processo GMF[3]

Para obter o editor visual gerado pelo GMF, é necessário seguir um procedimento regrado. Este processo é feito com o auxílio do *Dashboard*, que guia o utilizador de forma a efectuar o processo pela ordem correcta. Inicialmente importa-se o metamodelo desejado para se iniciar o processo de geração. No *Dashboard* selecciona-se o metamodelo, seguindo-se a derivação do modelo de domínio (*Domain Model*), modelo gráfico (*Graphical Model*) e finalmente do modelo de ferramentas (*Tool Model*). O modelo de domínio é um modelo conceptual que descreve as várias entidades envolvidas num sistema e as relações entre elas. O modelo gráfico descreve as componentes que vão constar de cada modelo feito no editor visual, tal como a sua aparência. Por fim, o modelo de ferramentas possui a definição de todas as ferramentas que vão estar disponíveis na palete do editor.

Com vista a estabelecer qual o elemento da palete que cria cada elemento do editor, procede-se à definição do *Mapping Model*. Este passo é feito através da instrução *combine*, e é neste modelo que se obtém o mapeamento entre os modelos gráfico e de ferramentas. No final deste

[3] IBM, http://www.ibm.com/developerworks/library/os-ecl-gmf/index.html, 2005.

processo, transforma-se o mapeamento obtido no modelo de geração do editor do diagrama, através da instrução *Transform*, e obtém-se o ficheiro de geração do editor do diagrama (*Diagram Editor Generation Model*). Após a obtenção deste, pode-se gerar o código do editor através da opção *Generate Editor Diagram*, e obter o *plug-in* do editor visual pretendido.

Actualmente existem dois novos *plug-ins* que oferecem suporte ao GMF, sendo eles, o Emfatic[4], e o EuGENia (EuGENia, 2008). O Emfatic engloba vários *plug-ins* do Eclipse que incluem um editor e um parser para a linguagem. Trata-se de uma linguagem para representar modelos EMF Ecore numa forma simples e textual. A sua vantagem é que consegue representar um modelo Ecore completo em apenas um ficheiro de código e utiliza uma sintaxe semelhante ao Java, que é familiar a vários programadores. Os *plug-ins* permitem ainda que um ficheiro Emfatic seja compilado num modelo Ecore e vice-versa.

A ferramenta EuGENia gera automaticamente e simultaneamente os modelos gmfgraph, gmftool e gmfmap necessários à implementação de um editor GMF, tendo como base um metamodelo Ecore. Esta fornece notações de alto nível capazes de facilitar a geração de editores. Algumas destas notações são:

- gmf – é aplicado ao pacote de topo e especifica que são esperadas anotações GMF nos elementos do modelo;
- gmf.diagram – especifica a classe raiz do metamodelo;
- gmf.node – especifica que a classe em questão é representada no editor como um nó. Suporta vários detalhes tais como *label*, *label.icon*, *figure*, *color*, *border.color*, *border.width*, *tool.name*, *tool.description*, entre outros;
- gmf.link – especifica que a classe em questão é representada no editor como um *link*. Suporta vários detalhes tais como *label.pattern*, *source*, *target*, *source.decoration*, *width*, *tool.name*, *tool.description*, entre outros;
- gmf.compartment – especifica que a referencia contida na classe vai criar um compartimento onde são colocados os elementos do tipo da referencia;
- gmf.label – especifica labels adicionais à classe em questão.

[4] Emfatic, http://wiki.eclipse.org/Emfatic/, 2008.

3.7.4. MetaEdit+

A ferramenta MetaEdit+ (Kelly *et* al., 2006) é a ferramenta do tipo *Meta-CASE* mais usada actualmente. Contrariamente aos *plug-ins* do Eclipse, ela já implementa graficamente o comportamento genérico CASE para os objectos e para as relações, incluindo o *Diagram Editor*, *Object* e *Graph Browsers*. O utilizador obtém os conceitos e símbolos através da ferramenta GUI, e tem apenas que especificar a sua linguagem de modelação. Nesta fase o utilizador pode, por exemplo, criar novos objectos, atribuir-lhe nomes, propriedades e relações. Assim, não há necessidade de efectuar código, pois esta ferramenta apenas segue a linguagem definida tal como o *Word* segue os seus templates.

A MetaEdit+ fornece também a opção de importar e exportar ficheiros XML, uma API para dados e controlo de acesso às suas funções e um gerador de código genérico. Como o gerador de código aceita qualquer linguagem de modelação, linguagem de código ou *framework* sobre a qual o código é executado, o utilizador tem toda a liberdade para produzir o melhor código possível para os modelos (Kelly *et* al., 2006).

A ferramenta descrita possui uma característica essencial que é a separação de assuntos, sendo perita em produzir as funcionalidades de uma ferramenta CASE, enquanto o utilizador é perito no seu domínio. Esta separação é inexistente nas ferramentas CASE de métodos fixos, e nas *frameworks* que suportam o código das LDEs (Kelly *et* al., 2006).

3.7.5. Outras

Outras ferramentas também utilizadas para produzir LDEs são AToM^3 (Lara e Vangheluwe 2002), e Meta Sketch (Nóbrega *et* al., 2006).

Segundo (Lara e Vangheluwe 2002), AToM^3 (*A Tool for Multi-Formalism and Meta-Modelling*) é uma ferramenta de meta modelação visual que suporta a modelação de sistemas complexos, ou seja, que possuem um elevado número de componentes, cuja estrutura e comportamento não pode ser descrito por um simples formalismo.

O AToM^3 tem como tarefas essenciais a meta modelação e a transformação de modelos, e funciona ao nível do metamodelo e do meta-metamodelo. O metamodelo é o resultado da meta-modelação e permite a construção de modelos válidos de acordo com um formalismo, sendo que o meta-metamodelo é usado para definir os próprios formalismos. A transformação de modelos refere-se ao processo automático de converter um modelo num dado formalismo, ou noutro modelo que pode ou não estar no mesmo formalismo.

Esta ferramenta possui diferentes formalismos, ao nível da meta modelação. A partir destes, é gerada uma ferramenta para processar os modelos descritos no formalismo especificado. Os modelos são representados através de grafos de sintaxe abstracta, e as operações que lhes são feitas (optimização de modelos, transformação de modelos, entre outros) podem ser expressas através de modelos de gramáticas de grafos.

Alguns exemplos de meta-metamodelos para esta ferramenta são ER, *Petri Nets* e DFD.

A componente principal do AToM^3 é o processador, pois é ele que manipula os modelos e gera o código para as ferramentas.

Outra ferramenta usada para modelação de domínios específicos é a Meta Sketch (Nóbrega *et* al., 2006). Trata-se de uma aplicação que permite o rápido desenvolvimento de linguagens de modelação, cuja tecnologia é baseada em OMG. É composta por três componentes, sendo estas o *editor*, o *merger* e o *designer*. O *editor* é a parte essencial da ferramenta, e é utilizada para definir tanto o modelo, como o metamodelo. O *merger* é utilizado no metamodelo com o objectivo de englobar todos os pacotes utilizados na definição do metamodelo em apenas um pacote. O *designer* encontra-se ainda em construção, e irá permitir a definição gráfica da sintaxe concreta da linguagem. Actualmente esta definição é feita em ficheiros XML.

3.8. Sumário

Este Capítulo aborda os assuntos LDE e Desenvolvimento Orientado a Modelos, tal como os conceitos que destes provêm. Estes conceitos são explicados, focando quando são podem ser utilizados e quais as suas mais-valias. Posteriormente são detalhadas as vantagens e desvantagens de uma LDE, e feita uma comparação entre estas e as GPL.

Neste Capítulo abordam-se ainda os conceitos Sintaxe e Semântica de uma linguagem, definindo-se cada uma delas, e dá-se a noção de OCL como uma linguagem capaz de impor restrições e complementar modelos ou linguagens.

Para que se compreenda melhor o processo de desenvolvimento de uma LDE, são detalhadas as fases que este abrange, sendo elas a Análise de Domínio, o Desenho da Linguagem, e a Implementação. É também descrito de forma pormenorizada o processo que se deve realizar para avaliar formalmente uma LDE.

Finalmente, são enunciadas várias ferramentas que podem ser utilizadas na definição de uma LDE, tais como GME, DSL Tools, EMF/GMF, MetaEdit+ e outras como AToM^3 e Meta Sketch.

4. Trabalho relacionado

Dentro da área científica em que se vai desenvolver esta dissertação, existem trabalhos desenvolvidos relacionados com o desenvolvimento de LDEs, com a Engenharia de Requisitos Orientada a Aspectos (EROA) e existe também uma elevada contribuição de trabalhos que aborda os dois temas em parceria.

4.1. LDEs para abordagens de requisitos

4.1.1. Uma Linguagem de Domínio Específico para a framework i*

*Uma Linguagem de Domínio Específico para a framework i** (Nunes, 2009) (Nunes *et* al., 2009) é um trabalho que implementa uma LDE capaz de lidar com a *framework i**.

A *framework i** é uma técnica de modelação de dados orientada para os objectivos. Trata-se de uma abordagem simétrica, que identifica requisitos não funcionais. Esta *framework* possui uma modelação ou uma representação dos seus elementos e das suas relações através de um metamodelo que é bastante incoerente. Desta forma, em programas específicos (e.g. OME) para trabalhar com a *framework*, o metamodelo dá origem a modelos com falhas e inconsistências. Para resolver estas incoerências, um estudo foi feito com vista a melhorar o metamodelo existente. Foi também objectivo do trabalho, fazer uma pesquisa de todas as ferramentas capazes de trabalhar com esta *framework*, descobrir as suas incoerências, objectivos e funcionalidades, para assim conseguir a construção de um metamodelo capaz de resolver as falhas existentes. Este metamodelo tem, como objectivo fundamental, gerir a escalabilidade dos modelos criados, e para tal, foi realizada a especificação de uma LDE.

O melhoramento do metamodelo da *framework i** foi resolvido através da construção de um novo metamodelo, e de novos mecanismos que dão auxílio à escalabilidade dos modelos gerados. Foi também introduzido o conceito de compartimento de forma a visualizar melhor as dependências entre os actores.

Os objectivos deste trabalho foram cumpridos, pois dele surgiu uma nova ferramenta capaz de implementar de forma total a *framework* em estudo, garantindo que todos os modelos são correctos e aceites de acordo com o metamodelo elaborado.

4.1.2. Uma Linguagem Específica do Domínio para uma abordagem orientada aos objectivos baseada em KAOS

A dissertação *Uma Linguagem Específica do Domínio para uma abordagem Orientada aos objectivos baseada em KAOS* (Dias, 2009) surge com o objectivo de melhorar a qualidade das especificações e da linguagem da metodologia KAOS.

O KAOS é um método da EROO que permite a definição de objectivos com diferentes níveis de abstracção e que possui um modelo conceptual para adquirir e estruturar os requisitos através de uma linguagem de aquisição. Este processo compreende as fases Elaboração de Objectivos, Modelação de Objectos, Modelação de Agentes e Operacionalização e compreende os níveis meta-nível, nível de domínio e nível de instância. O KAOS possui também um método para definição de modelos de requisitos baseado na mesma linguagem. Para tal serve-se dos conceitos Agente, Objecto, Conflito, Condição Fronteira, Obstáculo, Propriedades do Domínio, Acção, Restrição, Operação e Objectivo. Os modelos existentes são o modelo de Objectivos, Objectos, Responsabilidades e Operações.

Presentemente, existem algumas ferramentas, tais como a Dia e a Objectiver, que são capazes de modelar diagramas KAOS. No entanto estas ferramentas não são suficientemente escaláveis pois os modelos construídos através deste método tendem a ser complexos, devido ao aumento do número de elementos que o constitui. Não existe ainda nenhuma ferramenta capaz de verificar a consistência dos modelos relativamente à sua sintaxe. Em alguns casos essas ferramentas permitem erros de modelação permitindo a construção de modelos imprecisos.

Neste trabalho, procurou-se uma solução para os problemas acima focados. Assim, foi objectivo do mesmo a definição de um novo metamodelo para o KAOS adicionando alguns conceitos. De destaque é a definição do conceito *Compartimento* para lidar com o problema da escalabilidade.

Após a definição do metamodelo melhorado, obteve-se também uma LDE e um editor onde é possível especificar modelos KAOS. Foi portanto conseguida uma nova ferramenta capaz de lidar com os novos conceitos introduzidos e capaz de verificar a consistência sintáctica dos modelos. Também a introdução do conceito *Compartimento* foi muito bem conseguida, pois

dada a sua capacidade de colapsação, torna-se possível resolver o problema da escalabilidade dos modelos de tamanho elevado.

4.2. LDEs para Programação Orientada a Aspectos

4.2.1. Uma DSL para especificar a ordem de execução de aspectos

O objectivo da programação orientada a aspectos é a separação de assuntos, onde cada assunto pode ser desenvolvido por um perito no domínio do problema abordado por aquele.

Este trabalho (Marot e Wuyts, 2008) destaca a programação orientada a aspectos. É sabido que a programação orientada a aspectos (AOP) possui o problema da composição de múltiplos aspectos, que em geral é difícil de efectuar. Por este motivo, é apresentado um modelo em que os problemas da composição são considerados como assuntos transversais tratados por um aspecto composto, implementado utilizando uma linguagem de domínio específico para composição de aspectos. Esta linguagem é declarativa e permite trocar a ordem da composição durante o tempo de execução, especialmente se os aspectos compostos foram desenvolvidos isoladamente.

Quando vários *advices* executam no mesmo *join point*, é importante considerar a ordem pela qual eles executam, dado que uma ordem de execução errada pode mudar o objectivo do aspecto. Desta forma, podem ocorrer interacções indesejadas entre aspectos, que podem levar a programas mal construídos. É por estes motivos que este trabalho se centra na ordenação de *advices* à volta de um *join point*. É considerado que os problemas de composição aspectual devem ser tratados através de um aspecto composto.

O modelo que trata a ordem de execução dos *advices* em torno de um *join point* é uma abordagem baseada em regras de composição que declaram restrições na ordem de execução. As regras possuem uma prioridade e a que obtiver uma maior prioridade obtem precedência em caso de conflito. Estas regras são definidas nos próprios aspectos: se um aspecto necessita de ordenar os seus *advices*, então ele declara as suas próprias regras. Para lidar e gerir as restrições que atravessam os aspectos é então necessário criar um novo aspecto dedicado ao assunto de composição aspectual. É este aspecto que irá posteriormente declarar as regras sobre os *advices* dos diferentes aspectos.

A linguagem de domínio específico para composição de aspectos proposta neste trabalho é baseada nos conceitos regras de composição, restrições e contextos, que lidam directamente com a composição.

4.2.2. Outras DSALs

A linguagem TwisteR **(**Achencach e Ostermann, 2010) é um exemplo de uma DSAL com desenvolvimento dinâmico para Ruby (Thomas e Hunt, 2000). Ruby é uma linguagem tipificada e dinâmica que fornece características expressivas de meta programação. Os construtores da programação orientada a aspectos desenvolvidos desta forma facilitam a instanciação dinâmica dos aspectos e o desenvolvimento com estratégias de domínio expressivo.

JPPAL (Sousa e Sobral, 2010) é uma linguagem para programação paralela implementada à semelhança de uma biblioteca de aspectos reutilizáveis. Trata-se de uma linguagem implementada utilizando AspectJ, que fornece uma forma simples de exprimir paralelizações através de anotações Java.

A abordagem Linhas de Produto de Software Dinâmico utilizando Modelos Aspectuais em Tempo de Execução (Dinkelaker *et* al., 2010) utiliza modelos de *features* para descrever a variabilidade da DSLP, e uma linguagem de domínio específico para implementar de forma declarativa as variações e as suas restrições. É uma abordagem que combina vários conceitos da programação orientada a aspectos, tais como aspectos dinâmicos, modelos de aspectos em tempo de execução, e detecção e resolução de interacções aspectuais.

4.3. Outras LDEs

4.3.1. ALPH

Aplicações na área da saúde pervasiva utilizam as características da computação ubíqua para obter uma tecnologia avançada neste sector. Tal como várias outras indústrias, a saúde tem reconhecido os ganhos obtidos pela utilização das tecnologias.

Para desenvolver aplicações nesta área, várias dificuldades devem ser ultrapassadas, o que inclui tanto a computação ubíqua como problemas informáticos relacionados com a saúde. A incorporação dos assuntos da computação ubíqua requer a adopção de questões tais como a mobilidade e a sensibilização para o contexto. Estas questões são inerentemente transversais dado que toda a aplicação deve adaptar o seu comportamente em vários pontos da sua funcionalidade base.

As aplicações de saúde pervasiva têm sido geralmente desenvolvidas utilizando técnicas de programação tradicionais. Com a chegada de vários *middlewares* ubíquos e *frameworks*, o desenvolvimento computacional tem sido ajudado, apesar de ainda ser induficiente.

A linguagem ALPH (Munnelly e Clarke, 2007) foi elaborada com o objectivo de fornecer uma programação mais eficiente e dentro de um domínio específico. Construções de alto nível desempenham tarefas específicas ao domínio e fornecem um elevado nível de abstracção para os peritos. Isto leva a um código mais expressivo e facilita o desenvolvimento das aplicações. Também o uso de LDEs, em vez de linguagens de propósito geral (GPLS) reduz o esforço necessário para aprender os conhecimentos específicos exigidos ao perito, dado que as construções são mais intuitivas e representativas semanticamente. Esta nova linguagem ambiciona fornecer construções que representem tarefas ou entidades de domínio específico.

Neste trabalho, a linguagem ALPH foi mapeada numa linguagem de aspectos, AspectJ, e lida inicialmente apenas com as noções de aspectos, *pointcuts*, *joinpoints* e *advice*. Estra tradução é feita utilizando um tradutor dinamicamente extensível. O resultado destra tradução é um ou mais aspectos compilados, que são posteriormente inseridos na aplicação base em execução, nos pontos especificados pelos contructores da LDE. Como resultado, a aplicação corre com a funcionalidade de saúde pervasiva necessária incluída. Os aspectos podem conter código gerado através de uma biblioteca de aspectos existentes de assuntos transversais.

A LDE definida é capaz assim de identificar os assuntos transversais de domínio específico, o que elimina a necessidade de implementar os assuntos transversais através da aplicação base, aumentando a modularidade.

4.3.2. Mawl

Um serviço *Form-based* descreve o fluxo de dados entre o serviço e o utilizador através de uma sequência de perguntas e respostas de interacção ou *forms*. Um *form* fornece uma interface ao utilizador, que lhe apresenta dados do serviço, recolhe informação do utilizador e retorna dados ao serviço.

O objectivo deste trabalho consiste em conseguir programar estes serviços de forma independente e com um elevado poder de abstracção, separando o serviço lógico da descrição da interface do utilizador.

Uma LDE oferece uma solução mais completa para os problemas de engenharia de software, através das características abstracção e restrição que ela fornece. A escolha de uma abstracção apropriada facilita as fases de requisitos, desenho, implementação, e manutenção através da introdução de entidades de alto nível e relações que se enquadram de perto no domínio. A restrição da expressividade da linguagem permite uma análise automática e o suporte de verificação, modificação e manutenção.

A LDE Mawl (Atkins *et* al., 1999) foi assim construída com o objectivo de resolver problemas específicos da criação de serviços web dinâmicos: a falta de garantias de tempo de compilação sobre os serviços e o pequeno nível de programação envolvido em desenvolver programas CGI. Trata-se de uma linguagem para programar serviços baseados em *forms* num modo independente de dispositivos, que fornece auxílio a serviços *form-based* para web e telefone. A introdução do *form* obriga a que haja uma separação de assuntos entre o serviço lógico e a interface do utilizador.

O domínio desta LDE é todo o conjunto de serviços em que o utilizador interage com um computador remoto, com o objectivo de completar alguma transacção, o que pode envolver a pesquisa de informação, a selecção de opções ou itens, fornecer respostas a perguntas, entre outros. Um serviço pode ser tão simples como inserir um nome para obter um número de telefone ou tão complexo como ordenar um catálogo.

Neste trabalho é dado realce à abstracção *form* da Mawl e mostrado como ela suporta o ciclo de vida do software através de uma abstracção apropriada, restrição e suporte de compilação. É discutida a forma como uma LDE tem apoiado a resolver vários problemas de engenharia de software que surgem com a criação de serviços web e telefónicos. Quando é aplicável, descreve-se como é que estes resultados se generalizam a outros domínios. Finalmente são consideradas as carências e pontos fracos da Mawl e das LDEs em geral.

A abstracção baseada em *forms* fornece solução para vários problemas:

- Boa formação de serviços web;
- Implementação, flexibilidade e independência da plataforma;
- Serviços de prototipagem;
- Teste e validação;
- Suporte de múltiplos dispositivos;
- Composição de serviços web;
- Análise de uso.
-

4.3.3. PADS

PADS (Fisher e Gruber, 2005) é uma linguagem declarativa de descrição de dados que permite a análise de dados para descrever tanto o esboço físico dos recursos de dados Ad Hoc, como propriedades semânticas destes dados. O compilador PADS gera bibliotecas e

ferramentas para manipulação de dados, incluindo rotinas para *parsing*, ferramentas estatísticas, programas de tradução para produzir formatos bem formados como XML ou aqueles requeridos para carregar bases de dados relacionais e ferramentas para correr *XQueries* sobre fontes de dados brutos PADS.

Vários conteúdos de dados utilizáveis são guardados e processados no formato Ad Hoc. As bases de dados tradicionais e os sistemas XML fornecem infra-estruturas para processamento de dados, no entanto, existem carências quando se lida com formatos Ad Hoc. Em primeiro lugar, os analistas que recebem dados Ad Hoc, não possuem possibilidade de os pedir num formato tratado, pois os dados são fornecidos tal como estão. Em segundo lugar, a documentação do formato pode não existir, ou pode estar desactualizada, e ainda, os dados contêm erros frequentemente, devido a vários factores. Finalmente os recursos de dados Ad Hoc podem ter um elevado volume.

O objectivo deste trabalho consiste então em desenvolver um *parser* para tratar este tipo de dados. Tende-se a utilizar a linguagem C ou Perl para esta tarefa, o que se torna numa tarefa tediosa, propensa a erros e complicada, devido à falta de documentação. Desta forma, o sistema PADS torna a vida facilitada para os analistas de dados, fornecendo uma linguagem de descrição de dados que permite descrever o esboço físico dos seus dados. A linguagem também permite descrever propriedades semânticas esperadas dos seus dados, de modo a que os desvios podem ser sinalizados como erros. O objectivo é permitir ao analista capturar nas descrições PADS tudo o que eles sabem sobre um conjunto de dados fornecido e disponibilizar aos analistas uma biblioteca de rotinas úteis em troca.

4.3.4. Uma LDE para modelos de dinâmica de paisagens

O estudo da dinâmica natural de paisagens envolve também o uso de modelos de simulação para explorar potenciais mudanças ao longo dos tempos e áreas extensas de território. No entanto, colmatar a lacuna entre os modelos conceptuais de paisagens dinâmicas e a sua simulação em computadores pode levar a alguns inconvenientes. Se esta implementação for feita usando uma linguagem de propósito geral, o modelo subjacente fica escondido nos detalhes do código, tornando difícil a comparação entre os modelos conceptuais e implementados e a modificação dos modelos.

As linguagens de domínio específico têm sido desenvolvidas em várias áreas para facilitar a construção de modelos a um nível perto do modelo conceptual, tornando a implementação do modelo mais acessível aos peritos. É neste contexto que surge a LDE descrita neste trabalho.

O objectivo deste trabalho (Fall e Fall, 2001) foi criar uma linguagem para modelação de paisagens dinâmicas que fornece aos ecologistas e urbanistas uma ferramenta adequada para resolver alguns dos problemas que surgem no desenvolvimento dos modelos. Esta linguagem estruturada e de alto nível separa a especificação do comportamento dos modelos, dos mecanismos da sua implementação, facilitando o trabalho dos modeladores de paisagens e permitindo-lhes que se foquem no modelo subjacente. Dado o carácter declarativo desta linguagem, ela permite uma representação clara do modelo conceptual ao nível semântico ao contrário de uma especificação por procedimentos ou *step-by-step*.

4.3.5. Uma LDE para *drivers* de dispositivos de videos

Neste trabalho é proposta uma *framework* para desenvolvimento de geradores de aplicações que engloba duas alternativas: a abordagem baseada em compiladores, e a abordagem baseada em interpretadores. Ela é também estruturada em dois níveis: o primeiro consiste na definição de uma máquina abstracta, cujas operações podem ser visualizadas como uma componente genérica que captura as operações importantes do domínio; e o segundo é a definição de uma micro linguagem, em termos de operadores de uma máquina abstracta, mas fornecendo uma interface de alto nível para a máquina abstracta.

Este artigo (Thibault *et* al., 1997) descreve uma aplicação realística da *framework* descrita para a geração automática de *drivers* para placas de vídeo. Este domínio forma uma família de programas, para as quais as LDEs são bem adaptadas.

É feito o desenho e a definição completa de uma LDE para adaptadores de vídeo, mostrado como uma avaliação parcial pode qualificar *drivers* eficientes, e assegurando-se que todos os motores gerados podem ser comprovados para finalizar e definir algumas análises que podem aumentar a sua fiabilidade.

As LDEs cumprem a promessa de fornecer um elevado retorno em termos de reutilização de software, análise automática de programas e Engenharia de Software.

A linguagem GAL foi a linguagem desenvolvida neste trabalho. Foram demonstrados os benefícios desta linguagem, mostrando como a GAL atinge o nível de abstracção da especificação de um *driver* e identificando algumas análises que podem ser desempenhadas na especificação GAL, pois é específica ao domínio.

Uma contribuição adicional deste trabalho é a validação da *framework* através da sua aplicação a esta família de programas para fornecer uma implementação da linguagem GAL.

Dado que a implementação é baseada em avaliação parcial, isso não só forneceria um interpretador completo para prototipagem de dispositivos *drivers*, mas também gera automaticamente dispositivos *drivers* com sucesso.

4.3.6. ATMOL

Este trabalho (Engelen, 2001) descreve o desenho e a implementação de uma LDE para formular e implementar modelos atmosféricos. Com o objectivo de garantir a facilidade de uso, uma notação concisa e a adopção de notações convencionais comuns, a linguagem foi desenvolvida em colaboração com meteorologistas do Instituto Meteorológico de Royal Netherlands.

A linguagem ATMOL foi traduzida e compilada em códigos numéricos eficientes com CTADEL. Esta ferramenta de síntese de código é capaz de gerar código de especificações de alto nível de modelos baseados em PDE (*Partial Differential Equations*). As vantagens da síntese de código são consideradas as seguintes:

- Aumento da Produtividade;
- Manutenção melhorada;
- Aumento da fiabilidade;
- Flexibilidade;

As construções de alto nível da ATMOL são declarativas e com efeito colateral livre, o que é requerido pelas aplicações de transformações para traduzir e optimizar as fases intermédias do modelo e do seu código. A sua expressividade permite a formulação de modelos de alto e baixo nível como construtores de linguagem para problemas de refinamentos e síntese de código. A ferramenta é restrita e requer a inserção dos objectos antes de serem usados.

A ATMOL é basicamente uma linguagem de transformação com que as operações semânticas dos operadores dos modelos são definidas. Todos os operadores PDE *built-in*, métodos de solução e algoritmos são escritos em ATMOL e fornecidos como construções pré-definidas. As construções da linguagem ATMOL estão englobadas em cinco diferentes níveis de abstracção (*Meta-level*, *Model declarations*, a *Coordinate-free scalar PDE problem*, *The numerial schemes* e *Program Code*), e permitem especificações de modelos com uma mistura de níveis de abstracção.

4.4. Sumário

Neste capítulo é apresentado o estado da arte através da introdução de alguns trabalhos na área sobre a qual esta dissertação vai incidir. Na secção 1 são apresentadas LDEs para abordagens de requisitos, como é o caso da LDE para i* e da LDE para KAOS. Na secção 2 são apresentadas LDEs para programação orientada a aspectos. Aqui é apresentada uma LDE com mais pormenor, e é feito um resumo de três outras LDEs. Por último são apresentados trabalhos que focam a construção de LDEs noutros domínios. Em todos estes trabalhos são evidenciadas as vantagens de utilizar este tipo de linguagem, notando-se um aumento significativo da sua utilização.

5. LDE VisualAORE

A LDE VisualAORE é uma linguagem visual que disponibiliza um editor para a criação de modelos visuais AORE.

O desenvolvimento da LDE VisualAORE foi realizado segundo a sequência de fases descrita na secção 5 do Capítulo 3. Deste modo, foi realizada uma análise de domínio que se encontra descrita na secção seguinte deste capítulo. Na secção 5.2 é descrita a fase de desenho da linguagem, que consistiu em criar o seu metamodelo, e a secção 5.3 aborda a fase da sua implementação.

5.1. Análise do domínio

A análise do domínio objectiva estabelecer a terminologia e os conceitos da linguagem, assim como efectuar uma análise das variacões, combinações e da similaridade. Neste trabalho, a análise de domínio foi elaborada através do método FODA, que consistiu na construção de um diagrama de *features* (Figura 5.1).

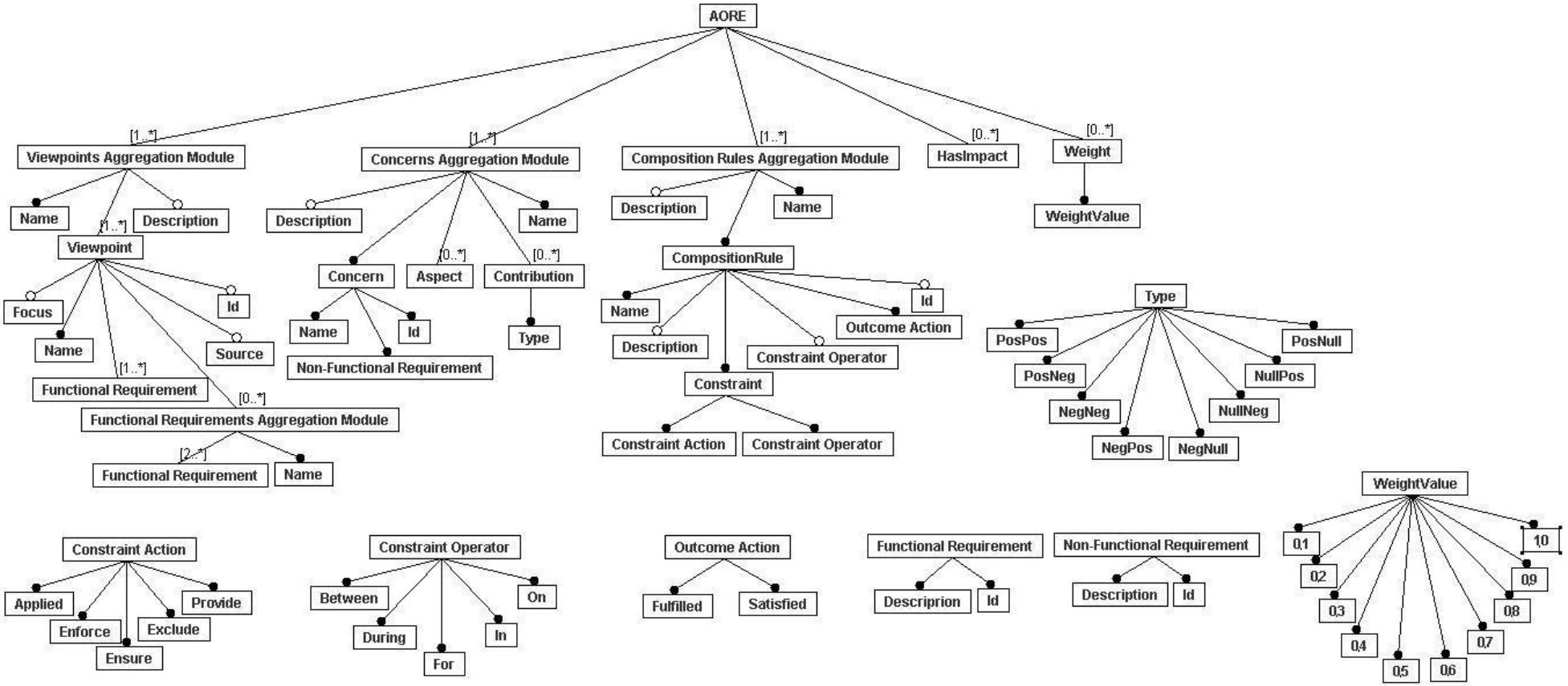

Figura 5.1. Diagrama de *features.*

No diagrama da Figura 5.1 observa-se que um modelo AORE é composto por uma ou várias *features Viewpoints Aggregation Module*, uma ou várias *features Concerns Aggregation Module*, uma ou mais *features Composition Rules Aggregation Module*, zero ou mais *features HasImpact* e zero ou mais *features Weight*.

A *feature Viewpoints Aggregation Module* representa um módulo de agregação de *viewpoints* e portanto, é composta por uma ou mais *features Viewpoint*. Possui a *feature Name* e opcionalmente a *feature Description*. A *feature Viewpoint* é composta pelas *features Focus, Name, Source,* e *Description* sendo que apenas a *feature Name* é obrigatória. É também composta por uma ou mais *features Functional Requirement* e zero ou mais *features Functional Requirements Aggregation Module*. A *feature Functional Requirement* é composta obrigatoriamente pelas *features Description* e *Id*. A *feature Functional Requirements Aggregation Module* possui obrigatoriamente a *feature Name* e no mínimo duas *features Functional Requirement*.

A *feature Concerns Aggegation Module* representa um módulo de agregação de *concerns* e é composta por uma *feature* obrigatória *Name*, uma opcional *Description*, uma ou mais *features Concern*, zero ou mais *features Aspect* e zero ou mais *features Contribution*. A *feature Concern* é composta pelas *features Name* e *Id*, sendo que *Name* é obrigatória. É também constituída por uma ou mais *features Non-Funcional Requirement*. Uma *feature Non-Functional Requirement* é composta obrigatoriamente pelas *features Description* e *Id*. A *feature Contribution* representa uma relação de contribuição entre dois *concerns* e é composta por uma *feature Type* que representa o tipo de contribuição. *Type* toma obrigatoriamente um e um só valor entre os valores +,+; +,-; -,-; -,+; -,*null*; *null*,-; +,*null* e *null*,+.

A *feature Composition Rules Aggregation Module* é composta por uma *feature* obrigatória *Name*, as *features* opcionais *Id* e *Description* e por uma ou várias *features Composition Rule*. Esta última é composta por uma *feature Constraint*, uma *feature Outcome Action* e opcionalmente uma *feature Constraint Operator*. A *feature Constraint* é composta por uma *feature Constraint Action* e uma *feature Constraint Operator*. A *feature Constraint Action* é de um e um só tipo de entre as *features Applied*, *Enforce*, *Ensure*, *Exclude* e *Provide*. A *feature Constraint Operator* é de um dos tipos entre o conjunto formado por *Between*, *During*, *In*, *On* e *Xor*. A *feature Outcome Action* é *Satisfied* ou *Fulfilled*, mas não ambas.

A *feature Weight* representa uma relação de atribuição de pesos entre um *concern* e um *viewpoint* e é composta por uma *feature WeightValue*. Esta última é a *feature* descritiva do

valor numérico do peso da relação e toma apenas um valor do conjunto de valores *0.1*, 0.*2*, *0.3*, *0.4*, *0.5*, *0.6*, *0.7*, *0.8*, *0.9* e *1.0*.

5.2. Metamodelo da linguagem VisualAORE

A análise de domínio detalhada na secção anterior foi introduzida como *input* na fase de desenho abordada nesta secção. Com base nas características identificadas no modelo de *features*, foi elaborado um metamodelo Ecore para a linguagem VisualAORE, que tem a função de relacionar estas características e adicionar restrições ao seu comportamento.

A linguagem VisualAORE foi criada a partir do metamodelo Ecore da Figura 5.2. A classe *Diagram* deste metamodelo é a classe AORE, e agrega as classes *ConcernsAggregationModule*, *CompositionRulesAggregationModule*, *ViewpointsAggregationModule*, *HasImpact* e *Weights*.

Uma lista dos conceitos do metamodelo que representam nós e o seu significado podem ser consultados na Tabela 5.1. A Tabela 5.2 contém os conceitos que representam ligações e uma breve explicação de cada um.

Com o objectivo de obter uma boa organização e visualização dos dados, um novo conceito foi introduzido na linguagem VisualAORE: o conceito de módulo de agregação. Este conceito surge com o intuito de organizar e agrupar dados semelhantes num módulo de informação, melhorando a organização dos modelos e facilitando a sua leitura. Um módulo de agregação é uma componente que agrupa informação distincta consoante se trata de um módulo de agregação de *concerns*, *viewpoints* ou regras de composição. A organização dos módulos de agregação é visível no excerto do metamodelo AORE da Figura 5.3. Nesta figura é visível que um modelo AORE deve conter, no mínimo, um módulo de cada tipo de módulo existente, sendo que dentro de cada um pode, eventualmente, existir um ou mais módulos do mesmo tipo.

A inserção deste novo conceito é importante pois permite o agrupamento de elementos do mesmo tipo dentro de uma estrutura, o que facilita a organização dos elementos e consequentemente origina modelos mais organizados. É um conceito que contribui também para o aumento da escalabilidade dos modelos devido ao facto de possuir um compartimento que pode ser minimizado, ocultando a informação do módulo.

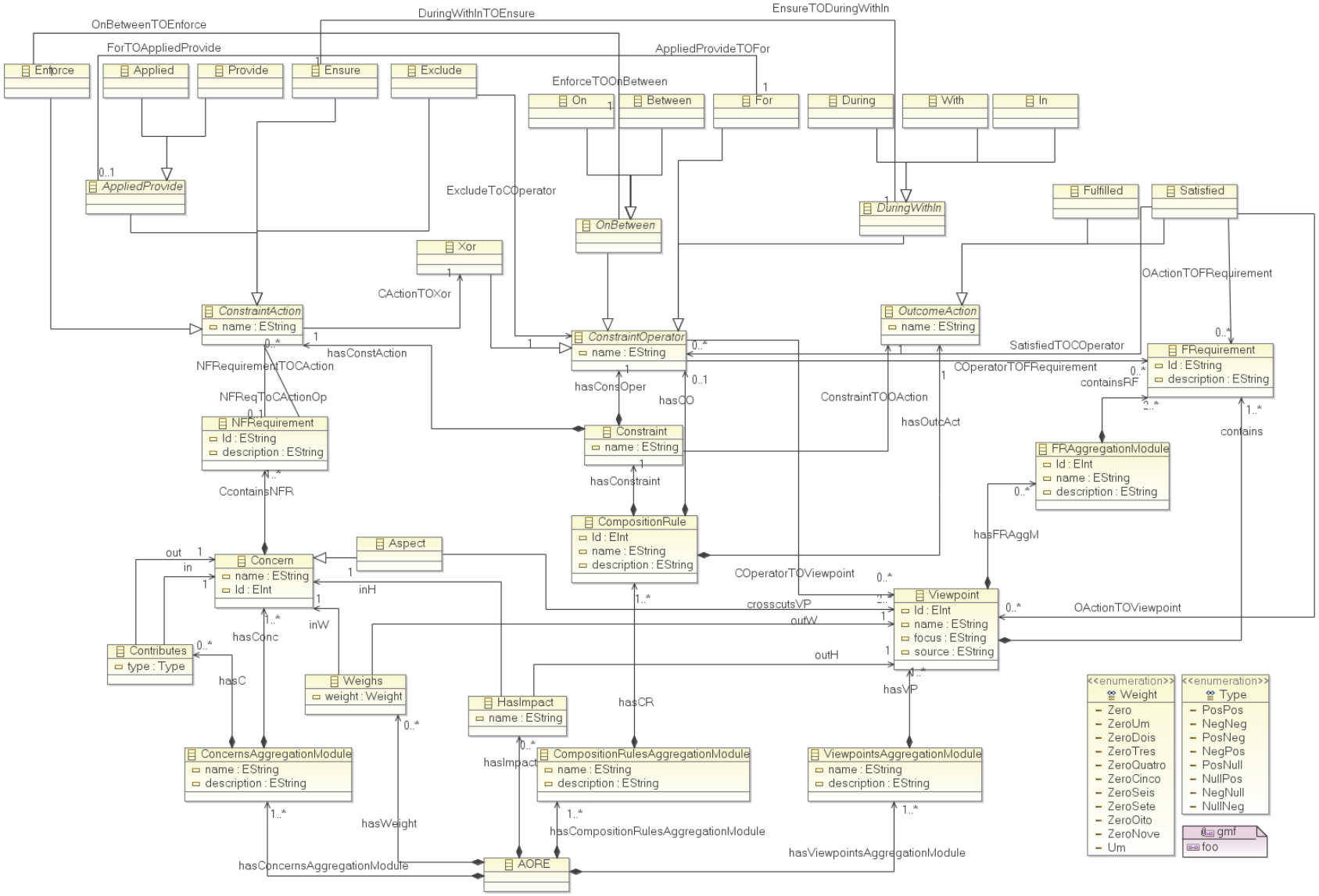

Figura 5.2. Metamodelo AORE.

Tabela 5.1. Classes do metamodelo que representam nós no editor VisualAORE e sua descrição.

Classes do metamodelo			Descrição
Nós	**ConcernsAggregationModule**		Representa um módulo para agregar um ou vários *concerns*. Possui os atributos *Name* e *Description*.
	CompositionrulesAggregationModule		Representa um módulo de agregação de uma ou mais de regras de composição. Possui os atributos *Name* e *Description*.
	ViewpointsAggregationModule		Representa um módulo de agregação de um ou vários *viewpoints*. Possui os atributos *Name* e *Description*.
	Concern		Representa um *concern*. Possui os atributos *Name* e *Id*.
	Aspect		Representa um aspecto. Possui os atributos *Name* e *Id*.
	Viewpoint		Representa um *viewpoint*. Possui os atributos *Name*, *Id*, *Focus* e *Source*.
	FRequirement		Representa um requisito funcional que compõe a especificação de um *viewpoint*. Possui os atributos *Id* e *Description*.
	NFRequirement		Representa um requisito não-funcional utilizado na especificação de um *concern* ou aspecto. Possui os atributos *Id* e *Description*.
	FRAModule		Representa um módulo de agregação de requisitos funcionais. Possui os atributos *Name*, *Id* e *Description*.
	CompositionRule		Representa uma regra de composição. Possui os atributos *Name*, Id e *Description*
	Contraint		Representa a restrição que existe dentro de uma regra de composição. Possui o atributo *Name*.
	Constraint Action	**Applied**	Utilizado para descrever as regras que se aplicam a um conjunto de requisitos de um *viewpoint* e que podem alterar o seu resultado. Todos os elementos que se englobam nesta secção herdam o atributo *Name*.
		Enforce	Utilizado para impor uma condição adicional sobre um conjunto de requisitos de um *viewpoint*
		Ensure	Usado para afirmar que uma condição que deve existir para um conjunto de requisitos de um *viewpoint* já existe.
		Exclude	Utilizado para excluir alguns *viewpoints* ou requisitos funcionais.
		Provide	Utilizado para especificar características adicionais a serem incorporadas num conjunto de requisitos de um *viewpoint*.
	Constraint Operator	**On**	Descreve o ponto no tempo depois de um conjunto de requisitos funcionais ter sido satisfeito. Todos os elementos que se englobam nesta secção herdam o atributo *Name*.
		Between	Descreve o intervalo de tempo compreendido entre a satisfação de dois requisitos funcionais. O intervalo começa quando o primeiro requisito é satisfeito, e termina quando o segundo está a começar a ser satisfeito.
		For	Descreve que características adicionais vão complementar os requisitos do *viewpoint*.
		During	Descreve o intervalo de tempo em que um conjunto de requisitos está a ser satisfeito.
		With	Descreve que uma condição vai ser suportada por dois conjuntos de requisitos, um em relação ao outro.
		In	Descreve que uma condição vai ser assegurada por um conjunto de requisitos que foram satisfeitos.
		Xor	Ou-Exclusivo (quando cada requisito é satisfeito, mas não ambos).

	Outcome Action	**Fullfiled**	Usado para afirmar que as restrições de um requisito aspectual foram aplicadas com êxito. Todos os elementos que se englobam nesta secção herdam o atributo *Name*.
		Satisfied	Usado para afirmar que um conjunto de requisitos de um *viewpoint* vão ser satisfeitas após serem aplicadas as restrições do requisito aspectual.

Tabela 5.2. Relações do metamodelo que representam *links* e a sua descrição.

Classes do metamodelo		**Descrição**
Ligações	**Contributes**	Representa uma ligação bi-direccional entre dois concerns indicando a contribuição que cada um efectua no outro. Possui o atributo *type* que é do tipo *Type*. Este tipo é um tipo enumerado que contém as possíveis contribuições entre dois *concerns*.
	HasImpact	Representa a existência de um impacto de um *concern* para com um *viewpoint*.
	Weights	Representa a ligação cuja origem é um *concern* e o destino é um *viewpoint*, indicando a existência de um peso no impacto exercido. Possui o atributo *weight* do tipo *Weight*. Este tipo é um tipo enumerado que contém todos os valores de contribuição possíveis.
	NFRequirementTOCAction	Ligação cuja origem é um requisito não-funcional de um *concern* e o destino é um elemento do tipo *Constraint Action*. Trata-se da primeira ligação de uma regra de composição.
	AppliedProvideToFor	Ligação cuja origem pode ser o elemento *Applied* ou o elemento *Provide* de *Constraint Action*, e o destino é o elemento *Constraint Operator For*.
	CActionToXor	Ligação cuja origem pode ser qualquer elemento de *Constraint Action*, e o destino é o elemento *Constraint Operator Xor*.
	EnforceToOnBetween	Ligação cuja origem é o elemento *Constraint Action Enforce*, e o destino pode ser o elemento *Constraint Operator On* ou *Between*.
	EnsureTODuringWithIn	Ligação cuja origem é o elemento *Constraint Action Ensure*, e o destino pode ser o elemento *Constraint Operator During*, *With* ou *In*.
	ExcludeTOCOperator	Ligação cuja origem é o elemento *Constraint Action Exclude*, e o destino pode ser qualquer elemento *Constraint Operator*.
	ConstraintTOOAction	Ligação cuja origem é o elemento *Constraint*, e o destino pode ser qualquer elemento *Outcome Action*.
	SatisfiedToCOperator	Ligação cuja origem é o elemento *Outcome Action Ensure*, e o destino pode ser qualquer *Constraint Operator*.
	COperatorTOFRequirement	Ligação cuja origem é um elemento *Constraint Operator*, e o destino é um requisito funcional.
	COperatorTOFRAModule	Ligação cuja origem é um elemento *Constraint Operator*, e o destino é um módulo de agregação de requisitos funcionais.
	COperatorTOViewpoint	Ligação cuja origem é um elemento *Constraint Operator*, e o destino é um *viewpoint*.
	OActionTOFRequirement	Ligação cuja origem é um elemento *Outcome Action*, e o destino é um requisito funcional.
	OActionTOFRAModule	Ligação cuja origem é um elemento *Outcome Action*, e o destino é um módulo de agregação de requisitos funcionais.
	OActionTOViewpoint	Ligação cuja origem é um elemento *Constraint Operator*, e o destino é um *viewpoint.*

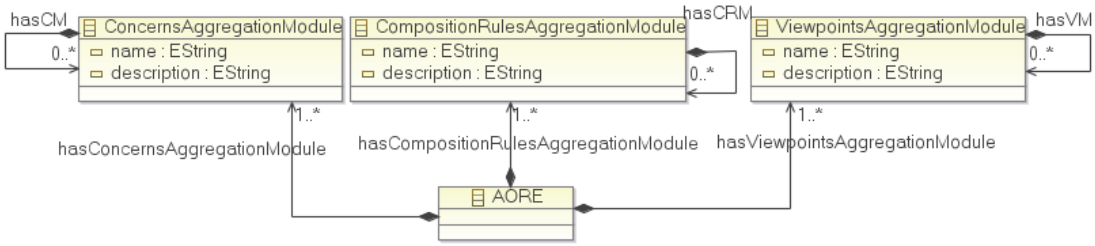

Figura 5.3. Excerto do metamodelo AORE que foca os módulos de agregação.

O conceito de módulo de agregação foi também utilizado para agrupar dois ou mais requisitos funcionais de um *viewpoint*, presentes numa regra de composição. O excerto do metamodelo que contém esta informação é ilustrado na Figura 5.4, onde se constata que um módulo de agregação de requisitos funcionais é constituído por dois ou mais requisitos funcionais. Nesta figura observa-se também que um *viewpoint* é composto por um ou vários requisitos funcionais e por zero ou mais módulos de agregação de requisitos funcionais.

O conceito de módulo de agregação aplicado neste contexto é importante pois reduz a quantidade de relacionamentos no modelo, na medida em que nas regras de composição que implicam mais do que um requisito do mesmo *viewpoint*, a ligação é efectuada ao módulo, e não a cada requisito de forma individual. Este facto contribui para a diminuição da complexidade dos modelos.

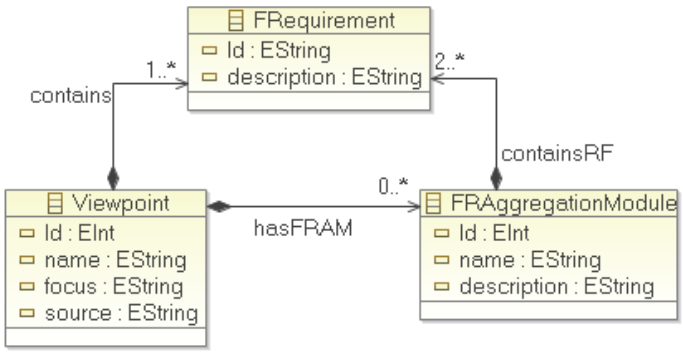

Figura 5.4. Excerto do metamodelo AORE que foca o módulo de agregação dos requisitos funcionais.

As regras de composição estão organizadas de acordo com a Figura 5.5. Existe um elemento chamado *Composition Rule* que contém um e um só element *Constraint*, apenas um elemento *Outcome Action* e opcionalmente um elemento *Constraint Operator*. O element *Constraint* possui um elemento *Constraint Action* e um elemento *Constraint Operator*. Para efectuar a conexão lógica entre estes elementos são utilizados relacionamentos entre as regras de composição, *concerns* e *viewpoints*. Deste modo, cada requisito ou sub-requisito de cada *concern* possui uma ligação a um elemento *Constraint Action*. Este elemento *Constraint Action* relaciona-se com um elemento *Constraint Operator* que se relaciona com um

viewpoint, módulo de agregação de requisitos funcionais ou requisito funcional em que interfere. Finalmente, o elemento *Constraint* está ligado ao *Outcome Action*. Caso o *Outcome Action* seja o elemento *Satisfied*, a *Constraint* é ligada ao mesmo *Satisfied*, e este pode ser posteriormente ligado a um *Constraint Operator*. No caso de se tratar apenas de um element *Satisfied*, este é ligado ao *viewpoint*, módulo de agregação de requisitos funcionais ou requisito funcional. No caso de existir um elemento *Constraint Operator*, a ligação é feita a este último. Se o *Outcome Action* for o elemento *Fulfilled*, não é efectuada qualquer ligação a outro elemento, pelo que a regra de composição termina neste elemento.

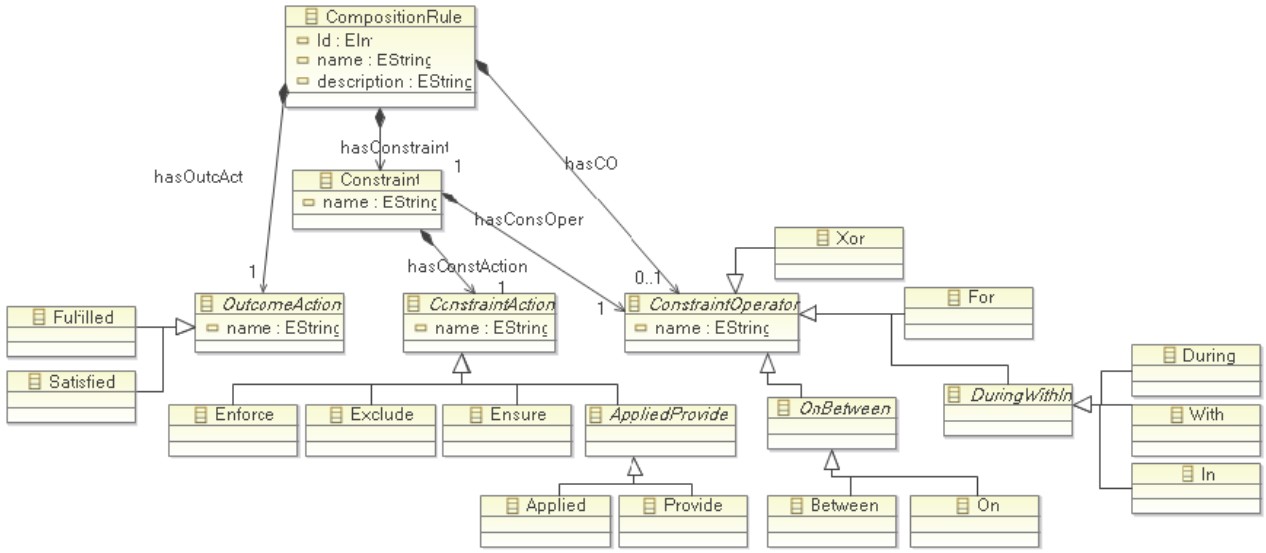

Figura 5.5. Excerto do metamodelo AORE para as regras de composição.

A Figura 5.6 ilustra as ligações entre os elementos das regras de composição. Com a finalidade de restringir quais os elementos que se podem ligar entre si, várias ligações do mesmo tipo foram efectuadas. Este processo poderia ter sido feito de outra forma, recorrendo à utilização de OCL. No entanto, esta opção foi tomada com o objectivo de auxiliar o utilizador a compreender quais as possíveis formas de combinar os elementos das regras de composição, dado que se trata de um processo restricto.

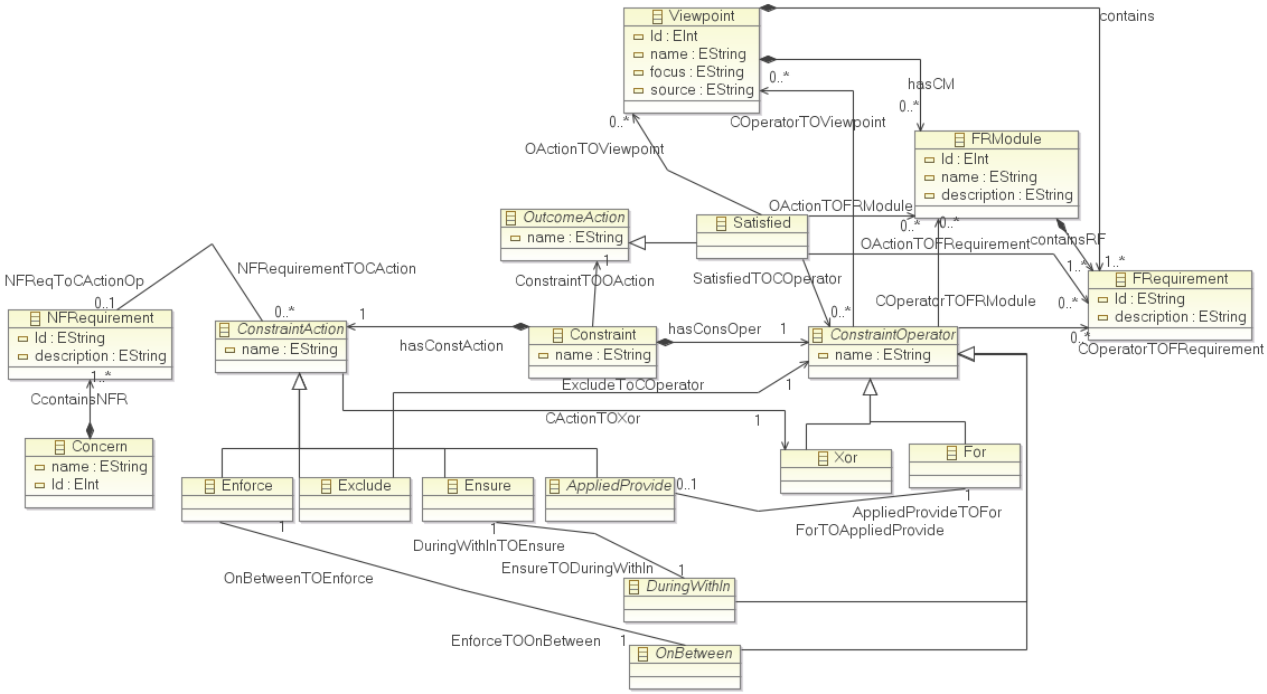

Figura 5.6. Ligações entre os elementos das regras de composição.

Na Figura 5.7 são ilustradas as possíveis ligações entre *concerns* e *viewpoints*. O *link HasImpact* representa a existência de um impacto de um *concern* num *viewpoint*. A relação *Contribution* representa a contribuição que dois *concerns* exercem entre si. A relação *Weights* surge com o objectivo de resolver situações de conflito detectadas com a relação *Contribution*. Deste modo, esta relação atribui um peso aos *viewpoints* afectados pelos *concerns* em conflito. Este excerto do metamodelo VisualAORE mostra ainda a relação de generalização entre um elemento *Concern* e um Aspecto. A diferença entre estes consiste no facto de que um Aspecto atravessa pelo menos dois *viewpoints*, ao passo que um *Concern* apenas influencia um *viewpoint*. Esta restrição é ilustrada através da relação *crosscutsVP* exposta na figura abaixo.

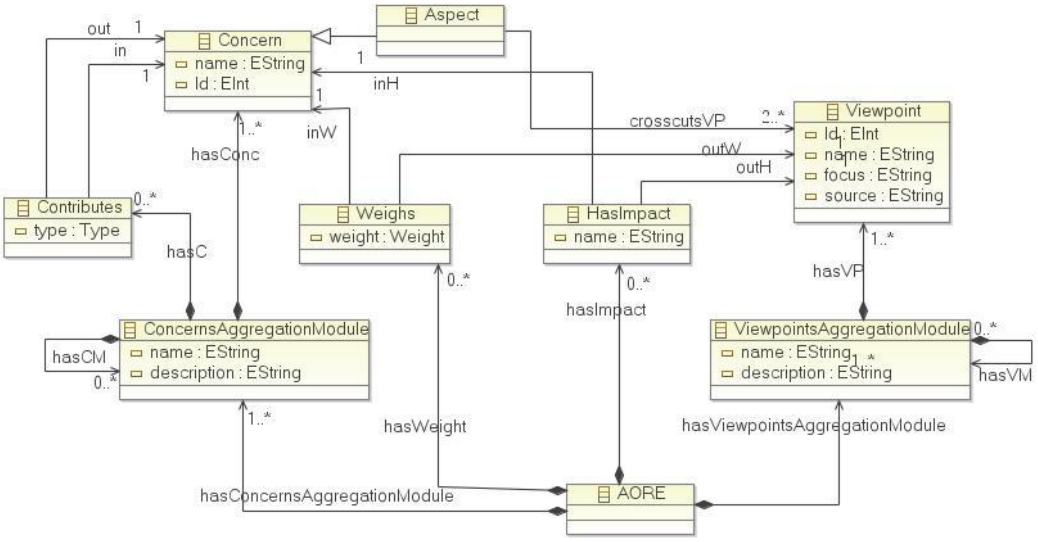

Figura 5.7. Relações entre *concerns* e *viewpoints.*

O metamodelo introduzido até aqui representa a sintaxe abstracta da LDE VisualAORE. Em seguida são introduzidos os símbolos que representam algumas das classes do metamodelo no editor final, assim como os ícones que estão associados a estas. O mecanismo de atribuir ícones e imagens aos elementos da linguagem dá origem à sua sintaxe concreta. A Tabela 5.3 contém os símbolos atribuídos às classes que extendem as classes abstractas *Contraint Action*, *Contraint Operator* e *Outcome Action* do metamodelo VisualAORE. Na Tabela 5.4 pode-se consultar o ícone atribuído a cada classe do metamodelo que possui uma representação visual no editor.

Tabela 5.3. Imagens externas dos conceitos das regras de composição.

Símbolo	Descrição
	Imagem das classes que estendem a classe abstracta *Constraint Action*.
	Imagem das classes que estendem a classe abstracta *Constraint Operator*.
	Imagem das classes que estendem a classe abstracta *Outcome Action*.

Tabela 5.4. Ícones da linguagem VisualAORE.

Ícone	Descrição
CAggM	Ícone que representa a classe *ConcernsAggregationModule*.
CRAggM	Ícone que representa a classe *CompositionRulesAggregationModule*.
VAggM	Ícone que representa a classe *ViewpointsAggregationModule*.
A	Ícone que representa a classe *Aspect*
C	Ícone que representa a classe *Concern*
NFR	Ícone que representa a classe *NFRequirement*.
CR	Ícone que representa a classe *CompositionRule*.
C	Ícone que representa a classe *Constraint*.
A	Ícone que representa a classe *Applied*.
Enf	Ícone que representa as classes *Enforce*.
Ens	Ícone que representa as classes *Ensure*.
Exc	Ícone que representa as classes *Exclude*.

P	Ícone que representa a classe *Provide*.
B	Ícone que representa a classe *Between*.
D	Ícone que representa a classe *During*.
F	Ícone que representa a classe *For*.
I	Ícone que representa a classe *In*.
O	Ícone que representa a classe *On*.
W	Ícone que representa a classe *With*.
X	Ícone que representa a classe *Xor*.
F	Ícone que representa a classe *Fullfiled*.
S	Ícone que representa a classe *Satisfied*.
V	Ícone que representa a classe *Viewpoint*.
FR	Ícone que representa a classe *FRequirement*.
FRAggM	Ícone que representa a classe *FRAggModule*.
	Ícone que representa a relação *Contribution.*
	Ícone que representa a relação *Weight.*
	Ícone que representa a relação *HasImpact.*
	Ícone que representa a *NFRequirementTOCAction*.
	Ícone que representa a relação *NFRAggModuleTOCAction*.

	Ícone que representa as relações que existem dentro de uma regra de composição. Estas são: *ConstraintTOOAction*, *EnforceToOnOrBetween*, *EnsureTODuringORWithORIn*, *ExcludeToCOperator*, *SatisfiedTOCOperator*, *SatisfiedTOCOperator*, *SatisfiedTOCOperator*.
	Ícone que representa as relações *OActionTOFRequirement*, *OActionTOFRAggModule* e *OActionTOViewpoint*.
	Ícone que representa as relações *COperatorTOFRequirement*, *OperatorTOFRAggModule*, *COperatorTOViewpoint*

5.3. Implementação

O ambiente de desenvolvimento utilizado para desenvolver a LDE VisualAORE foi o Eclipse Galileo[5], a sua plataforma EMF/GMF, e os *plug-ins* Emfatic e EuGENia. Esta escolha deve-se ao facto do Eclipse possuir as características desejadas para a elaboração de uma Linguagem de Domínio Específico, possuir também uma elevada quantidade de documentação disponível para ajuda, e ser uma ferramenta *open-source*. A introdução dos *plug-ins* Emfatic e EuGENia foi também um factor determinante, pois estes facilitam o processo de criação da LDE, permitindo especificar as suas propriedades ao nível do metamodelo, o que anteriormente só era possível em fases finais do seu desenvolvimento.

O processo de desenvolvimento da LDE no ambiente de desenvolvimento Eclipse foi semelhante ao apresentado na secção 3 do Capítulo 3, apresentando no entanto ligeiras alterações, devido à utilização dos *plug-ins* Emfatic e EuGENia. O procedimento seguinte ilustra a sequência de passos efectuada.

1. Criar o metamodelo Ecore ou a especificação do modelo ecore Emf. A partir de qualquer um destes modelos é possível obter o outro;
2. Com o auxílio do Dashboard, gerar o modelo Genmodel através do metamodelo Ecore. Deve fazer-se *Generate All* no modelo obtido para que seja gerado o código fonte correspondente aos elementos do metamodelo;
3. Através do ficheiro Ecore obter os modelos Gmftool, Gmfgraph e Gmfmap;
4. Obter o modelo Gmfgen a partir do modelo Gmfmap;
5. Gerar o diagrama recorrendo ao modelo Gmfgen.

[5] Eclipse Galileo, http://www.eclipse.org/downloads/packages/release/galileo/sr2, 2009.

O metamodelo da linguagem VisualAORE deu origem ao editor base e a três sub-editores. O editor base contém anotações em todas as suas classes concretas, tal como é visível no Anexo B. Os sub-editores foram feitos a partir de uma cópia do modelo ecore Emf do editor base, e cada um deles possui anotações apenas nas classes que constam do respectivo sub-editor. Assim, no caso do sub-editor do elemento *ConcernsAggregationModule*, a classe *ConcernsAggregationModule* possui a anotação *gmf.diagram*, indicando que esta é a classe raiz do metamodelo. As classes *Concern*, *Aspect* e *NFRequirement* são anotadas como *gmf.node* e a classe *Contribution* como *gmf.link*. O sub-editor do elemento *ViewpointsAggregationModule*, possui a classe *ViewpointsAggregationModule* como classe raiz do metamodelo, e as classes *Viewpoint*, *FRequirement* e *FRAggModule* são anotadas como *gmf.node*. O sub-editor do elemento *CompositionRulesAggregationModule* possui esta classe como raiz do metamodelo e possui as classe *CompositionRule*, *Constraint* e as classes que extendem as classes abstractas *ConstraintAction*, *ConstraintOperator* e *OutcomeAction* como *gmf.nodes*. Todas as classes que representam relações entre os elementos das regras de composição são anotadas com *gmf.link*. Os metamodelos ecore Emf dos sub-editores podem ser visualizados nos anexos C, D e E. O anexo C contém o metamodelo do sub-editor do elemento módulo de agregação de *concerns*, o anexo D contém o metamodelo do sub-editor do elemento módulo de agregação de regras de composição e finalmente o anexo E descreve o metamodelo do sub-editor do módulo de agregação de *viewpoints*.

O processo de criar um sub-editor é semelhante ao de criar o editor base. Nesta dissertação foi criada uma pasta para cada editor dentro do projecto da LDE. Realizou-se o processo descrito anteriormente para cada sub-editor em separado, até à fase da geração do modelo Gmfgen. Quando todos os modelos Gmfgen dos sub-editores se encontravam gerados, foi realizado o mesmo processo para o editor base, e foi efectuada a ligação entre o editor base e os sub-editores. Para efectuar a ligação entre os editores foi necessário carregar os ficheiros Gmfmap dos sub-editores no editor base, através da opção *load* do modelo Gmfmap do editor base. Feito isto, foi necessário editar as propriedades dos elementos que possuem um sub-editor, no modelo Gmfgen do editor base. Para tal, acedeu-se ao modelo Gmfgen do editor base, abriu-se o separador *Gen Editor Generator Aore.diagram*, em seguida o separador *Gen Diagram AoreEditPart* e seleccionou-se o nó que possui o sub-editor. A título de exemplo, no caso do nó *ConcernsAggregationModule*, seleccionou-se *Gen Top Level Node ConcernsModuleEditPart* e abriu-se o separador *Open diagram Behaviour ConcernsModuleDiagramEditPolicy*. São visíveis as propriedades deste separador na Figura

5.8. Nestas propriedades foi necessário editar os campos *Diagram Kind*, *Editor Id* e *Edit Policy Class Name.*

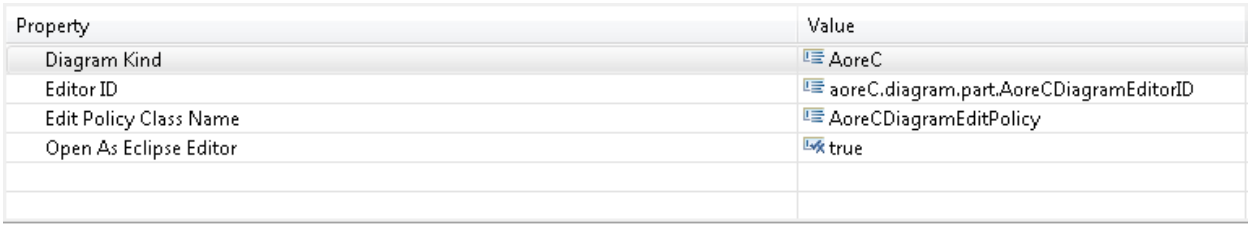

Figura 5.8. Propriedades editadas no modelo Gmfgen.

Este processo foi realizado para cada sub-editor. Finalmente, feitas as ligações entre os editores, foi necessário sincronizar o Gmfgen do editor base, e em seguida efectuou-se a geração dos diagramas de todos os editores.

Com vista restringir o comportamento do elemento *Contribution*, foi adicionada uma regra OCL no modelo Gmgmap do editor base e do sub-editor do módulo de agregação de *concerns*. Esta é uma relação entre elementos do mesmo tipo, e como tal foi necessário proibir que o elemento de origem fosse o mesmo que o elemento de destino. A regra OCL utilizada é visível na Figura 5.9.

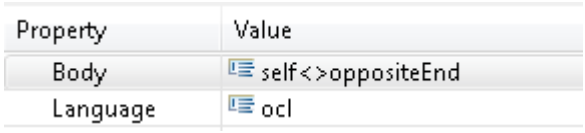

Figura 5.9. Regra OCL para o elemento *Contribution*.

5.4. Apresentação da ferramenta

A apresentação da ferramenta efectuada neste sub capítulo, recorre ao caso de estudo Via Verde. Este caso de estudo é real e o seu propósito é implementar o sistema Via Verde ao nível de auto-estradas e parques de estacionamento.

Neste trabaho, é considerada a vertente do caso de estudo direccionada ao funcionamento de auto-estradas.

Num sistema Via Verde, os utilizadores que usufruem do serviço são cobrados automaticamente nas portagens. Para tal, o sistema é implementado em faixas especiais onde o utilizador pode passar sem ter que parar. Para que este processo seja possível, é necessário que o utilizador possua um dispositivodenomidado de Identificador na sua viatura. Este Identificador possui os dados do seu proprietário, assim como o número da conta bancária que lhe está associada e alguns detalhes do veículo. Na fase inicial, o Identificador é fornecido ao proprietário, e este necessita de o activar através de uma estação multibanco, para que seja feita a ligação entre o sistema Via Verde e a conta bancária do proprietário.

O Identificador é geralmente localizado no vidro da frente do veículo, onde pode ser facilmente lido pelos sensores. Cada passagem feita pelo identificador nas faixas da Via Verde é lida pelos sensores, e é enviada informação para o sistema. Aquando das passagens do identificador pelos corredores Via Verde, vários cenários podem acontecer. Se o veículo for autorizado, uma luz verde é mostrada, e o valor a ser debitado é exibido. Caso o veículo não seja autorizado, é mostrada uma luz amarela, e uma fotografia é tirada à matrícula do veículo, para que este seja penalizado. No sistema Via Verde exitem 3 tipos de cancelas: cancela única, cancela de entrada e cancela de saída. A cancela única é utilizada por exemplo nos parques de estacionamento, onde o mesmo tipo de veículo paga uma quantia fixa. A cancela de entrada é utilizada aquando da entrada numa auto-estrada e a cancela de saída é utilizada aquando da sua saída. De notar que o montante pago numa auto-estrada depende do tipo de veículo e da distância percorrida.

É com base nesta informação que foi elaborado um pequeno excerto do caso de estudo Via Verde na ferramenta VisualAORE.

A LDE VisualAORE é composta por um editor base e três sub-editores, como já foi referido. Os sub-editores estão presentes nos elementos *ConcernsAggregationModule*, *CompositionRulesAggregationModule* e *ViewpointsAggregationModule*. Um sub-editor permite editar e/ou visualizar o conteúdo de um módulo num editor em separado, o que

simplifica a edição/leitura dos seus elementos. Cada editor contém no seu menu os conceitos que podem ser editados no módulo em questão. Para aceder ao sub-editor de cada elemento é necessário efectuar um duplo-clique com o botão esquerdo do rato no rebordo do elemento. As figuras abaixo (Figuras 5.10, 5.11, 5.12 e 5.13) ilustram os menus dos vários editores.

Na Figura 5.10 pode observar-se o menu do editor base. Este encontra-se dividido em várias secções:

- *Modules* – secção onde se pode aceder aos vários módulos de agregação existentes;
- *Concern* – secção onde estão colocados todos os conceitos que devem constar de um módulo de agregação de *concerns*;
- *Composition Rule* – secção que contém todos os conceitos necessários à elaboração das regras de composição e que devem constar de um módulo de agregação de regras de composição;
- *Viewpoint* – secção onde estão colocados todos os conceitos que devem constar de um módulo de agregação de *viewpoints*;
- *Connections* – secção que contém todas as relações possíveis de efectuar num modelo AORE. Esta secção encontra-se dividida segundo as situações em que são utilizadas e por tipo de relação.

A Figura 5.11 ilustra o menu do sub-editor do módulo de agregação de *concerns*. Neste menu estão contidos os nós *Aspect*, *Concern*, *Non-Functional Requirement*, e a relação *Contributes*. Esta última é utilizada para efectuar a ligação de contribuição entre *concerns* e/ou aspectos.

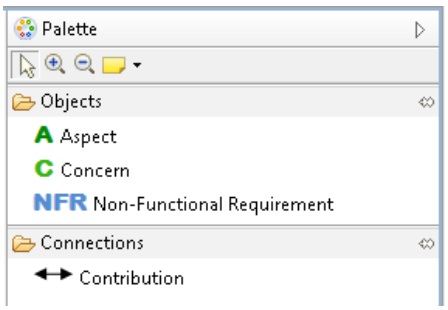

Figura 5.10. Menu do sub-editor do elemento *ConcernsAggregationModule*.

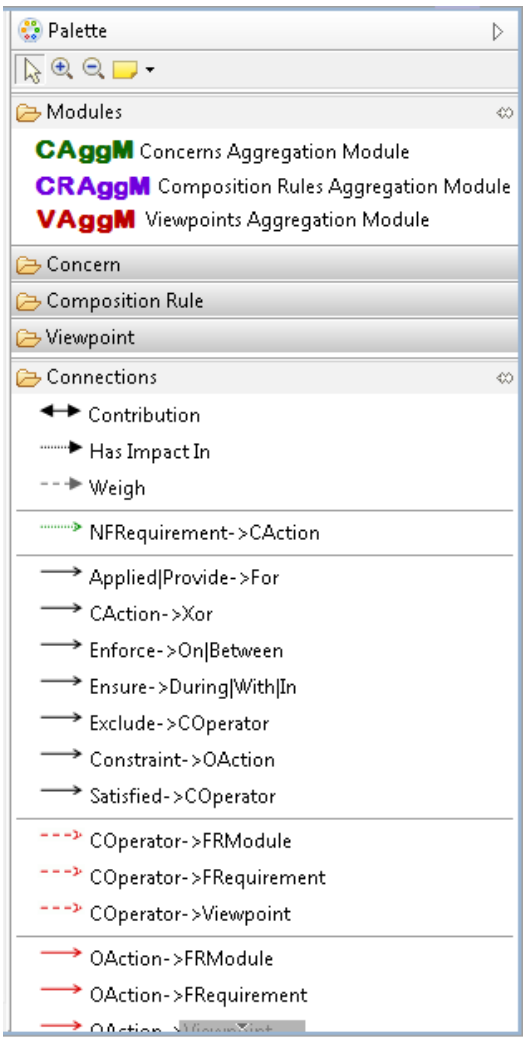

Figura 5.11. Menu do editor base.

Na Figura 5.12 é ilustrado o menu do sub-editor do módulo de agregação de regras de composição. Este menu encontra-se organizado de acordo com a ordem pela qual os elementos devem ser inseridos no editor. Assim, surgem inicialmente os conceitos *Composition Rule* e *Constraint* na secção de elementos genéricos. Em seguida, surgem no menu, as secções *Contraint Action*, *Constraint Operator* e *Outcome Action*. Dentro de cada uma destas secções, o utilizador deve escolher o elemento que pretende utilizar. Finalmente, surgem as relações que são feitas entre os nós. Para cada tipo de associação de elementos da restrição, ou seja, para cada combinação possível de *Constraint Action* e *Constraint Operator* existe uma relação distinta.

Na Figura 5.13 pode observar-se o menu do sub-editor do módulo de agregação de *viewpoints*. Este menu contém os conceitos *Viewpoint*, *Functional Requirement* e *Functional Requirement Aggregation Module*.

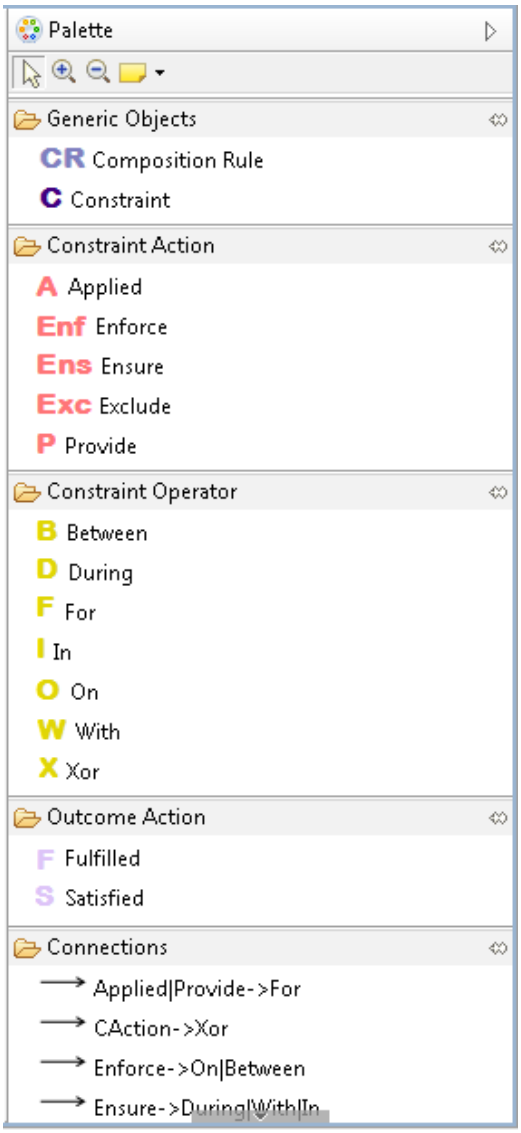

Figura 5.12. Menu do sub-editor do elemento *CompositionRulesAggregationModule*.

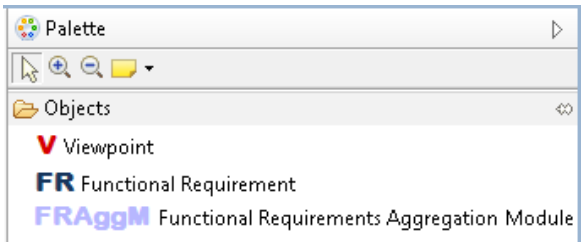

Figura 5.13. Menu do sub-editor do elemento *ViewpointsAggregationModule*.

Com o objectivo de proceder a uma explicação da LDE VisualAORE recorreu-se ao caso de estudo Via Verde apresentado no Capítulo 2 desta dissertação.

A ferramenta VisualAORE é composta por nós e *links.* Os *links* são representados através de setas, e a maioria dos nós são representados por rectângulos. No entanto, existem nós que são representados por imagens adicionadas de forma externa, através de um *plug-in* criado no decorrer deste trabalho, denominado *AOREFigures*.

A Figura 5.14 ilustra o conteúdo de um módulo de agregação de *concerns*. Neste módulo pode visualizar-se o *concern Correctness* e a sua especificação através de requisitos não funcionais. O módulo contém também os aspectos R*esponse Time* e C*ompatibility* e suas respectivas especificações. O aspecto *Response Time* exerce uma contribuição negativa sobre o *concern* C*orrectness* que é representada pela relação *Contribution: -, null*. A relação *Contribution* é uma relação bidireccional e como tal, o primeiro parâmetro corresponde à contribuição do elemento que se encontra à esquerda em relação àquele que está à direita, e a segunda contribuição é do elemento que se encontra mais à direita para com o que se encontra mais à esquerda. Caso os *concerns* se encontrem alinhados verticalmente, o primeiro parâmetro é da contribuição do elemento que se encontra acima para com o elemento que se encontra abaixo, e a segunda contribuição é do elemento que se encontra abaixo para com aquele que se encontra acima.

No caso da figura 5.14, não existe uma relação de contribuição inversa, pelo que o segundo parâmetro da relação *Contribution* é *null*.

Como se pode observar, um módulo de agregação de *concerns* é representado por um rectângulo verde com as extremidades num tom verde mais carregado. Tanto os *concerns* como os aspectos são representados por um rectângulo cinzento, no entanto possuem as extremidades com diferentes tons de verde. Os requisitos não funcionais são representados por um rectângulo cinzento cuja extremidade é cinzenta. Estas características surgem com o objectivo de facilitar a aprendizagem da linguagem.

A Figura 5.15 ilustra um módulo de agregação de *viewpoints*. Este módulo é denominado de *Toll Gate* e contém as especificações dos *viewpoints* de todas as cancelas existentes no sistema Via Verde. As especificações dos *viewpoints* são efectuadas através de requisitos funcionais. Como se pode observar na Figura 5.15, um módulo de agregação de *viewpoints* é representado por um rectângulo vermelho com as extremidades num tom carregado. Os *viewpoints* são representados por um rectângulo cinzento, cuja extremidade apresenta um tom

de vermelho mais leve. Cada *viewpoint* possui dois compartimentos: o primeiro compartimento suporta a adição dos requisitos funcionais; enquanto o segundo suporta a adição de módulos de agregação de requisitos funcionais. Os módulos de agregação de requisitos funcionais são representados por um rectângulo cinzento cujas extremidades são azuis. A separação entre os compartimentos do *viewpoint* é realizada através da linha vermelha que se encontra após a sua especificação. Os requisitos funcionais dos v*iewpoints* são também representados por um rectângulo cinzento cuja extremidade é cinzenta. Um módulo de agregação de regras de composição pode ser visualizado na Figura 5.16.

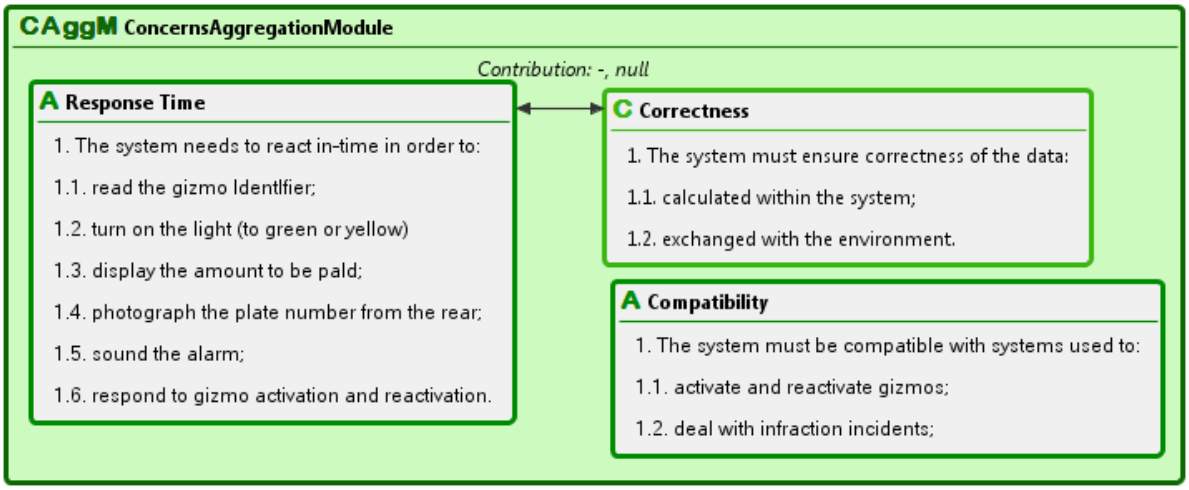

Figura 5.14. Módulo de agregação de *concerns* do caso de estudo Via Verde.

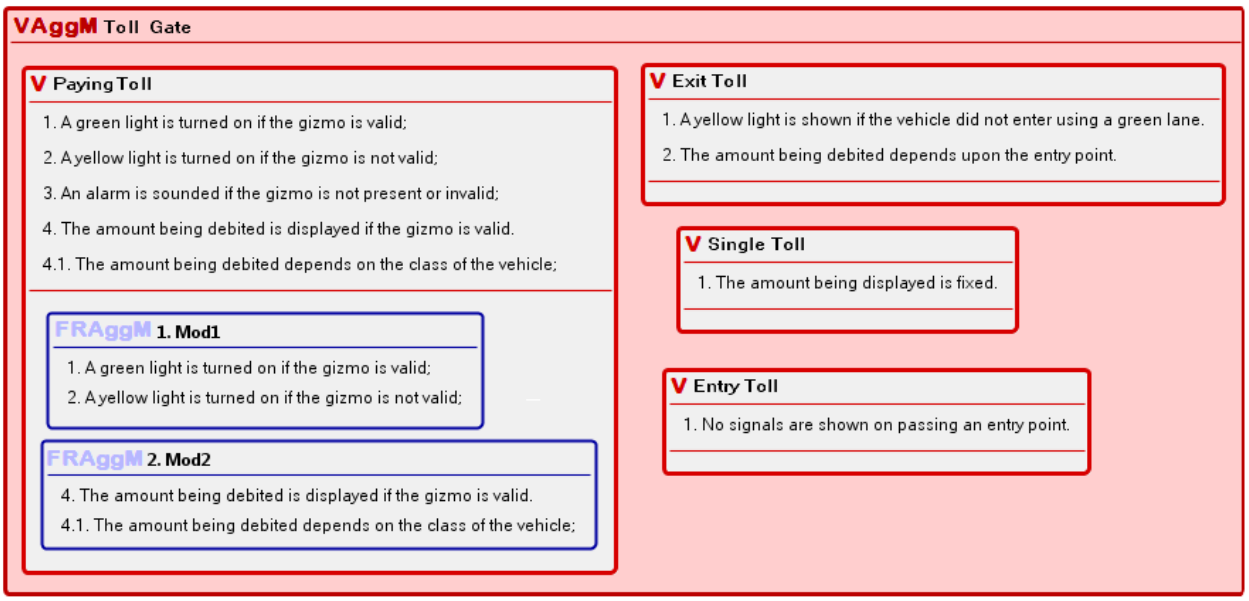

Figura 5.15. Módulo de agregação de *viewpoints* do caso de estudo Via Verde.

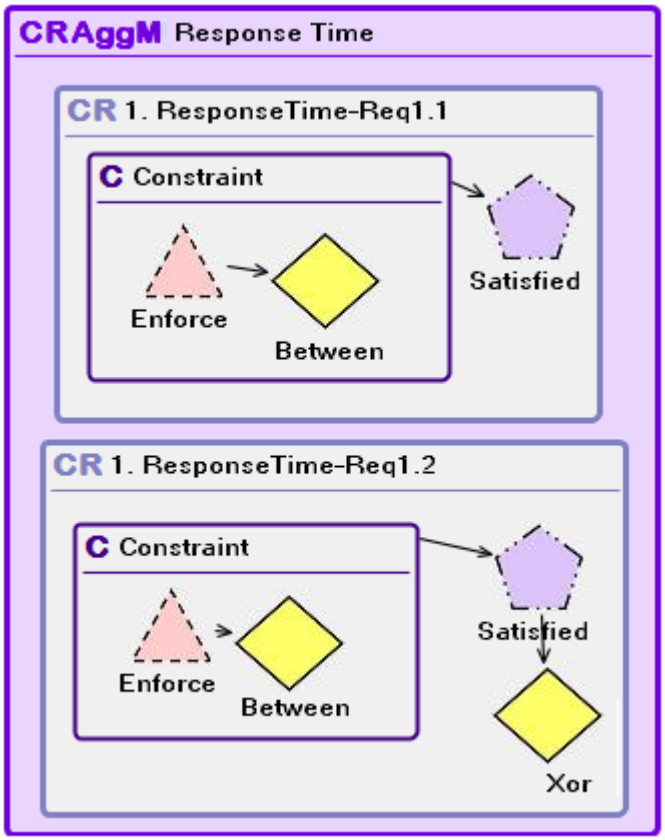

Figura 5.16. Módulo de agregação de regras de composição do caso de estudo Via Verde, sem relações com os *viewpoints* nem com os *concerns*.

Os módulos de agregação de regras de composição, as regras de composição e as restrições são representados através de rectângulos. Cada rectângulo possui uma diferente tonalidade da cor roxa para que sejam facilmente distinguidos. Os restantes elementos que constituem as regras de composição são representados por de imagens externas conforme foi mostrado anteriormente.

O módulo representado acima denomina-se *Response Time* pois possui as regras de composição dos requisitos não funcionais deste aspecto. Para cada requisito não funcional do aspecto é necessário efectuar uma regra de composição, pois cada requisito não funcional restringe o comportamento de um requisito funcional de um *viewpoint*, vários requisitos funcionais do mesmo *viewpoint* ou de *viewpoints* diferentes, ou mesmo de um ou mais *viewpoints* na sua totalidade.

No módulo da Figura 5.16 estão representadas apenas as regras de composição os requisitos 1.1 e 1.2. do aspecto *Response Time*.

A regra de composição do requisito 1.1. do aspecto *Response Time* e as suas ligações com os *viewpoints* e o *concern* que nela interferem podem ser observadas na Figura 5.17. Esta regra é denominada *ResponseTime-Req1.1*, e é composta pela *Constraint Action Enforce* e pelo

Constraint Operator Between. A acção de restrição *Enforce* indica que vai existir uma condição adicional sobre um conjunto de requisitos funcionais de um *viewpoint*. O operador de restrição *Between* indica que o requisito aspectual deve restringir o intervalo de tempo que se inicia com a satisfação do primeiro requisito funcional e termina no instante em que o segundo requisito funcional está a iniciar a sua satisfação. A acção que resulta desta restrição é *Satisfied* o que indica que um conjunto de requisitos de um *viewpoint* vão ser satisfeitos após serem aplicadas as restrições do requisito aspectual. Assim, o requisito 1.1 do aspecto *Response Time* deve ser cumprido no intervalo de tempo entre os requisitos do módulo de agregação de requisitos funcionais *Mod1* do *viewpoint Vehicle*. O resultado desta composição é o cumprimento dos requisitos do *viewpoint Gizmo*.

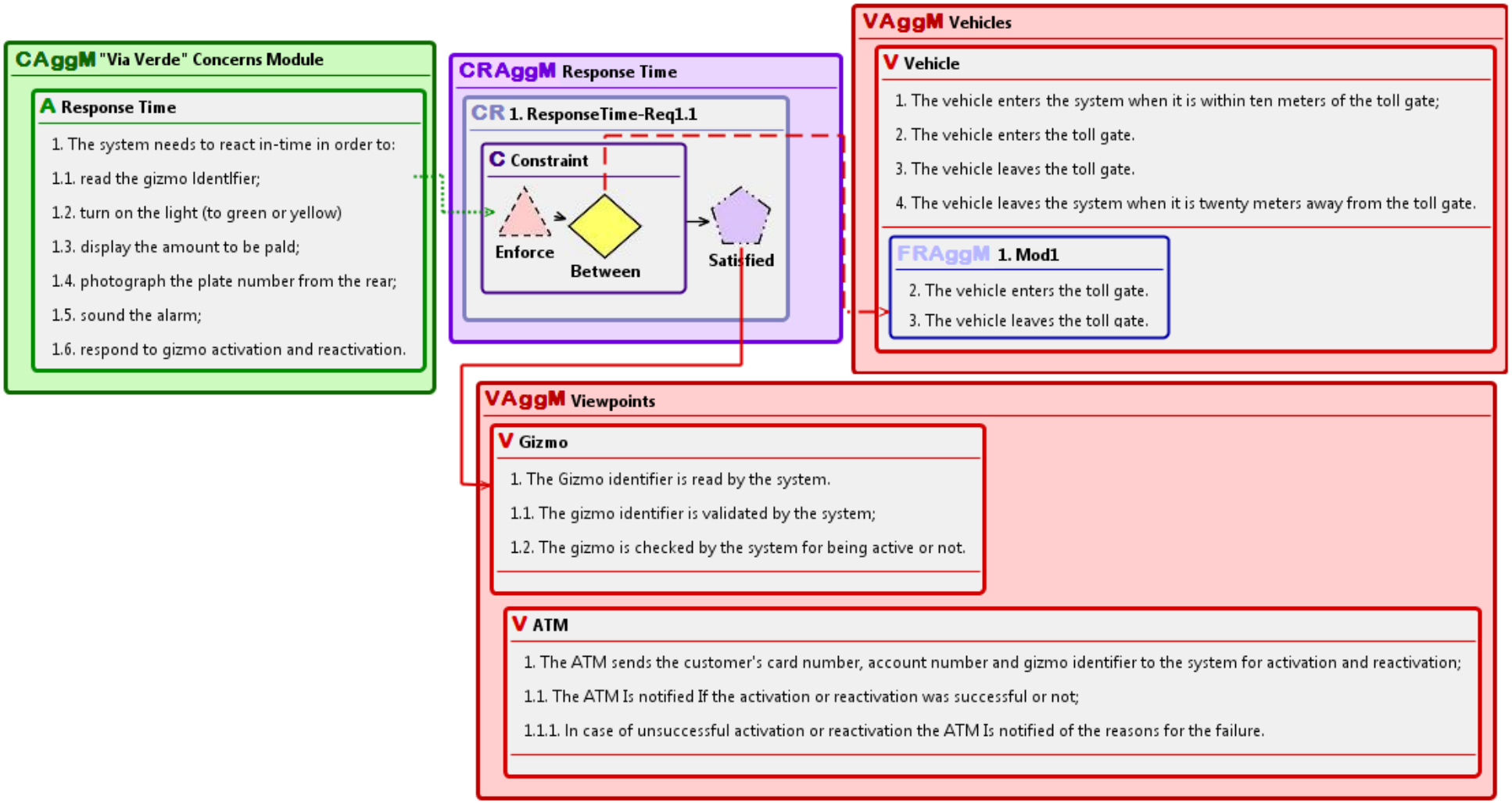

Figura 5.17. Regra de composição do requisito 1.1 do *concern Response Time*.

A regra de composição do requisito 1.2. do aspecto *Response Time* pode ser observada na Figura 5.18. Esta é denominada *ResponseTime-Req1.2*, e é composta pela acção de restrição *Enforce*, pelo operador de restrição *Between* e pela acção resultante *Satisfied* que se encontra associada ao operador de restrição *Xor*. Esta regra de composição é idêntica à regra anterior, no entanto difere no resultado final. Nesta última, a acção resultante *Satisfied* está associada a um operador de restrição *Xor*. Esta associação indica que só um dos requisitos funcionais associado ao operador *Xor* é que é satisfeito a cada altura, e não ambos. De forma resumida, o requisito 1.2 do aspecto *Response Time* deve ser cumprido no intervalo de tempo entre os requisitos do *viewpoint Gizmo*. O resultado desta composição é o cumprimento de apenas um dos requisitos do módulo *Mod1* do *viewpoint PayingToll*.

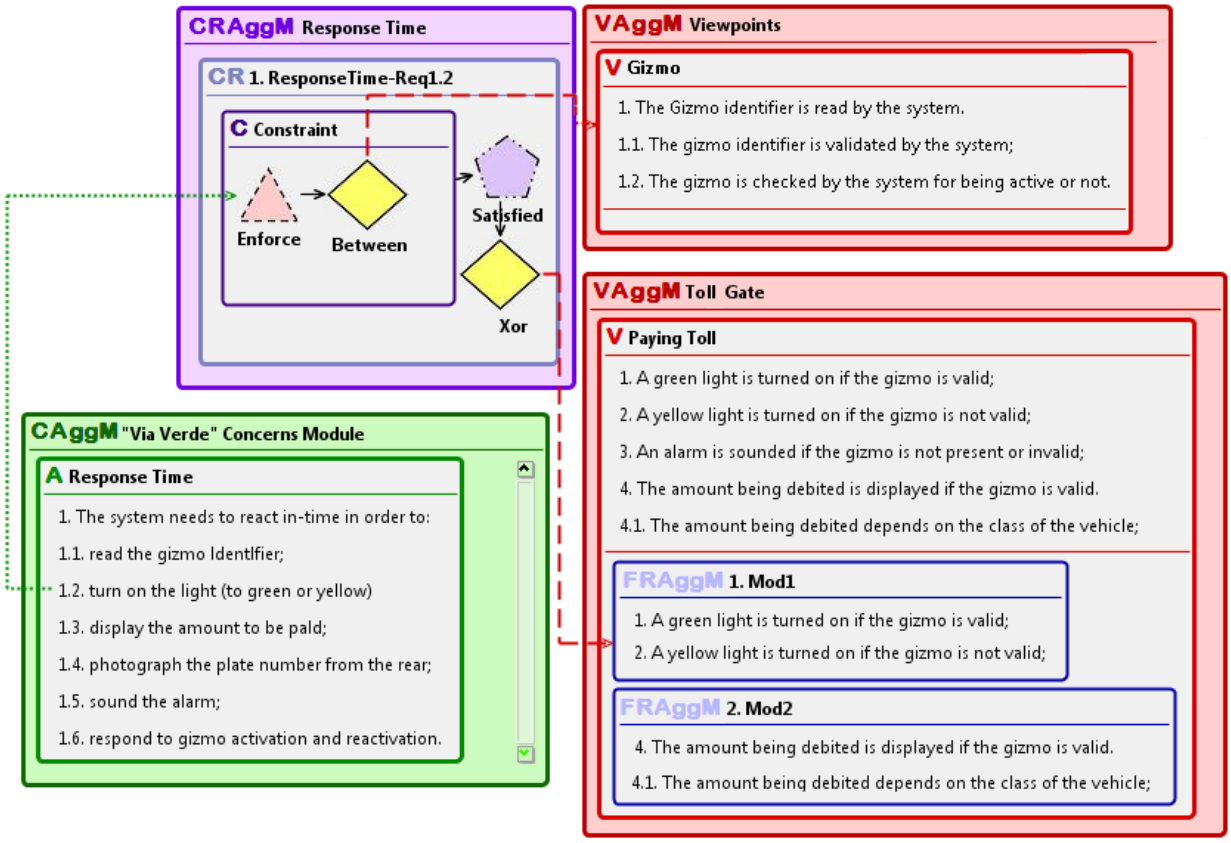

Figura 5.18. Regra de composição do requisito 1.2 do *concern* Tempo de Resposta.

O Anexo F ilustra o modelo AORE do caso de estudo Via Verde. Neste modelo é visível um módulo de agregação de *concerns*, três módulos de agregação de *viewpoints* e dois módulos de agregação de regras de composição.

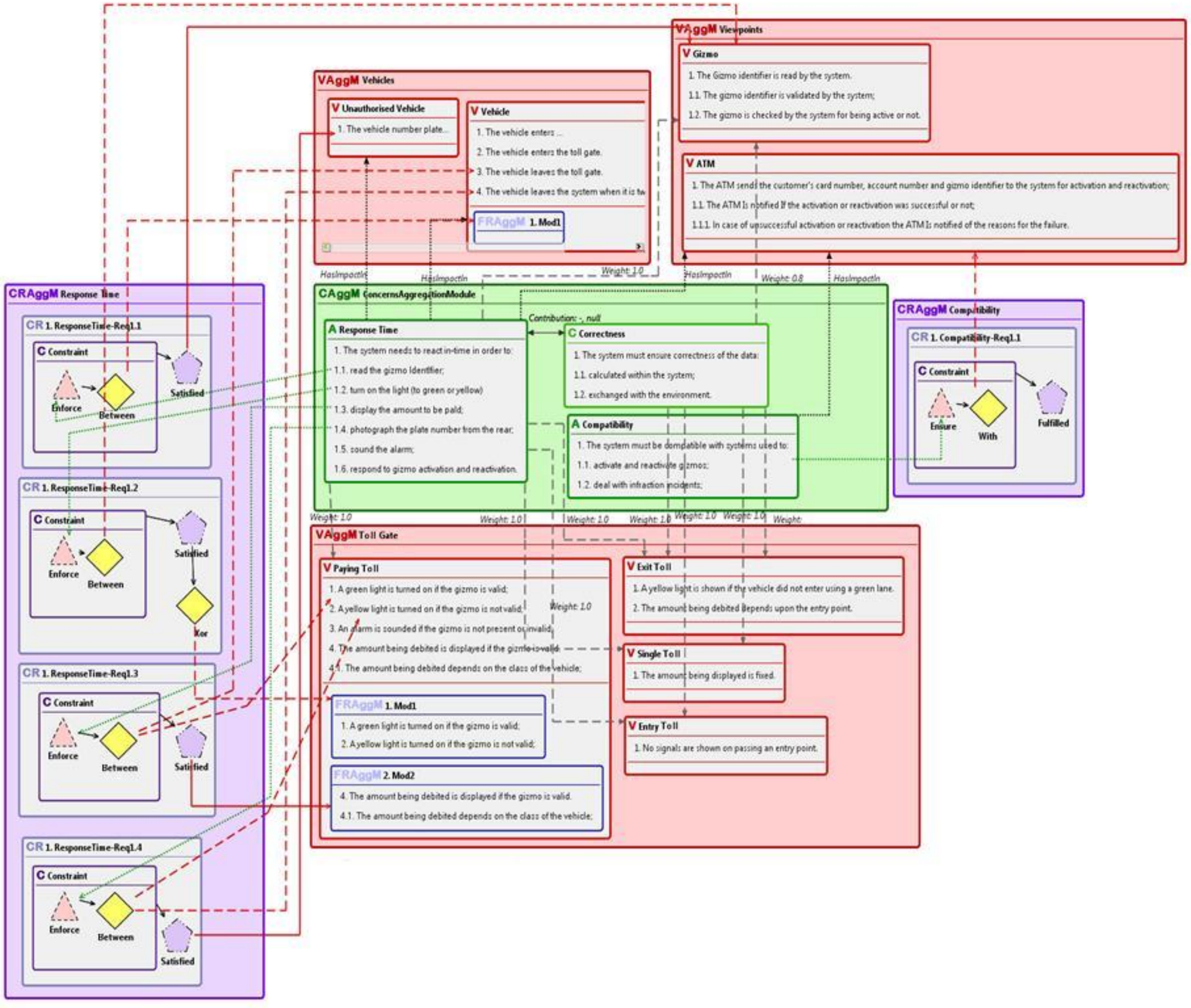

Figura 5.19. Modelo AORE do caso de estudo Via Verde.

O módulo de agregação de *concerns* possui a especificação dos *concerns Response Time*, *Compatibility* e *Correctness*. Dentro deste é possível observar as relações de contribuição existentes entre os *concerns*. Uma relação de contribuição existe caso haja uma relação de contribuição mútua de valor positivo ou negativo, ou apenas de um dos lados. Para escolher o tipo de contribuição possível, é necessário recorrer-se ao tab *Property Views* do editor, e alterar o valor da variável *type* do *link Contribution* para o valor desejado, dentro daqueles que lhe são oferecidos pelo tipo enumerado.

Os módulos de agregação de *viewpoints* são denominados *Vehicles*, *Toll Gate* e *Viewpoints*. Dentro do módulo *Vehicles* estão presentes os *viewpoints Vehicle* e *Unhauthorized Vehicle*. O *viewpoint Vehicle* possui no seu segundo compartimento o módulo de agregação de requisitos funcionais *Mod1*, pois a regra de composição *ResponseTime-Req1.1*. refere-se a dois requisitos deste *viewpoint*.

O módulo de agregação de *viewpoints Toll Gate* possui os *viewpoints Entry Tol*, *Exit Toll*, *Single Toll* e *Paying Toll*. Este útlimo contém dois módulos de agregação de requisitos funcionais, denominados *Mod1* e *Mod2*.

A atribuição de pesos a *viewpoints* para a resolução de conflitos existentes entre *concerns* é feita através da relação *Weight*. Esta relação tem como origem um *concern* e como destino um *viewpoint*. Após se inserir a relação, deve-se recorrer ao tab *Property Views* do editor, e alterar o valor da variável *weight* para o valor desejado, dentro dos valores oferecidos pelo tipo enumerado.

Caso haja um *concern* que exerça um impacto em determinado *viewpoint*, é utilizada a relação *HasImpact* para ilustrar esse impacto.

As setas que possuem cor vermelha são relações de ligação a requisitos funcionais, módulos de requisitos funcionais ou *viewpoints*. Destas, as que são tracejadas estabelecem relações cuja origem é um operador de restrição ou *Constraint Operator*. As setas com linha preenchida representam ligações cuja origem é uma acção resultante ou *Outcome Action*.

A seta verde ponteada é destinada a estabelecer a ligação entre os requisitos não funcionais dos *concerns* e as acções de restrição de cada regra de composição.

De notar que os elementos das regras de composição se encontram relacionados entre sí através de uma seta simples com o objectivo de estabelecer a ordem lógica da regra de composição.

5.5. Sumário

Neste capítulo é detalhado o processo de desenvolvimento da ferramenta VisualAORE. Na secção 5.1 é introduzida a análise de domínio realizada através de um diagrama de *features*. Nesta fase é feita uma análise dos conceitos que a linguagem abrange. Na secção 5.2 é abordada a fase de desenho da ferramenta VisualAORE, onde é introduzido o metamodelo proposto para a linguagem. A secção 5.3 fornece os detalhes principais da implementação da ferramenta, incidindo na explicação do procedimento da criação de sub-editores. A ferramenta é apresentada na secção 5.4 através da modelação do caso de estudo Via Verde.

6. Caso de Estudo

O caso de estudo utilizado para validar a ferramenta VisualAORE é a modelação de um sistema para uma *Smart Home*. Este caso de estudo foi retirado do projecto Europeu AMPLE[6], e encontra-se descrito em seguida.

Na Europa é habitual encontrarem-se casas com uma vasta gama de dispositivos eléctricos e electrónicos. Exemplos destes são luzes, termóstatos, interruptores eléctricos, sensores de fogo e de quebra de vidros, entre outros. Os sensores são dispositivos que medem as propriedades físicas do ambiente e as tornam disponíveis para o sistema *Smart Home*. Os actuadores activam os dispositivos cujos estados podem ser geridos e alterados.

O objectivo dos projectos na área das casas inteligentes é interligar os dispositivos e disponibilizar aos seus habitantes a sua gestão e controlar através de várias interfaces. Uma solução mais rudimentar permite controlar os dispositivos através de certas áreas técnicas dentro da casa, e executa aplicações centradas na casa. Por outro lado, uma solução mais ambiciosa integra mais tipos de dispositivos e inclui uma plataforma externa que disponibiliza acesso remoto e serviços externos. Tarefas como facturação, registo de utilizadores e gestão da plataforma fazem parte desta segunda solução.

A rede segundo a qual a casa funciona permite que dispositivos controlem o seu comportamento com o objectivo de executar as tarefas complexas. O estado dos dispositivos pode também ser alterado pelos habitantes, através de uma interface a eles destinada, ou pelo sistema, usando políticas predefinidas. Estas políticas permitem que o sistema reaja de forma autónoma a certos eventos. Por exemplo, em caso de detecção de fumo, as janelas devem fechar-se, as portas devem ser desbloqueadas e os bombeiros devem ser chamados.

[6] AMPLE, *Ample Project*, http://ample.holos.pt/, 2007.

Independentemente do nível de inteligência da casa, a sua personalização é um problema. Um utilizador deve ser capaz de definir as suas preferências e o sistema deve facilitar a configuração dos dispositivos. Pode ser necessário oferecer determinados serviços apenas a determinadas pessoas. Deste modo, são necessários mecanismos de autorização e autenticação, sendo que esta última se torna ainda mais importante quando o sistema contém uma plataforma externa e serviços externos.

Os requisitos funcionais mais importantes para uma plataforma S*mart Home* são gerir e alterar o estado dos dispositivos automaticamente, gerir as mudanças feitas manualmente, permitir que o sistema reaja de forma autónoma de acordo com políticas pré-definidas, personalização, autorização e autenticação. Os dispositivos que devem ser utilizados são luzes, interruptores de luzes, dispositivos de abertura de janela, sensores de janelas, persianas, sensores de quebra de vidro, radiadores e termóstatos, sensores de portas e dispositivos de abertura de portas, detectores de movimento e de luz, sensores de presença, detectores de fumo e fogo, sistema *sprinkler* (sistema de extinção de incêndios) e dispositivos de alarme.

O sistema *Smart Home* deve oferecer um alto nível de funcionalidade em que vários sensores e actuadores trabalham em conjunto. Este alto nível deve ser opcional, mas se for seleccionado, requer que determinados dispositivos sejam instalados na casa. As principais funções podem ser:

- **Sistema de energia**: termóstatos, aquecimento, dispositivos de abertura de janelas, dispositivos de abertura de portas e electrodomésticos devem ser combinados de forma a gastar o mínimo de energia possível.

- **Sistema de controlo de temperatura**: aquecimento, termóstatos, persianas e janelas devem ser combinadas para obter uma temperatura preferida nas divisões da casa.

- **Sistema de segurança**: sensores de vidros partidos, sensores de portas e detectores de movimento devem ser usados para detectar se alguma pessoa que não tem autorização para entrar em casa o tentou fazer. Se a casa detectar intrusão, ela deve dar um alarme ou com sirenes ou campainhas, ou informar a polícia ou uma empresa de segurança.

- **Sistema de Fumo e Fogo**: detectores de fogo e fumo, sistema de *sprinkler*, sensores de janela e portas e dispositivos de abertura e fecho, dispositivos de alarme e comunicação devem trabalhar juntos para prevenir danos humanos em caso de

fogo e fumo. Além disso, o fogo deve ser extinto e os bombeiros devem ser chamados. O sistema deve ser inteligente o suficiente para não começar a extinguir fogo caso haja apenas algumas pessoas a fumar numa sala.

- **Sistema de alarme:** o sistema de segurança é controlado através de um painel no *hall* de entrada. Este painel também funciona como uma máquina de resposta da casa. O uso do alarme irá activar o sistema central de bloqueamento. Isto irá fechar todas as janelas e portas e pode desligar qualquer dispositivo. Pode desligar as luzes quaisquer luzes que ficaram ligadas do dia. O sistema de segurança pode ser usado através de um telefone interno ou por ligação para a casa, de uma linha externa. Nestas situações, os comandos de voz actual do mesmo modo que usando o painel directamente. Qualquer um dos dispositivos do controlo remoto pode ser utilizado para lidar com o sistema de alarme.

6.1. Especificação textual

Os *viewpoints* identificados para o sistema são Habitante, Administrador, Bombeiros, Polícia, Operadora, Telemóvel, Sensores (de Luzes, Persianas, Portas, Janelas, Movimento, Fumo e Fogo, Temperatura Externa e Interna, Alarme e *Sprinkler*), Actuadores (de Portas, Luzes, Janelas, Persianas e Interruptores), e dispositivos (Ar Condicionado (AC), Alarme e *Sprinkler*). Tal como é focado na descrição da abordagem AORE, os *viewpoints* são especificados em XML. No entanto, nesta dissertação, a especificação dos mesmos é feita de forma informal, embora estruturada, com o objectivo de facilitar a sua leitura e compreensão. Nesta secção são apresentados alguns *viewpoints* e *concerns* considerados mais relevantes, sendo que os restantes podem ser consultados no Anexo F.

Abaixo pode ler-se a especificação dos *viewpoints* Administrador, Operadora, Sensor de Movimento, Alarme e *Sprinkler*.

Administrador

1. O administrador escolhe uma opção de configuração:
2. Se o administrador selecciona a opção de alterar configuração de utilizador:
    2.1. Insere os novos dados;
    2.2. Confirma as novas configurações;
3. Se o administrador selecciona registar novo utilizador:
    3.1. Insere os dados;
    3.2. Confirma os dados.

Operadora

1. Em caso de emergência:
    1.1. Recebe informação do tipo de emergência;
    1.2. Recebe informação do local da ocorrência;
    1.3. Faz chamada de emergência.
2. Se o utilizador pretende configurar o sistema através do telemóvel:
    2.1. Recebe chamada do habitante;
    2.2. Efectua comunicação entre o habitante e o sistema central.

Sensor de movimento

1. Detecta movimento no interior da casa;
2. Envia informação de existência de movimento e da(s) divisão/divisões em que esta foi detectada, para o sistema central.

Alarme

1. Se o sistema central receber informação das condições necessárias para que o alarme seja activado, ele activa-se;
2. Se o sistema central receber informação das condições necessárias para que o alarme seja desactivado, ele desactiva-se.

Sprinkler

1. Se o sistema central receber informação das condições necessárias para que o *sprinkler* seja activado, ele activa-se;
2. Se o controlo remoto receber informação das condições necessárias para que o *sprinkler* seja desactivado, ele desactiva-se.

Os assuntos (*concerns*) identificados para o sistema são Segurança (segurança de prevenção de crimes – alarme anti-roubo, câmaras de videovigilância, etc.), *Safety* (segurança de prevenção de incidentes – alarmes de fogo, inundação, etc.), Disponibilidade, Tempo de Resposta, Compatibilidade e Usabilidade. Abaixo encontra-se a especificação dos assuntos Segurança e Tempo de Resposta, também esta realizada de forma informal. Lembramos que um *concern* em PREview é não funcional.

Segurança

1. O sistema necessita de ter segurança de prevenção de crimes na identificação dos habitantes e do administrador;
2. O sistema necessita de ter segurança de prevenção de incidentes ao dar as permissões correctas a cada habitante e ao administrador;
3. O utilizador (administrador ou habitante) acede ao sistema depois de fazer login no sistema e introduz a senha:
    3.1. Se o utilizador inserir o login e a senha correctamente, têm acesso ao sistema.
    3.2. Se o utilizador inserir o login e/ou a senha incorrectamente, têm mais duas possibilidades de voltar a inserir a senha correcta;

3.2.1. Se o utilizador não inserir o login e/ou a senha correctamente em nenhuma das tentativas efectuadas, pode pedir assistência remota, para se identificar no sistema.

Tempo de resposta

1. O sistema deve possuir um bom tempo de resposta ao comunicar com a operadora;
2. O sistema deve possuir um bom tempo de resposta para ligar/desligar o *sprinkler*.
3. O sistema deve responder atempadamente para ligar/desligar o alarme.
4. O sensor de movimento deve responder atempadamente na detecção de movimento;
5. O sensor de fumo e fogo deve responder atempadamente na detecção de fumo e fogo;
6. Os actuadores devem responder atempadamente aos pedidos do sistema;
7. A polícia deve responder atempadamente aos pedidos de emergência;
8. Os bombeiros devem responder atempadamente aos pedidos de emergência;
9. O sistema deve responder atempadamente à identificação dos habitantes e do administrador;
10. O sistema deve responder atempadamente na verificação e aceitação das configurações feitas pelo habitante;
11. O sistema deve responder atempadamente na verificação e aceitação da alteração de dados de utilizadores registados e na verificação e aceitação do registo de novos utilizadores.

O passo seguinte consiste em elaborar as regras de composição ao nível dos requisitos individuais e não apenas dos módulos que os encapsulam (*viewpoints* e *concerns*), com vista a compor os requisitos do *concern* em estudo com os requisitos dos *viewpoint*s. As regras de composição, como a abordagem AORE foca, devem ser especificadas em XML, no entanto, também estas serão, nesta dissertação, feitas em linguagem informal. Abaixo pode ler-se a especificação da composição dos assuntos Segurança e Tempo de Resposta.

Composição de Segurança

1. O requisito 1 do *concern* Segurança deve ser satisfeito antes de todos os requisitos do *viewpoint* Administrador e do *viewpoint* Habitante;
2. O requisito 2 do *concern* Segurança deve ser satisfeito antes de todos os requisitos do *viewpoint* Administrador e do *viewpoint* Habitante;
3. O requisito 3 do *concern* Segurança deve ser satisfeito antes de todos os requisitos do *viewpoint* Administrador e do *viewpoint* Habitante.

Composição de Tempo de Resposta

1. O requisito 1 do *concern* Tempo de Resposta deve restringir os requisitos 1.3 e 2.2 da *viewpoint* Operadora;
2. O requisito 2 do *concern* Tempo de Resposta deve restringir todos os requisitos do *viewpoint Sprinkler*;
3. O requisito 3 do *concern* Tempo de Resposta deve restringir todos os requisitos do *viewpoint* Alarme;
4. O requisito 4 do *concern* Tempo de Resposta deve restringir todos os requisitos do *viewpoint* Sensor de Movimento;
5. O requisito 5 do *concern* Tempo de Resposta deve restringir todos os requisitos do *viewpoint* Sensor de Fumo e Fogo;
6. O requisito 6 do *concern* Tempo de Resposta deve restringir todos os requisitos dos actuadores;
7. O requisito 7 do *concern* Tempo de Resposta deve restringir o requisito 1 do *viewpoint* Polícia;

8. O requisito 8 do *concern* Tempo de Resposta deve restringir o requisito 1 do *viewpoint* Bombeiros;
9. O requisito 9 do *concern* Tempo de Resposta deve restringir todos os requisitos do *viewpoint* Habitante;
10. O requisito 10 do *concern* Tempo de Resposta deve restringir todos os requisitos do *viewpoint* Administrador.

A Tabela 6.1. ilustra as relações entre os *viewpoints* e os *concerns*, tal como foi introduzido no Capítulo 2. Como exemplo, pode observar-se que o *concern* Segurança possui impacto nos *viewpoints* Administrador e Habitante. Esta tabela fornece uma ideia inicial dos *concerns* que podem ser aspectos, como é o caso do *concern* Segurança.

Tabela 6.1. Impacto entre *viewpoints* e *concerns*.

Viewpoint / *Concern*	AJa	APe	APo	Ad	Ala	AC	Bo	Hab	Int	Ope	Pol	SAl	SPe	SFF	SJa	SLu	SMo	SPo	SSp	STE	STIn	Spr	Tlm
Segurança				√				√															
Safety					√					√		√	√	√	√	√	√	√	√	√	√	√	
Compatibilidade	√	√	√		√	√			√	√		√	√	√	√	√	√	√	√	√	√	√	√
Disponibilidade	√	√	√		√	√	√		√	√	√	√	√	√	√	√	√	√	√	√	√	√	
Usabilidade				√				√															
Tempo de Resposta	√	√	√	√	√	√	√	√	√	√	√	√		√			√		√			√	

Legenda:

AJa – Actuador de Janelas
APe – Actuador de Persianas
APo – Actuador de Portas
Ad – Administrador
Ala -Alarme
AC – Ar Condicionado
Bo – Bombeiros
Hab – Habitante
Int – Interruptor
Ope - Operadora
Pol – Polícia
SAla – Sensor de Alarme
SPe – Sensor de Persianas
SFF – Sensor Fumo e Fogo
SJa – Sensor de Janelas
SLu – Sensor de Luz
SMo – Sensor de Movimento
SPo – Sensor de Portas
SSp – Sensor Sprinkler
STE – Sensor de Temperatura Externa
STIn – Sensor de Tempreratura Interna
Spr - Sprinkler
Tlm – Telemóvel

A Tabela 6.2 possui as relações de contribuição entre os *concerns*. Pode observar-se, a título de exemplo, que existe um conflito entre os *concerns* Segurança e Tempo de Resposta, uma vez que contribuem negativamente um para o outro.

Tabela 6.2. Contribuições entre os *concerns*

Concern / ***Concern***	**Segurança**	***Safety***	**Compatibilidade**	**Disponibilidade**	**Usabilidade**	**Tempo de Resposta**
Segurança				+		-
Safety				-	+	-
Compatibilidade	+			+		+
Disponibilidade	-		+			-
Usabilidade	-	+				
Tempo de Resposta	-			+		

A Tabela 6.3 possui a resolução dos conflitos identificados. Esta resolução é feita através da atribuição de pesos aos *viewpoints* que são influênciados pelos *concerns* que entram em conflito. Pode observar-se o caso do conflito entre o *concern* Segurança e Tempo de Resposta, em que é dada prioridade à Segurança pois é *concern* que é considerado mais importante a ter em consideração neste ponto da especificação do sistema. Os *viewpoints* influenciados por este *concern* obtêm o peso 1 e quando são influenciados pelo *concern* Tempo de Resposta obtêm o peso 0.9.

Tabela 6.3. Resolução dos conflitos

Viewpoint / *Concern*	**AJa**	**APe**	**APo**	**Ad**	**Ala**	**AC**	**Bo**	**Hab**	**Int**	**Ope**	**Pol**	**SAl**	**SPe**	**SFF**	**SJa**	**SLu**	**SMo**	**SPo**	**SSp**	**STE**	**STIn**	**Spr**	**Tlm**
Segurança					1				1														
Safety						0.6					0.6		0.6	0.6	0.6	0.6	0.6	0.7	0.6	0.6	0.6	0.6	0.6
Compatibilidade	√	√	√	√		√	√			√	√		√	√	√	√	√	√	√	√	√	√	√
Disponibilidade	0.9	0.9	0.9	0.9		0.7	0.9	0.8		0.9	1	0.8	0.8	0.7	0.8	0.7	0.7	1	0.7	0.8	0.7	0.7	0.7
Usabilidade					0.9				0.9														
Tempo de Resposta	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.9	0.8	0.8	0.8	0.9	0.7		0.7			0.9		0.7			0.8

6.2. Especificação na ferramenta VisualAORE

Com o objectivo de obter uma avaliação da ferramenta, o caso de estudo *Smart Home* foi especificado na ferramenta VisualAORE, conforme mostra a figura .

Nesta especificação, os *viewpoints* foram divididos em cinco módulos de agregação: módulo de agregação de entidades externas, actuadores, sensores, dispositivos e finalmente, módulo de agregação de utilizadores. Na ferramenta, foram especificados todos os *viewpoints* e *concerns* e apenas algumas regras de composição.

A Figura 6.1 ilustra o módulo de agregação de entidades externas. Este módulo contém os *viewpoints* Bombeiro, Polícia, Operadora e Telemóvel. No *viewpoint* Operadora podem-se observar os módulos de agregação de requisitos funcionais utilizados nas regras de composição dos *concerns* Compatibilidade e Tempo de Resposta.

O módulo de agregação de *concerns* pode ser observado na Figura 6.2, onde é possível visualizar a especificação de todos os *concerns* do caso de estudo, assim como as contribuições que exercem entre si.

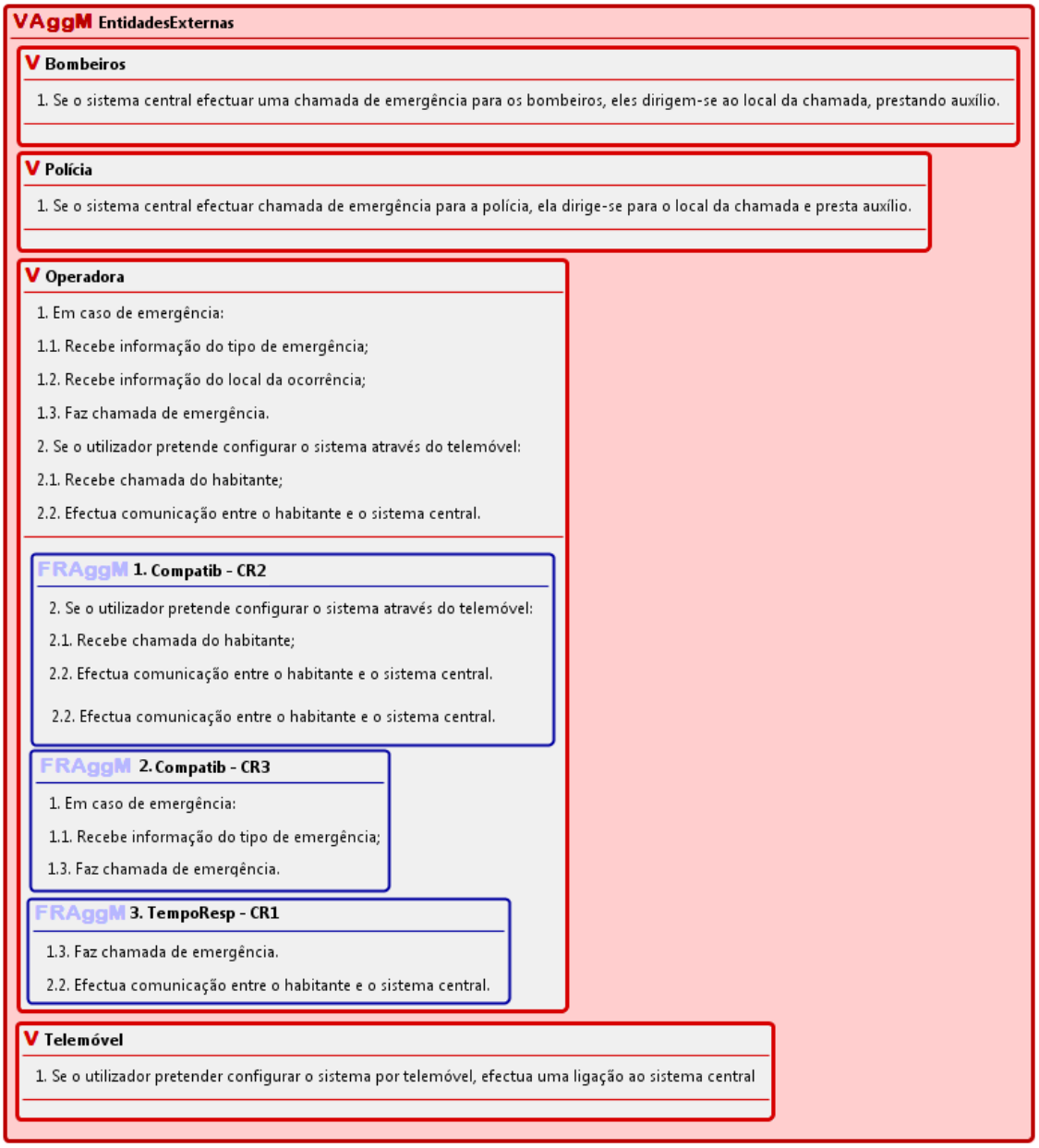

Figura 6.1. Módulo de agregação de entidades externas.

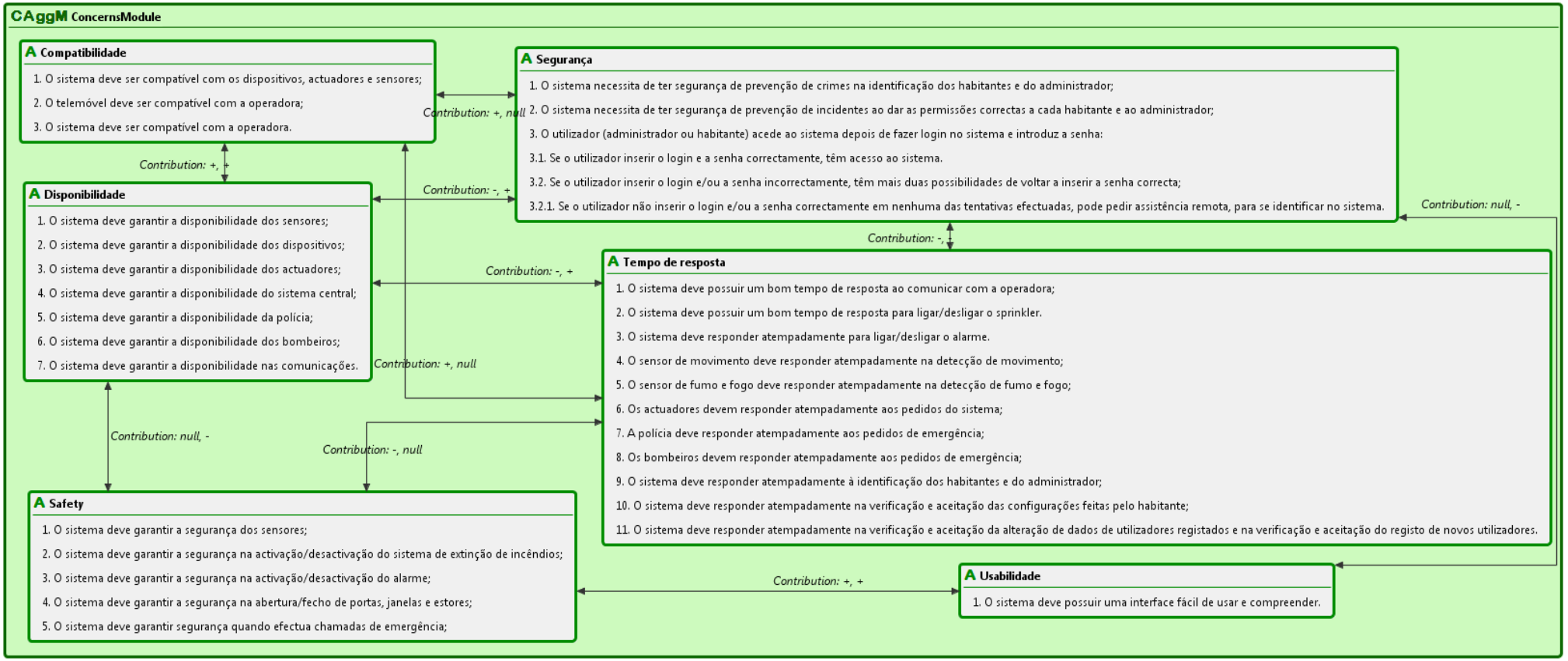

Figura 6.2. Módulo de agregação de *concerns*

O aspecto Segurança foi composto apenas nos seus requisitos 1, 2 e 3, como mostra a Figura 6.3. A regra de composição 1 significa que o requisito 1 de Segurança deve ser proporcionado aos requisitos dos *viewpoints* Administrador e Habitante, e resultado desta composição é o cumprimento deste *viewpoints*. A regra de composição 2 significa que o requisito 2 de Segurança deve ser proporcionado aos requisitos dos *viewpoints* Administrador e Habitante e o resultado desta composição é o cumprimento destes últimos requisitos. A regra 3 é semelhante às anteriores.

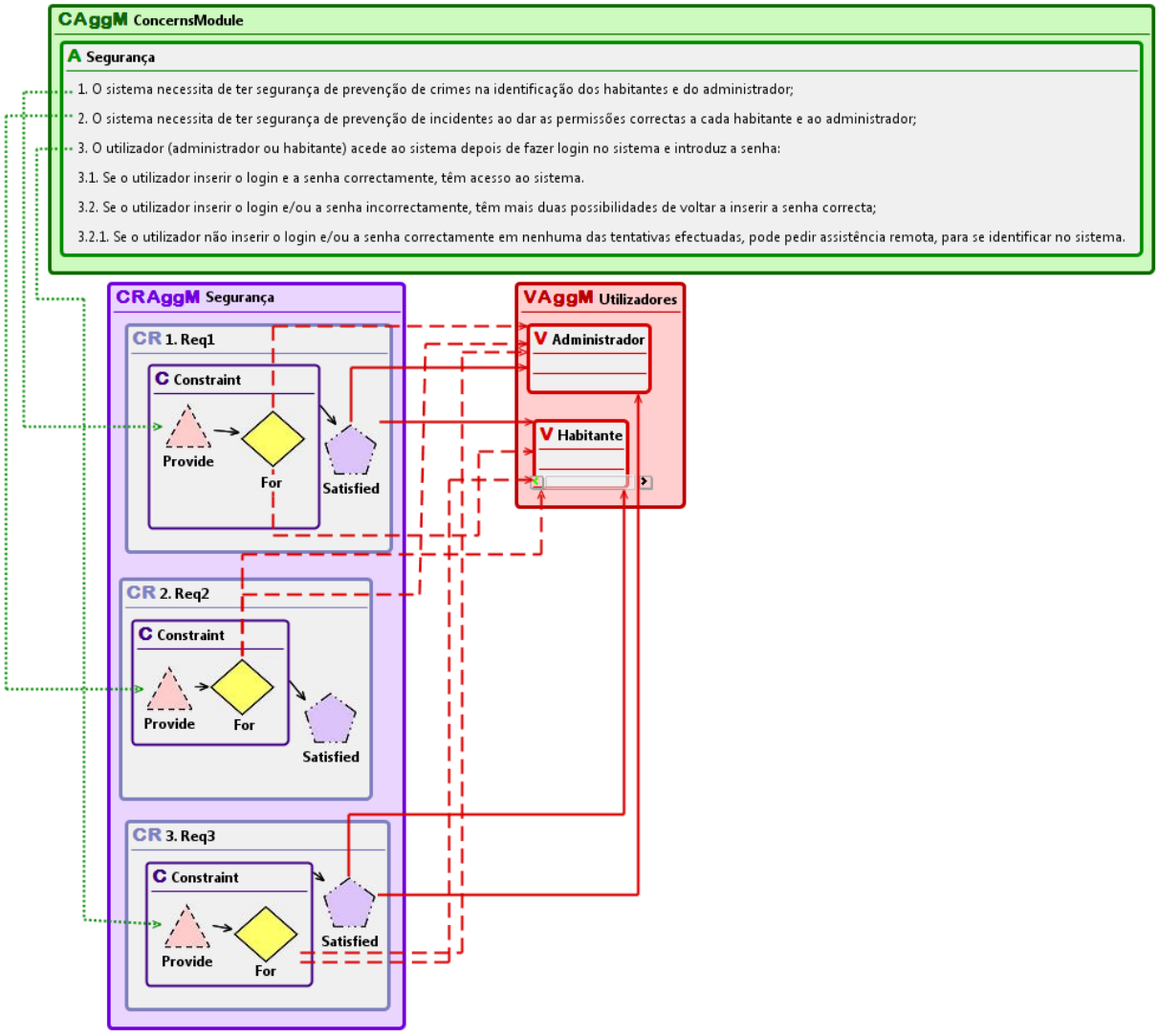

Figura 6.3. Regras de composição do *concern* Segurança.

Como a Figura 6.4 indica, o aspecto Tempo de Resposta apenas foi composto nos seus requisitos 1, 2, 3 e 4. A regra de composição 1 significa que o requisito 1 de Tempo de Resposta deve ser imposto aos requisitos 1.3 e 2.2 do *viewpoint* Operadora, e o resultado desta composição é o cumprimento destes últimos requisitos. A regra de composição 2 significa que o requisito 2 de Tempo de Resposta deve ser imposto aos requisitos do *viewpoint Sprinkler* e o resultado desta composição é o cumprimento dos requisitos do *Sprinkler*. A regra de composição 3 significa que o requisito 3 de Tempo de Resposta deve ser imposto aos requisitos do Alarme e o resultado desta composição é o cumprimento destes últimos requisitos. A regra de composição 4 significa que o requisito 4 de Tempo de Resposta deve ser imposto aos requisitos do *viewpoint* Sensor de Movimento e o resultado desta composição é o cumprimento dos requisitos deste *viewpoint*.

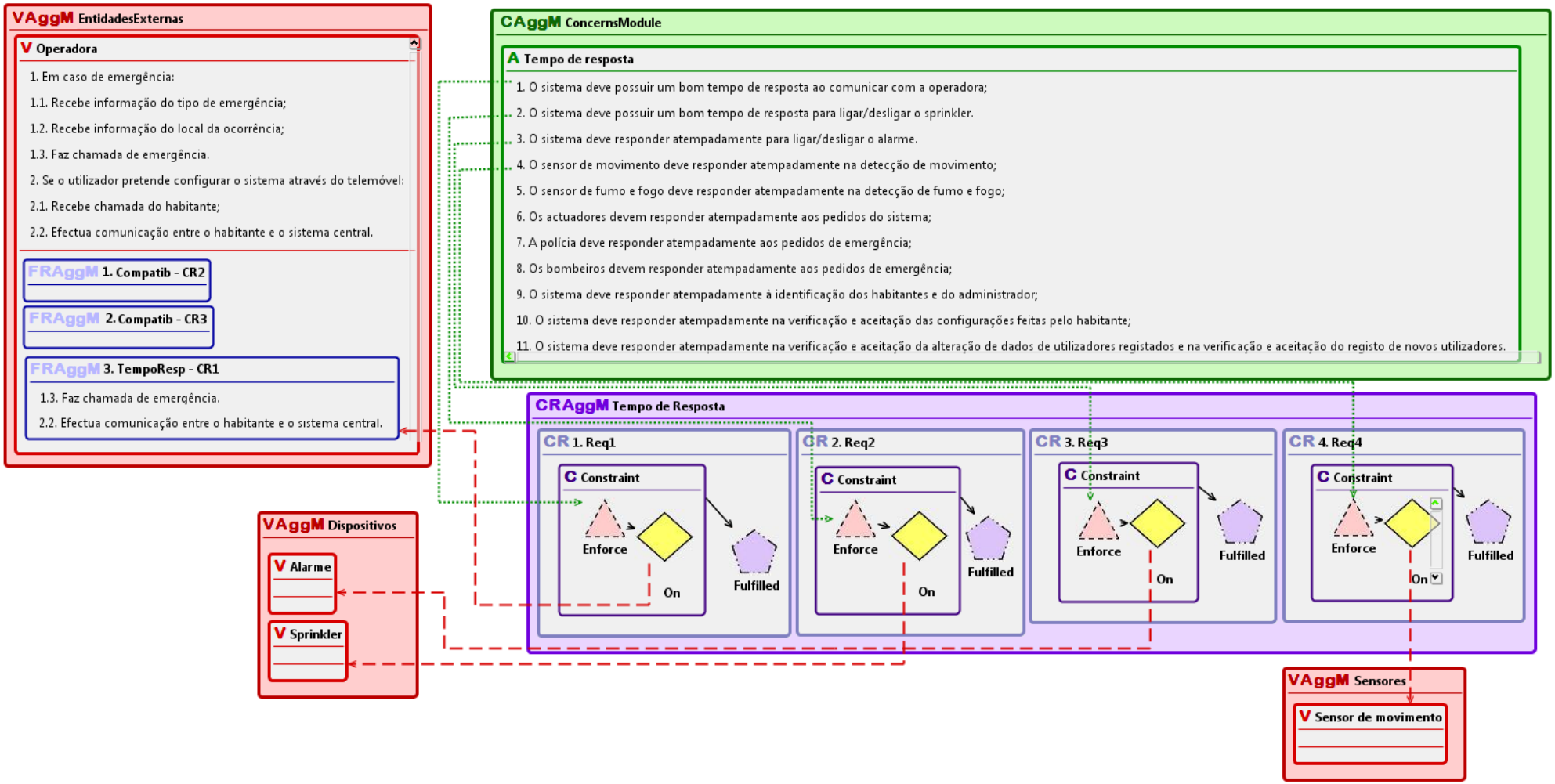

Figura 6.4. Regras de composição do *concern* Tempo de Resposta.

Como se pode ver de forma mais geral na Figura 6.5, foi realizada a composição para todos os requisitos do aspecto Compatibilidade. A regra de composição 1 significa que o requisito 1.1 de Compatibilidade deve ser assegurado no que diz respeito a todas as exigências dos actuadores, sensores e dispositivos. O resultado desta composição é o cumprimento dos *viewpoints* focados. A regra de composição 2 significa que o requisito 1.2 de Compatibilidade deve ser assegurado no que diz respeito ao requisito 2 e sub-requisitos do *viewpoint* Operadora e todos os requisitos do *viewpoint* Telemóvel. O resultado desta composição é o cumprimento dos requisitos 2 de Operadora e o *viewpoint* Telemóvel. A regra de composição 3 significa que o requisito 1.3 de Compatibilidade deve ser assegurado no que diz respeito ao requisito 1 e seus sub-requisitos do *viewpoint* Operadora. O resultado desta composição é o cumprimento do requisito 1 do *viewpoint* Operadora.

Para o aspecto *Safety* foram realizadas as regras de composição dos requisitos 2 e 3. A regra de composição 2 significa que o requisito 2 de *Safety* deve ser proporcionado aos requisitos do *viewpoint Sprinkler*, e o resultado desta composição é o cumprimento destes últimos requisitos A regra de composição 3 significa que o requisito 3 de *Safety* deve ser fornecido aos requisitos do *viewpoint* Alarme, e o resultado desta composição é o cumprimento destes últimos requisitos.

A especificação apresentada na Figura 6.5 não possui a representação dos pesos entre o aspecto Tempo de Resposta e os *viewpoints* dos actuadores, do Habitante e do Administrador.

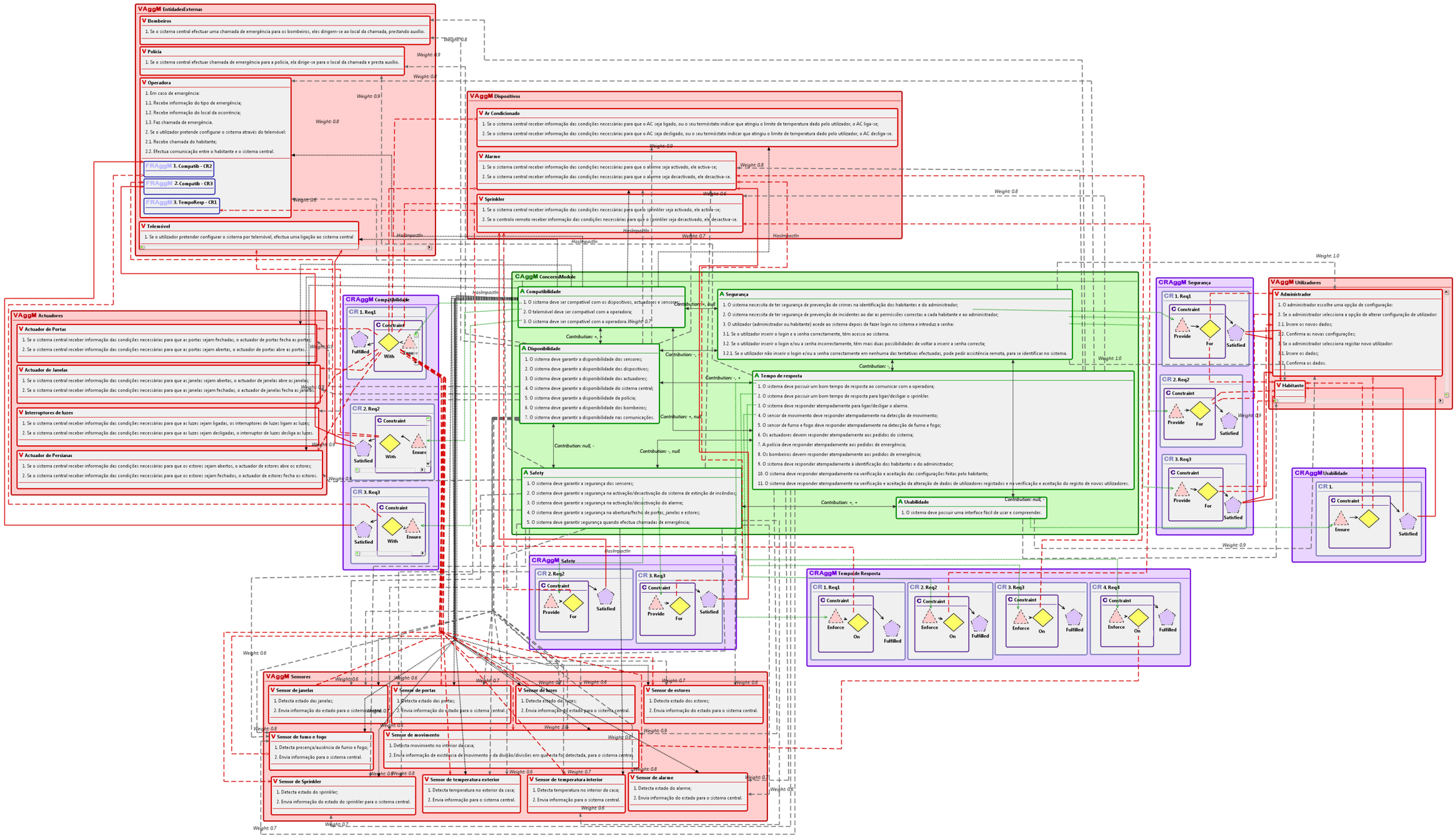

Figura 6.5. Modelo AORE do caso de estudo Smart Home.

6.3. Sumário

Neste capítulo é apresentado o caso de estudo *Smart Home*. Na secção 6.1 o caso de estudo é resolvido textualmente. Na secção 6.2 é apresentada uma proposta de resolução visual do caso de estudo na ferramenta VisualAORE e é dada uma breve explicação da mesma, recorrendo ao modelo obtido.

7. Avaliação Experimental

Com vista a efectuar a avaliação da LDE foi efectuado um questionário a um grupo de 16 utilizadores. Estes utilizadores testaram a ferramenta através da modelação do caso de estudo Via Verde. O caso de estudo foi fornecido aos utilizadores com uma solução textual pré-feita que pode ser consultada no Anexo H, para que estes se pudessem dedicar inteiramente à exploração da ferramenta.

O grupo de utilizadores convidado a efectuar o teste, é composto por pessoas com conhecimento em LDEs, Engenharia de Requisitos e Desenho de Software ou ambos. Este grupo de utilizadores frequenta ou frequentou no ano lectivo 2008/2009 o Mestrado em Engenharia Informática na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa. Após o teste da LDE, os utilizadores foram convidados a preencher um questionário relativo ao teste efectuado.

O questionário consiste numa adaptação de (Gabriel, 2010) e divide-se em 3 partes. Na primeira parte é obtido o grau de conhecimento dos utilizadores na área de LDEs. A segunda parte consiste numa avaliação conceptual com questões directamente relacionadas com a linguagem. Por fim surgem as questões relacionadas com a usabilidade da ferramenta e sugestões ou comentários de âmbito geral. O questionário pode ser consultado no Anexo I.

Na secção seguinte são apresentadas as questões do questionário, através de uma breve explicação do que se pretende avaliar com cada uma. Por fim é feita uma análise estatística das mesmas.

7.1. Análise das questões e dos resultados obtidos

O objectivo do grupo de questões A é obter o grau de competência dos utilizadores na área de LDEs, e o seu gosto pela mesma.

Questão A1. Quantas vezes possuíu contacto com um pacote de ferramentas de desenvolvimento de LDEs? (“How often did you use a DSL workbench tool? ”)

Nesta questão pretende-se saber quantas vezes é que um utilizador já possuiu contacto com um ambiente de desenvolvimento de LDEs. O valor 1 significa que o utilizador já possuíu imensos contactos com um ambiente de desenvolvimento de LDEs e o valor 5 significa que o utilizador não possuíu qualquer contacto.

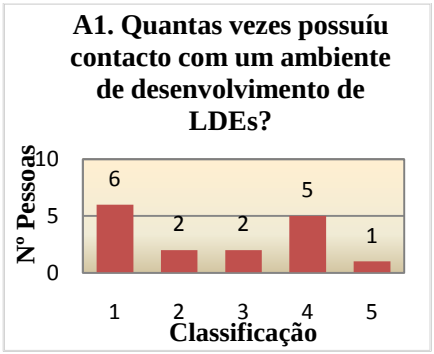

Figura 7.1. Análise da frequência de utilização de ambientes de desenvolvimento de LDEs.

No gráfico da Figura 7.1 observa-se que existem 6 utilizadores com conhecimento na área de LDEs, 2 utilizadores com bom conhecimento e 2 utilizadores com conhecimento razoável.

Questão A1.1. Durante quanto tempo utilizou? (“How long have you used it?”)

Com esta questão pretende-se saber o período de tempo que cada utilizador que já utilizou um ambiente de desenvolvimento de LDEs esteve em contacto com o mesmo.

Tabela 7.1. Período de tempo de utilização de um ambiente de desenvolvimento de LDEs.

Período de Tempo (semestres)	Nº Pessoas
1	2
2	1
3	1
4	1
6	2
8	1

Questão A1.2. Gostou de utilizar? ("Have you enjoyed it?")

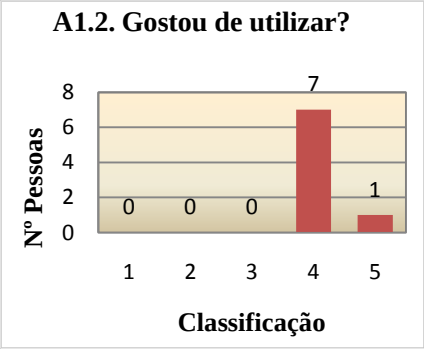

Figura 7.2. Análise do gosto dos utilizadores pela área de LDEs.

Do conjunto de utilizadores peritos na área de LDEs é notável um elevado gosto pela mesma.

Questão A1.3. Em que circunstâncias utilizou? ("What for?")

Todos os utilizadores tiveram o seu primeiro contacto com a área de LDEs na cadeira de Linguagens e Domínios Específicos leccionada na Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, onde foi realizada uma LDE textual capaz de oferecer um comportamento ao robot *Lego NXT*. Após esta experiência, 4 utilizadores trabalharam na área ao longo da sua dissertação de mestrado. Existem ainda 2 pessoas que trabalharam nessa área após concluir a dissertação de mestrado feita na área. Existe uma pessoa que não efectuou a dissertação na área de LDEs mas trabalhou em projectos da área posteriormente. De forma geral, os utilizadores afirmam que utilizam ambientes de desenvolvimento de LDEs para desenhar, especificar e implementar linguagens visuais ou textuais, e ainda para definir e validar metamodelos. Associado à especificação da linguagem vem a produção de editores de modelação e transformação e o domínio de sistemas de controlo complexos. Estes ambientes de desenvolvimento são, portanto, utilizados para produzir linguagens e ferramentas essencialmente na área de Engenharia de Software.

As questões do grupo B têm o objectivo de fornecer uma validação ao nível conceptual. Este grupo de questões pretende obter o grau de compreensão e facilidade que linguagem VisualAORE oferece.

Questão B1. Compreendeu a linguagem VisualAORE? ("Did you understand the VisualAORE language?")

Esta questão tem o objectivo de avaliar se a linguagem VisualAORE é fácil de compreender. No gráfico abaixo (Figura 7.3), o valor 1 corresponde a uma compreensão muito fraca da LDE VisualAORE, e o valor 5 corresponde a uma óptima compreensão da mesma.

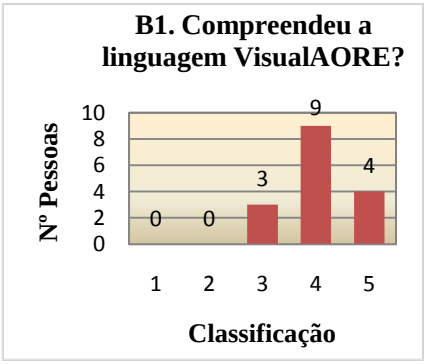

Figura 7.3. Análise do grau de compreensão da linguagem VisualAORE.

Como se pode verificar no gráfico, 9 utilizadores consideram que a linguagem VisualAORE é fácil de compreender, e 4 utilizadores consideraram mesmo muito fácil.

Questão B1.1. Caso não tenha compreendido, o que faltou? ("*If not, what was missing?*")

Os utilizadores que compreenderam a linguagem de forma razoável apontam o problema ao facto de não estarem muito familiarizados com a metodogia AORE no geral. Alguns utilizadores não compreenderam de imediato a diferença entre um elemento *Viewpoint* e um *ViewpointModule*. Também as regras de composição não foram percebidas de imediato por uma minoria dos utilizadores.

Questão B2. Qual o grau de facilidade em aprender os conceitos da linguagem? ("How easy did you find learning the concepts?")

Com a questão B2 pretende-se avaliar se os conceitos da linguagem são fáceis de compreender e assimilar. O valor 1 significa que os conceitos são demasiado difíceis de aprender e o valor 5 significa que os conceitos são muito fáceis de aprender.

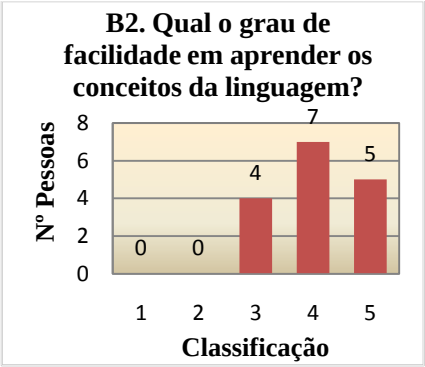

Figura 7.4. Análise da facilidade em aprender os conceitos da linguagem.

A maioria dos utilizadores considera que os conceitos da linguagem são fáceis de assimilar e compreender, 5 pessoas consideram mesmo muito fáceis.

Questão B3. Como identifica os símbolos que representam os conceitos? ("*How do you identify the symbols representing the concepts?*")

Nesta questão avalia-se se a sintaxe concreta é adequada, ou seja, se oferece um fácil reconhecimento dos conceitos através dos seus símbolos. O valor 1 significa muito difícil e o valor 5 significa fácil.

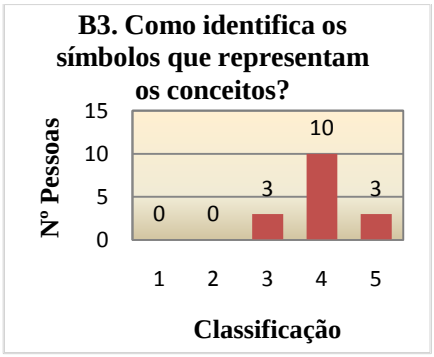

Figura 7.5. Análise da facilidade de reconhecimento dos símbolos.

Pelos resultados do gráfico da Figura 7.5 é notório que a identificação dos símbolos foi feita de forma facilitada, não havendo problemas significativos.

Questão B3.1. Qual o símbolo que considerou inadequado? ("*Which one did you find inadequate?*")

A questão B3.1 tem o objectivo de obter os símbolos da linguagem que não são capazes de representar os conceitos de forma sugestiva e não ambígua.

Alguns utilizadores consideram que os ícones amarelos que representam os elementos *Constraint Operator* são pouco legíveis pois são demasiado claros. Foi também referido que os ícones dos elementos *Constraint Actions Enforce*, *Ensure* e *Exclude* deveriam ser

diferentes, dado que para todos ele o símbolo é igual. Houve também quem considerasse que a sintaxe concreta atribuída aos módulos de *Viewpoints* e aos *Viewpoints* era confusa.

Questão B4. Como identifica o texto que representa os conceitos? (“*How do you identify the text representing the concepts?”*)

Esta questão possui a finalidade de avaliar se a sintaxe concreta é adequada, ou seja, se oferece um fácil reconhecimento dos conceitos através do seu texto. O valor 1 significa muito difícil e o valor 5 significa muito fácil.

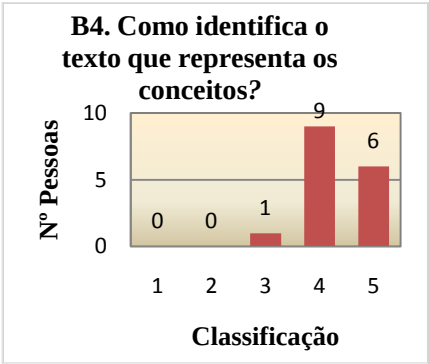

Figura 7.6. Análise da dificuldade em identificar o texto que representa os conceitos.

A maioria dos utilizadores considerou que o texto representativo dos conceitos é fácil de identificar.

Questão B4.1. Qual o texto que achou inadequado? (“*Which one did you find inadequate?*”)

Alguns utilizadores consideraram que o nome *Module* associado aos módulos de agregação não é sugestivo, pois não fornece a ideia de agrupamento. Para resolver esse conflito, foi adicionado “A*greggation”* ao nome dos módulos. A título de exemplo, o nome *ViewpointsModule* dado ao módulo de *viewpoints* foi alterado para *ViewpointsAggregationModule*.

Questão B5. **Quantas vezes cometeu erros devido a semelhanças entre símbolos?** (“*How often did you find committing errors due to symbols similarity?*”)

A questão B5 pretende avaliar se existem conceitos cujos símbolos são semelhantes, levando a erros na elaboração dos modelos. O valor 1 indica que nunca foram cometidos erros, e o valor 5 indica que foram cometidos com elevada frequência.

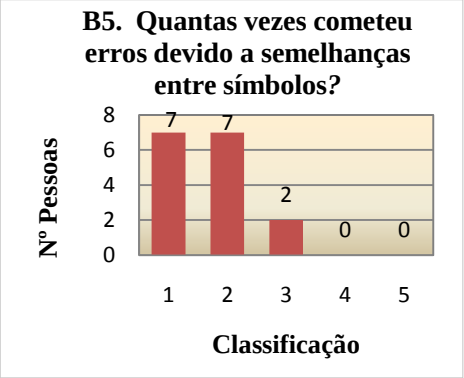

Figura 7.7. Análise de erros devido a semelhança entre símbolos.

Através da observação da Figura 7.7 conclui-se que apenas 2 pessoas cometeram uma percentagem razoável de erros. As restantes pessoas, de forma geral, não cometeram erros. Esta análise indica que, de forma geral, a sintaxe concreta da linguagem não leva a confusões entre os símbolos devido à sua possível semelhança.

Questão B6. Quantas vezes cometeu erros devido a vocabulário ambíguo? ("*How often did you find committing errors due to ambiguous vocabulary?*")

A questão B6 pretende avaliar se é frequente cometer-se erros devido à existência de um vocabulário ambíguo. O valor 1 significa que não foram cometidos erros, e o valor 5 significa que foram cometidos erros sempre.

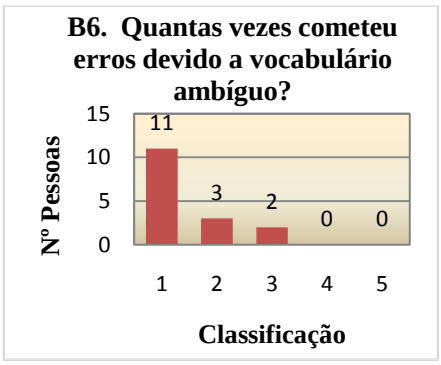

Figura 7.8. Análise de erros causados devido a vocabulário ambíguo.

Apenas duas pessoas cometeram alguns erros devido à existência de vocabulário ambíguo. A esmagadora maioria não cometeu erros. Esta análise indica que o vocabulário utilizado na linguagem não é ambíguo, e portanto não leva à produção de erros aquando da especificação de modelos.

Questão B7. Que mudanças ou características adicionais propõe à linguagem? ("*What changes or additions do you propose to the language?*")

A questão B7 tem como objectivo recolher sugestões de alteração à linguagem em teste.

Alguns utilizadores consideram que seria interessante colocar numeração automática nos requisitos. Também foi sugerido alterar o nome *Module* para *Group*.

As questões do grupo C avaliam a usabilidade da linguagem e do editor assim como as funcionalidades de modelação deste último.

Questão C1. Quanto tempo despendeu a desenvolver o modelo através do plug-in VisualAORE? ("*What time did you spent in developing the model through the VisualAORE plug-in?*")

Nesta questão pretende-se obter a duração média necessária para elaborar o caso de estudo fornecido na ferramenta.

Tabela 7.2. Duração da elaboração do caso de estudo.

Duração (minutos)	Nº pessoas
150	1
100	2
90	10
80	1
70	1
60	1

O tempo médio gasto para efectuar o modelo AORE da Via Verde foi de uma hora e 30 minutos.

Questão C2. Qual a facilidade com que elaborou o modelo AORE? ("*How easily did you create the AORE model?*")

A questão C2 possui o objectivo de avaliar a facilidade em criar modelos AORE na ferramenta VisualAORE. O valor 1 significa muito difícil e o valor 5 significa muito fácil.

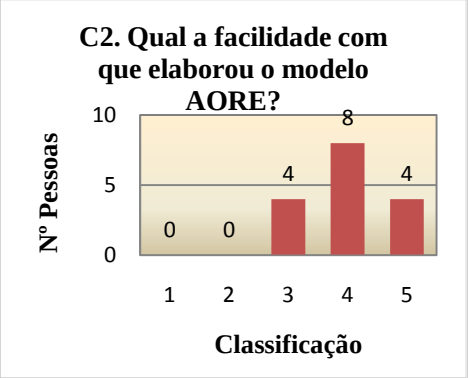

Figura 7.9. Análise da facilidade de elaboração de modelos AORE.

Por observação da Figura 7.9 pode concluir-se que de forma geral existe facilidade em criar modelos AORE.

Questão C3. Como avalia o processo de elaborar um modelo no plug-in VisualAORE? ("*How do you evaluate the process of creating a model in the VisualAORE plug-in?*")

A questão C3 tem como objectivo recolher a opinião dos utilizadores quanto ao processo de elaborar modelos AORE através do *plug-in* VisualAORE. O valor 1 indica que o processo é muito mau, e o valor 5 indica que o processo é muito bom.

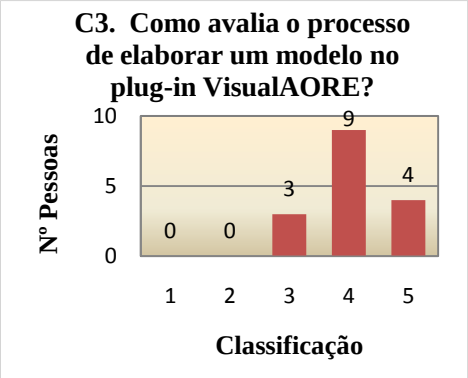

Figura 7.10. Análise da qualidade do processo de elaboração de modelos através do *plug-in* VisualAORE

A maioria dos utilizadores considera que o processo de elaborar um modelo na ferramenta é bom.

Questão C4. Possuiu dificuldade a utilizar o *plug-in* VisualAORE? ("*Did you have difficulty using the VisualAORE plug-in?*")

A questão C4 tem o objectivo de avaliar se houve dificuldades em utilizar o *plug-in* VisualAORE.

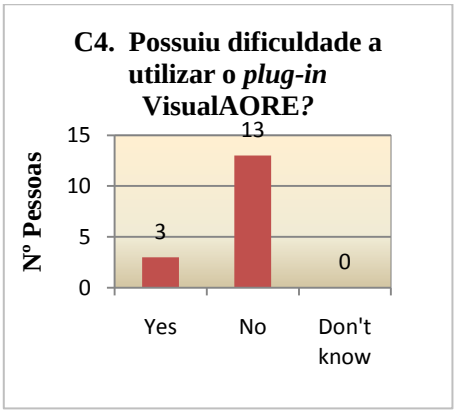

Figura 7.11. Análise da dificuldade em utilizar o *plug-in* VisualAORE.

Como se pode observar na Figura 7.11, a grande maioria de utilizadores não obteve dificuldades na utilização do *plug-in* VisualAORE.

Questão C4.1. Se sim, qual a maior dificuldade que encontrou? ("*If yes, what's the biggest difficulty you found?*")

A questão C4.1 tem o objectivo obter as dificuldades específicas que os utilizadores sentiram ao utilizar o *plug-in* VisualAORE.

Alguns utilizadores sentiram dificuldade na medida em que alguns conceitos da técnica AORE já se encontravam esquecidos. Um utilizador considerou que é difícil encontrar um *link* específico entre aqueles que o menu disponibiliza. No entanto as necessidades mais sentidas foram na elaboração das regras de composição.

A maioria dos utilizadores que não possuem experiência na utilização de editores desenvolvidos na plataforma GMF mostrou uma dificuldade considerável na utilização do *plug-in* VisualAORE que se prende com as limitações da plataforma, tais como organizar os modelos de forma *user-friendly* e definitiva.

Questão C5. Como se sentiu em relação a efectuar mudanças nos modelos? ("*How did you feel about performing changes?*")

A questão C5 pretende avaliar o processo de efectuar alterações aquando do desenho de modelos AORE. O valor 1 significa que é muito difícil efectuar mudanças nos modelos e o valor 5 significa que é muito fácil efectuar mudanças nos modelos.

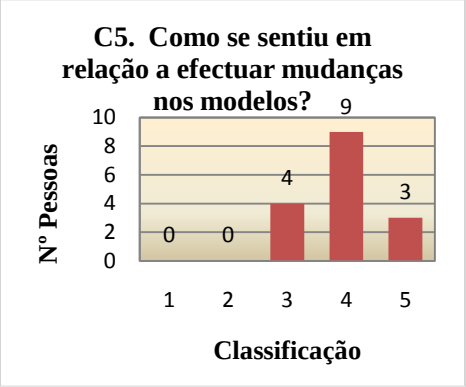

Figura 7.12. Análise da capacidade em realizar alterações nos modelos.

Através da observação da Figura 7.12 conclui-se que os utilizadores não consideram difícil efectuar mudanças nos modelos AORE.

Questão C6. Qual o esforço físico necessário para efectuar o caso de estudo? ("*How physically demanding was performing the case study?*")

A questão C6 pretende avaliar o esforço mental necessário para realizar o modelo AORE do caso de estudo Via Verde. O valor 1 significa que é muito difícil desempenhar o caso de esstudo e o valor 5 significa que é muito fácil.

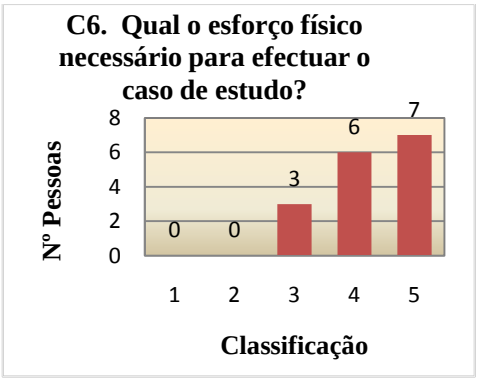

Figura 7.13. Análise do esforço físico necessário à realização do caso de estudo.

Os utilizadores consideraram que é fácil desempenhar o caso de estudo.

Questão C7. O resultado reflecte o que esperava? ("*The outcome reflects what you were expecting?*")

Esta questão pretende obter a capacidade da LDE em permitir que os peritos do domínio realizem tarefas específicas com precisão e perfeição, avaliando a sua eficácia. O valor 1 significa que o resultado não reflecte o que o utilizador esperava, e o valor 5 significa que o resultado corresponde exactamente ao que o utilizador esperava.

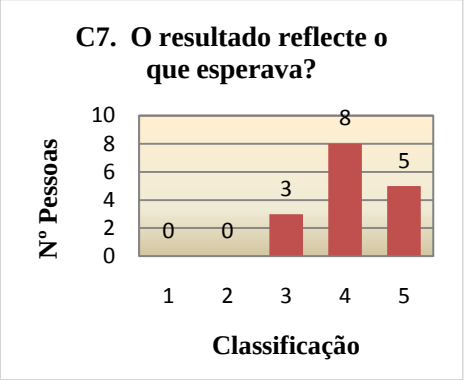

Figura 7.14. Análise dos resultados esperados.

Através da observação da Figura 7.14 pode-se constatar que na maioria das vezes o resultado obtido no teste foi o esperado.

Questão C8. Quantas vezes se sentiram incapaz de exprimir o que pretendia? ("*How often did you feel unable to express what you intended?*")

Na questão seguinte pretende-se avaliar se a LDE é compacta e restrita ao exprimir as intenções do utilizador. O valor 1 significa que o utilizador nunca se sentiu incapaz de exprimir o que desejava e o valor 5 significa que o utilizador se sentiu sempre incapaz de exprimir o que desejava.

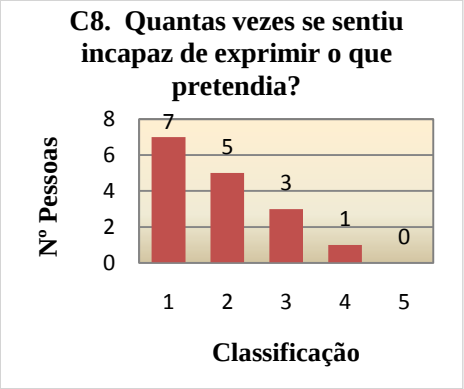

Figura 7.15. Análise da frequência com que o utilizador se sentiu incapaz de exprimir o que desejava.

A Figura 7.15 mostra que na maioria das vezes os utilizadores sentiram-se capazes de exprimir o que desejavam. No entanto houve 2 experiências em que os utilizadores se sentiram incapazes de exprimir o que desejavam. Este facto pode dever-se a serem utilizadores que estavam menos familiarizados com a técnica AORE ou com LDEs.

Questão C9. Quais as vantagens do modelo obtido através do *plug-in* relativamente à especificação feita textual e manualmente? ("*What are the advantages of the model obtained in the plug-in relatively to the specification done textual and manually*?")

A questão C9 tem a finalidade de obter as vantagens do modelo feito no modelo AORE relativamente à especificação textual fornecida.

As principais vantagens referidas pelos utilizadores são:

- O modelo possui uma especificação esquematizada;
- O modelo possui uma melhor organização visual;
- O modelo possui maior facilidade e rapidez de compreensão;
- É modelado mais rapidamente;
- O modelo permite a visualização dos módulos de agregação envolvidos no modelo separadamente (sub-editores);
- O modelo é feito numa ferramenta que oferece uma fácil utilização;
- O modelo permite obter uma ideia imediata de todos os elementos que o constituem.
- O modelo oferece a distinção de cores dos elementos permitindo uma fácil e rápida distinção e identificação dos mesmos.
- O modelo é mais preciso pois a ferramenta restringe o acontecimento de alguns erros.
- O modelo permite identificar a ocorrência de erros mais facilmente.
- O modelo oferece uma melhor compreensão da relação entre os *viewpoints* e os *concerns*.
- O modelo oferece uma maior interactividade, facilidade de correcção e evolução do padrão;
- O modelo obtido pode ser integrado com outras ferramentas, para verificação e transformação;
- O editor ao evoluir poder fornecer uma assistência maior ao processo de criação de modelos.

Questão C10. E as desvantagens? (“*And the disadvantages?*”)

A questão C10 pretende obter as desvantagens do modelo feito no *plug-in* VisualAORE em relação à especificação textual fornecida.

- O modelo não tem a plena liberdade e extensibilidade da descrição em papel;
- As limitações do GMF reflectem-se no desenvolvimento e resultados das soluções;
- Efectuar o *print* de soluções grandes, pode tornar-se complicado.

- A existência de uma curva de aprendizagem um pouco acentuada para utilizadores com poucos conhecimentos nas áreas de Linguagens de Domínio Especifico e Engenharia de Requisitos e Desenho de Software;
- Ao ser menos textual, o modelo pode correr o risco de ser menos explicativo;

Questão C11. O modelo especificado no *plug-in* contém toda a informação da especificação feita textual e manualmente? ("*When creating the model in the plug-in, did you kept all the information specified textual and manually?*")

A questão C11 pretende avaliar se o modelo feito na ferramenta é capaz de garantir toda a informação da especificação textual fornecida, não havendo portanto perdas de informação.

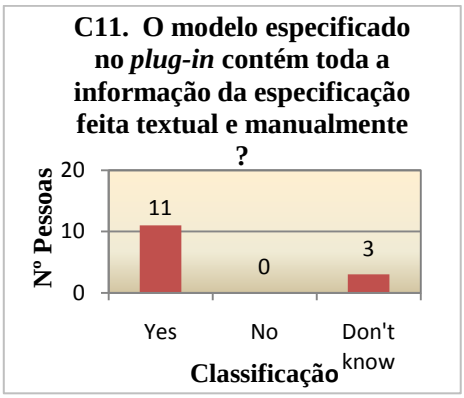

Figura 7.16. Análise da informação obtida, comparando com a especificação feita textual e manualmente.

A maioria dos utilizadores são da opinião de que o modelo efectuado na ferramenta possui toda a informação da especificação fornecida textualmente. No entanto, existem 3 utilizadores que não conseguiram responder a esta questão.

Questão C12. Quantas vezes fez questões ao supervisor? ("*How often did you perform questions to the supervisor?*")

A questão C12 pretende obter o poder da LDE em ser assimilada e compreendida, através da análise da necessidade de ajuda aquando da especificação de modelos. O valor 1 significa que nunca houve necessidade de pedir ajuda ao supervisor para a execução do teste e o valor 5 significa que foi sempre necessário pedir ajuda ao supervisor.

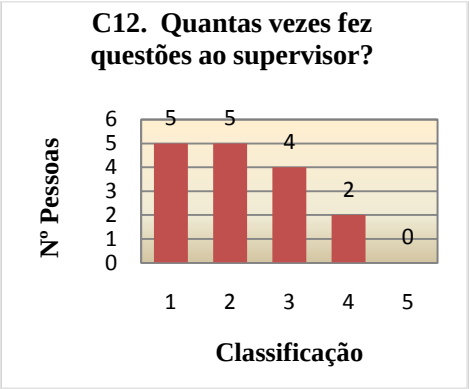

Figura 7.17. Análise da frequência com que o utilizador recorreu ao supervisor.

A Figura 7.17 mostra que a maioria dos utilizadores não necessitou de recorrer à ajuda do supervisor. No entanto, uma quantidade um pouco significativa necessitou de efectuar um pouco mais de perguntas.

As questões seguintes pretendem testar a facilidade em aplicar a ferramenta ao caso de estudo fornecido.

A questão C13 tem o objectivo de obter o grau de confiança na ferramenta aquando da especificação de um caso de estudo. O valor 1 significa que os utilizadores não se sentiram confiantes e o valor 5 significa que os utilizadores se sentiram muito confiantes.

Questão C13. Qual o grau de confiança que sentiu durante a execução do caso de estudoenário? ("*How confident did you feel during the case study execution?*")

Esta questão possui a finalidade de obter o grau de confiança dos utilizadores aquando da utilização do *plug-in*.

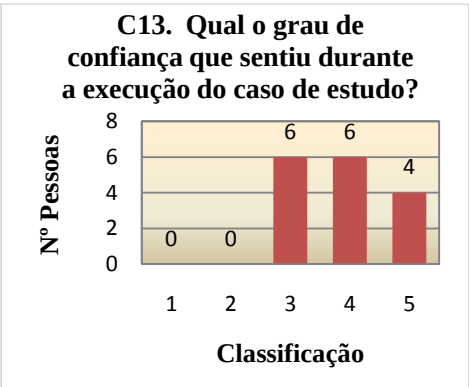

Figura 7.18. Análise da confiança dos utilizadores durante a elaboração do caso de estudo.

De acordo com a Figura 7.18, a maioria dos utilizadores sentiu-se confiante aquando da execução do caso de estudo. Este resultado permite concluir que a ferramenta oferece um nível bom de confiança à produção de modelos AORE.

Questão C14. Quantas vezes se sentiu incapaz ou confuso durante a execução do caso de estudo? ("*How often did you find trapped or confused during the case study?*")

A questão C14 tem o objectivo de avaliar o grau de confusão dos utilizadores durante a execução do caso de estudo. O valor 1 significa que o utilizador nunca se sentiu confuso aquando da execução do caso de estudo, e o valor 5 significa que o utilizador se sentiu sempre confuso.

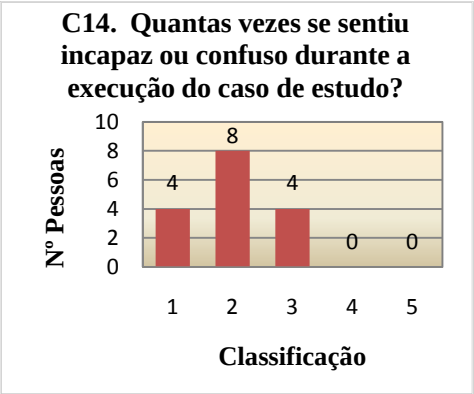

Figura 7.19. Análise da frequência com que o utilizador se sentiu confuso durante a execução do caso de estudo.

Os resultados mostrados na Figura 7.19 apontam para o facto de os utilizadores não se sentirem confusos, ou pouco confusos aquando da execução do caso de estudo.

Questão C15. Qual o esforço mental necessário para executar o caso de estudo? ("*How mentally demanding was the case study?*")

A questão C15 possui a finalidade de avaliar a dificuldade mental em executar o caso de estudo. O valor 0 significa que a dificuldade é muito elevada e o valor 5 significa que a dificuldade é nula.

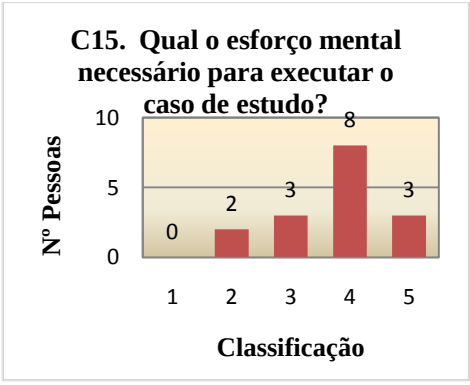

Figura 7.20. Análise do esforço mental necessário para realizar o caso de estudo.

De acordo com a Figura 7.20, a maioria dos utilizadores considerou o caso de estudo fácil. No entanto, duas pessoas consideraram o caso de estudo difícil. É possível que estas últimas pessoas não possuam conhecimentos significativos na área de ERDS.

Questão C16. Em que é que sentiu mais dificuldade de desempenho ou raciocínio? (“*What did you feel more difficult to reason/perform?* “)

A questão C16 tem o objectivo de recolher as maiores dificuldades encontradas na utilização da ferramenta para a elaboração do caso de estudo.

As dificuldades encontradas foram:

- Relembrar conceitos já esquecidos;
- Organizar a informação no editor de modo a ser mais facilmente visualizada;
- Realizar as regras de composição.

Questão C17. Qual a sua opinião acerca da correcção do caso de estudo efectuado? (“*How do you feel about the correctness of the performed case study?*”)

A questão C17 tem o objectivo de avaliar a correcção dos modelos efectuados através da ferramenta VisualAORE. O valor 1 significa que o utilizador considera que o modelo obtido não é correcto, o valor 5 significa que o modelo é muito correcto.

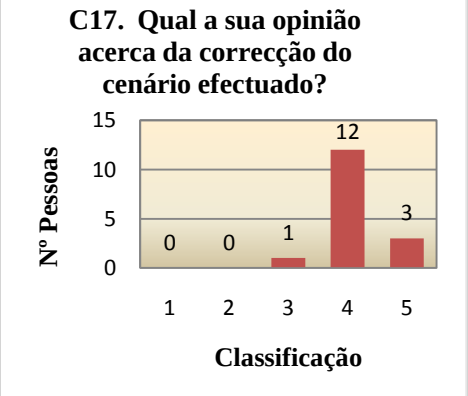

Figura 7.21. Análise da correcção do caso de estudo.

De acordo com os resultados obtidos, pode afirmar-se que a maioria dos utilizadores considera que a ferramenta oferece um elevado nível de correcção.

Questão C18. Já utilizou a ferramenta ARCaDe? (“*Have you ever used the ARCaDe tool?*”)

A questão C18 pretende avaliar o conhecimento dos utilizadores em ferramentas que lidam com a metodologia AORE.

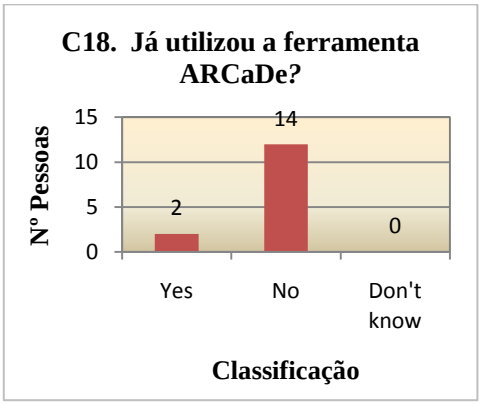

Figura 7.22. Análise da utilização da ferramenta ARCaDe.

Apenas duas pessoas já trabalharam com a ferramenta ARCaDe.

Questão C18.1. Como classifica o plug-in VisualAORE relação à ferramenta ARCaDe? ("*How do you rate de VisualAORE plug-in comparing with the ARCaDe tool?*")

Esta questão tem como objectivo comparar as ferramentas VisualAORE e ARCaDe. O valor 1 significa uma má classificação e o valor 5 significa uma excelente classificação.

As duas pessoas que conhecem a ferramenta ARCaDe consideram que a ferramenta apresentada nesta dissertação possui o valor 4 quando comparada com a ferramenta ARCaDe.

Questão C18.2. O plug-in VisualAORE contém mais informação do que a ferramenta ARCaDe? ("*The VisualAORE plug-in contains more information than the one that is provided by the ARCaDe?*")

Os utilizadores consideram que uma vez que se trata de uma ferramenta visual, é mais rápido obter e compreender a informação. No entanto, a informação ainda que representada de modo diferente, é a mesma, e portanto não existe mais informação.

Questão C18.3. O plug-in VisualAORE contém menos informação do que a ferramenta ARCaDe? ("*The VisualAORE tool contains less information than the one provided by the ARCaDe?*")

Os utilizadores consideram que a informação é a mesma.

Questão C18.4. Comparando a ferramenta VisualAORE com a ARCaDe, qual, na sua opinião, oferece uma melhor e mais rápida compreensão dos modelos? ("*Comparing*

VisualAORE with ARCaDe, what's the tool you think that offers a faster and better understanding of the models?")

Ambos os utilizadores são da opinião de que a ferramenta VisualAORE fornece uma maior compreensão dos modelos.

Questão C18.5. Considera que a ferramenta VisualAORE é uma ferramenta de valor acrescentado quando comparada com a ARCaDe? ("*Do you feel the VisualAORE tool is a value-added compared to ARCaDe?*")

Ambos os utilizadores consideram que sim.

Questão C18.6. Porquê? ("*Why?*")

Os utilizadores são da opinião de que "A ARCaDe *é uma ferramenta que usa templates em XML, o que obriga a uma compreensão e domínio nesta linguagem, além de que facilmente uma pessoa se perde nas estruturas dos templates. O VisualAORE consegue contornar esta questão apresentando um aspecto visual bastante conseguido". "Como a ferramenta VisualAORE não é baseada em texto, torna-se mais atractiva e devido ao facto de ser baseada na metodologia drag&drop, a construção de modelos é facilitada. É também mais fácil visualizar as relações entre os conceitos."*

Questão C19. Considera a ferramenta útil? ("*Do you consider the tool helpful?*")

A questão C19 tem o objectivo de avaliar o grau de utilidade da ferramenta em teste. O valor 1 significa que a ferramenta não é definitivamente útil e o valor 5 significa que a ferramenta é muito útil.

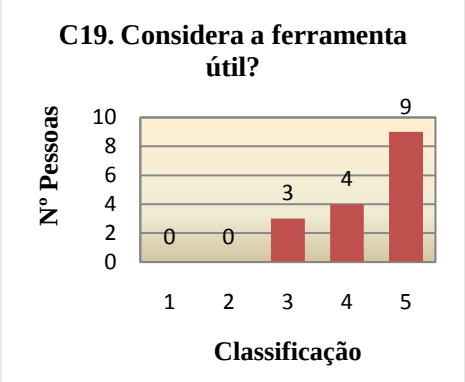

Figura 7.23. Análise da utilidade da ferramenta VisualAORE.

A maioria dos utilizadores considerou a ferramenta muito útil. Três utilizadores consideraram a ferramenta apenas razoavelmente útil.

Questão C20. Qual é a sua apreciação global da LDE VisualAORE? (“*What is your overall appreciation of the VisualAORE DSL?*”)

A questão C20 tem como objectivo obter a apreciação global da ferramenta VisualAORE. O valor 1 significa apreciação global muito negativa, e o valor 5 significa apreciação global óptima.

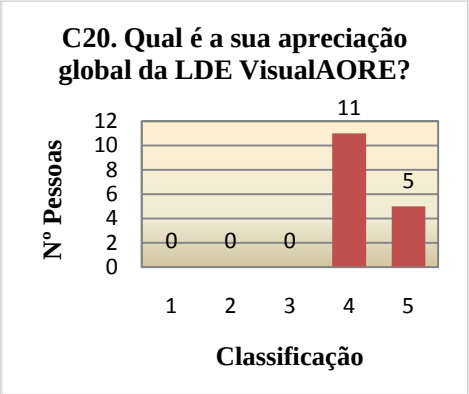

Figura 7.24. Análise da apreciação global da LDE.

A maioria dos utilizadores atribui uma boa apreciação global à ferramenta. Cinco utilizadores atribuíram mesmo uma óptima apreciação.

As questões que se seguem são de resposta livre. O objectivo é avaliar a opinião dos utilizadores em relação a algumas características da ferramenta.

Questão C21. Quais são, na sua opinião, as fraquezas/dificuldades do *plug-in* VisualAORE? (“*What are, in your opinion, the weaknesses/difficulties of the VisualAORE plug-in?*”)

O objectivo da questão C21 é obter as fraquezas ou as dificuldades encontradas no *plug-in* VisualAORE.

As respostas a esta questão focaram os seguintes pontos:

- O *plug-in* possui alguma instabilidade, e modelação redundante dentro e fora dos sub editores;
- O *plug-in* possui um menu complex0;
- O *plug-in* permite agregações dentro de agregações desnecessárias;
- A complexidade visual em caso de modelos grandes, pois pode ser dificil relacionar as entidades do modelo como um todo, essencialmente quando o zoom é normal (100%);

- O posicionamento dos elementos, especialmente quando eles não estão colapsados devia ajustar-se automaticamente ao modelo;
- O espaço de edição dentro de cada objecto é pequeno e por vezes os objectos filhos ficam cortados, não sendo possível vê-los na sua totalidade, no entanto, este problema é defeito do EMF/GMF e não do VisualAORE;
- O *plug-in* possui bugs de usabilidade devido ao GMF;
- O GMF impõe muitas limitações gráficas;
- O facto de se tratar de um *plug-in* faz com que a ferramenta será sempre dependente do Eclipse;
- A criação das regras de composição é o processo mais difícil.

Questão C22. Quais são, na sua opinião, os pontos fortes/facilidades do plug-in VisualAORE? ("*What are, in your opinion, the strengths/facilities of the VisualAORE plug-in?*")

A questão C22 tem como objectivo saber quais os pontos fortes ou as facilidades oferecidas pelo *plug-in* VisualAORE.

Os pontos fortes identificados são os seguintes:

- O *plug-in* possui uma forte sintaxe visual, muito agradável e bastante directa;
- O *plug-in* oferece facilidade de identificação dos módulos e dos restantes elementos através das cores;
- O *plug-in* oferece a possibilidade de colapsar as entidades dos modelos;
- O *plug-in* oferece o conceito de sub editores, onde se pode definir cada objecto com mais pormenor, e evitar a ocorrência de alguns erros;
- O *plug-in* oferece o encapsulamento de conceitos, o que permite uma melhor organização e gestão dos conceitos;
- O *plug-in* oferece a possibilidade de visualizar as ligações entre aspectos e *viewpoints*, o que facilita a validação da própria composição das regras;
- A plataforma do Eclipse é boa permitindo a criação de *plug-ins* com facilidade;
- O *plug-in* oferece facilidade de uso.

Questão C23. Quais são as inovações oferecidas pelo *plug-in* VisualAORE? ("*What are the innovations brought by the VisualAORE plug-in?*")

Os utilizadores consideram que o *plug-in* VisualAORE oferece as seguintes inovações:

- Trata-se de uma linguagem visual para efectuar especificação de modelos AORE;
- Facilidade de especificação;
- Facilidade de inserir e eliminar elementos;
- Permite a agregação de entidades.

Questão C24. O que sugere que pode melhorar o *plug-in* VisualAORE? ("*What do you suggest that can improve the VisualAORE plug-in?*")

Os utilizadores sugeriram as seguintes melhorias ao *plug-in* VisualAORE:

- Reduzir para apenas uma ferramenta o grupo das ferramentas;
- Utilizar *views* ao invés de sub-editores e introduzir código para facilitar a filtragem das *views*;
- Proteger a identificação dos requisitos, não permitindo que haja requisitos com o mesmo identificador.
- Melhorar aspectos de usabilidade.
- Utilizar uma ferramenta *stand-alone* ao invés de usar um *plug-in*.
- Diminuir o número de relações representadas por setas.
- Introduzir o a*uto-complete* das regras de composição, disponibilizando apenas as ligações possíveis aquando das ligações.
- Melhorar os símbolos das regras de composição, efectuando um símbolo diferente para cada elemento.

Questão C25. Sugestões / comentários gerais: ("*Suggestions / global comments*:")

A questão C25 é a última questão do questionário, como tal tem o objectivo de recolher possíveis sugestões ou comentários que os utilizadores não tenham tido oportunidade de expor até então. Os comentários obtidos foram os seguintes:

- Trata-se de uma ferramenta bastante útil;
- Oferece um elevado nível de expressividade aos peritos;
- Disponibilidade de uma ferramenta que suporta a especificação de elementos usados normalmente;
- Ferramenta interessante de utilizar e conhecer.

As respostas a esta questão possuem um carácter de comentário geral. Deste modo, pode-se concluir que o *plug-in* efectuado nesta dissertação obteve uma boa aceitação na medida em

que se trata de uma ferramenta útil, e que oferece um elevado nível de expressividade aos peritos, contrariamente à única ferramenta desenvolvida na área até agora (ARCaDe).

7.2. Ameaças à avaliação

Um processo de avaliação pode conter ameaças que colocam em causa os resultados e as conclusões obtidas.

A avaliação efectuada nesta dissertação teve como base os testes efectuados a um grupo de 16 alunos da faculdade, que possuem pouca ou nenhuma experiência profissional. Estes utilizadores estão em contacto permanente com as tecnologias mais actualizadas que nem sempre existem na indústria. Este facto é visto como uma mais-valia para o processo de avaliação realizado. No entanto, não consta deste processo qualquer validação na área industrial, pelo que seria conveniente efectuar uma avaliação da LDE no seio empresarial e industrial. Esta experiência forneceria um maior grau de confiança nos resultados obtidos, pois os testes seriam efectuados a uma amostra composta por profissionais experientes na área abrangida pelo *plug-in*. Um reforço da avaliação na área empresarial permitiria ainda a generalização dos resultados ao meio profissional.

O número de casos de teste efectuado (16) pode ser visto como uma ameaça à avaliação da ferramenta por se tratar de uma amostra reduzida. Ainda assim, esta amostra de estes já é significativa, e portanto é capaz de originar conclusões fiáveis. Para uma certificação dos resultados obtidos, seria interessante avaliar a LDE com base num número mais significativo de utilizadores.

De focar que o questionário preenchido pelos utilizadores foi devidamente supervisionado pelos orientador e co-orientador desta dissertação, minimizando o risco de erros.

7.3. Sumário

Neste capítulo foi detalhada a fase de avaliação da linguagem e do editor propostos nesta dissertação.Esta fase foi feita através de um teste efectuado a vários utilizadores, que posteriormente preencheram um questionário acerca do teste efectuado. Na secção 7.1 é introduzida cada questão do questionário de forma individual seguida da sua explicação e objectivos. Por fim é apresentado o resultado obtido na respectiva questão. A secção 6.3 foca alguns aspectos que são considerados ameaças à validação efectuada.

8. Conclusão

Nesta dissertação foi elaborada uma LDE gráfica para a metologia AORE denominada VisualAORE.

A linguagem VisualAORE foi realizada no ambiente de desenvolvimento Eclipse, na plataforma GMF/EMF e com o auxílio dos plug-ins Emfatic e EuGENia. O seu desenvolvimento assentou na elaboração de um metamodelo para linguagem que abrange todos os conceitos do método AORE e um novo conceito, o conceito de módulo de agregação. Este novo conceito permite a agregação de elementos do mesmo tipo dentro de um elemento chamado módulo de agregação. Nesta dissertação foram propostos três módulos de agregação: módulo de agregação de *viewpoints*, módulo de agregação de *concerns* e módulo de agregação de regras de composição. Cada módulo de agregação referido contém um sub-editor associado. O conceito de módulo de agregação foi também utilizado para agrupar requisitos funcionais dentro de um *viewpoint*, que participem numa mesma regra de composição. A utilização de um módulo de agregação de requisitos funcionais diminui o número de relações existentes no modelo, aumentando a sua escalabilidade. Os módulos de agregação, de forma geral, fornecem uma maior organização aos modelos, pois permitem a sua minimização, ocultando a informação que agregam. Esta característica facilita a leitura dos modelos AORE. A introdução de sub-editores nos módulos de agregação de *viewpoints*, *concerns* e regras de composição facilita a edição de modelos AORE, pois oferece a possibilidade de editar os elementos num editor em separado. Os sub-editores permitem também a visualização dos elementos de cada módulo separadamente, o que facilita a sua leitura.

A linguagem proposta nesta dissertação oferece uma notação gráfica à técnica AORE através de uma linguagem visual e de um editor que suporta a linguagem. Actualmente, a metodologia AORE é suportada somente pela ferramenta ARCaDe, que, como foi referido

anteriormente, especifica os elementos do método através de templates XML. A linguagem VisualAORE torna a metodologia AORE mais compreensível e facilita a especificação de modelos AORE através de uma representação gráfica. O editor que suporta a linguagem melhora o processo de modelação de requisitos através da técnica AORE para o Engenheiro de Software comum, dada a sua preferência por modelos visuais.

8.1. Limitações

Tal como grande parte das ferramentas de modelação de sistemas, a ferramenta desenvolvida nesta dissertação apresenta problemas ao nível da escalabilidade dos modelos. Este facto é evidente quando se pretende visualizar os modelos na sua integridade, como um todo. A modelação de sistemas origina modelos de tamanho elevado, o que complica a sua visualização. No entanto, o *plug-in* VisualAORE oferece a possibilidade de ocultar informação e visualizar e/ou editar determinadas porções de informação em editores separados. Esta característica facilita a leitura dos modelos quando esta é feita por partes.

O *plug-in* VisualAORE é suportado pelo ambiente de desenvolvimento Eclipse, pelo que a sua utilização está condicionada ao uso do mesmo.

A plataforma GMF fornece algumas limitações pois possui uma curva de aprendizagem um pouco acentuada, pelo que se torna por vezes difícil aprender a utilizar os plug-ins que esta suporta. Uma vez que se trata de uma plataforma em desenvolvimento, espera-se que a sua usabilidade melhore, passando a ser mais fácil lidar com este tipo de ferramentas.

O facto da avaliação da LDE VisualAORE não ter sido efectuada num ambiente industrial/empresarial pode ser também uma limitação a considerar.

8.2. Trabalho futuro

O metamodelo proposto neste trabalho pode ser estendido com vista a abranger algumas características do método VAODA. Com esta integração, a metodologia AORE ficaria mais completa pois resolveria problemas de *crosscuting concerns* não funcionais e funcionais. O metamodelo proposto nesta dissertação pode também ser utilizado para obter uma linguagem textual e um editor que a suporte, de forma a obter uma ferramenta textual mais simples que a ARCaDe.

A avaliação efectuada ao *plug-in* VisualAORE poderia ser refeita ao nível empresarial/ industrial, abrangendo um maior número de testes. Esta avaliação forneceria resultados mais precisos e abrangentes.

Com o objectivo de melhorar a usabilidade da plataforma GMF/EMF, devem ser efectuadas mudanças essencialmente ao nível da visualização dos elementos e da organização dos modelos num editor. Seria interessante a realização de um novo mecanismo de disposição de elementos automático, de acordo com o espaço disponível e os elementos possuídos.

Dado que a ferramenta ARCaDe é a única ferramenta que dá suporte ao método AORE, seria interessante elaborar um mecanismo de tradução entre os modelos XML obtidos e modelos VisualAORE.

9. Bibliografia

(AMPLE, 2007) AMPLE, *Ample Project*, http://ample.holos.pt/, 2007.

(Achenbach e Ostermann, 2010) M. Achenbach, K. Ostermann *Growing a Dynamic Aspect Language in Ruby*, em *AOSD '10*, Rennes e Saint Malo, França, 2010.

(Araújo e Moreira, 2003) J. Araújo e A. Moreira, *An Aspectual Use-Case Driven Approach*, em VIII Jornadas de Engenharia de Software e Bases de Dados (JISBD), págs. 463-468, Thompson, Novembro de 2003.

(Araújo *et* al., 2005) J. Araújo, E. Baniassad, P. Clements, A. Moreira, A. Rashid, B. Tekinerdoğan, *Early Aspects: The Current Landscape*, Relatório Técnico, Fevereiro de 2005.

(Atkins *et* al., 1999) D. Atkins, T. Ball, G. Bruns, K. Cox, *Mawl: a Domain-specific Language for Form-based services*, em *IEEE Transactions on Software Engineering*, V.25, Nº 3, págs. 334-346, Maio e Junho de 1999.

(Baniassad e Clarke, 2002) E. Baniassad e S. Clarke, *Theme: An Approach for Aspect-Oriented Analysis and Design*, em *Proceedings of the International conference on RE, IEEE CS Press*, págs. 199-202, Departamento de Computação Científica, Universidade de Trinity, Dublin, Irlanda, 2002.

(Baniassad *et* al., 2006) E. Baniassad, P. Clements, J. Araújo, A. Moreira, A. Rashid, B. Tekinerdogan, *Discovering Early Aspects*, em *IEEE Software*, Fevereiro de 2006.

(Bézivin, 2005) J. Bézivin, *On the unification power of models, ATLAS Group, (INRIA & LINA)*, Universidade de Nantes, França, 2005.

(Brand *et* al., 2001), M. Brand, A. Deursen, J. Heering, H. Jong, M. Jonge, T. Kuipers, P. Klint, L. Moonen, P. Olivier, J. Scheerder, J. Vinju, E. Visser e J. Visser, *The ASF +SDF*

Meta-environment: A Component-Based Language Development Environment, V. 2027/2001, págs. 365-370, Editora Springer, Berlin/Heidelberg, Janeiro de 2001.

(Brito, 2008) I. Brito, *Aspect-Oriented Requirements Analysis*, Dissertação apresentada para obtenção do grau de Doutor em Informática pela Universidade Nova de Lisboa, Faculdade de Ciência e Tecnologia, Quinta da Torre, Caparica, Portugal, 2008.

(Brito e Moreira, 2003) I. Brito, A. Moreira, *Advanced Separation of Concerns for Requirements Engineering*, em VIII Jornadas de Engenharia de Software e Bases de Dados (*JISBD*), Alicante, Espanha, 2003.

(Budinsky *et* al., 2003) F. Budinsky, S. Brodsky e E. Merks, *Eclipse Modeling Framework*, Editora Pearson Education, 2003.

(Chauvel *et* al., 2007) F. Chauvel, Z. DreyEngineer e F. Fleurey, *Kermeta Language Overview - The Triskell Metamodeling Language*, 31 de Janeiro de 2007.

(Chitchyan *et* al., 2007), R. Chitchyan, A. Rashid, P. Rayson, R. Waters, *Semantics-based Composition for Aspect-Oriented Requirements Engineering, AOSD 07,* Vancouver, Canadá, Março de 2007.

(Clarke e Baniassad, 2005) S. Clarke e E. Baniassad, *Aspect-Oriented Analysis and Design – The Theme Approach*, Editora Addison-Wesley, 2005.

(Cook *et* al., 2007) S. Cook, G. Jones, S. Kent, A. Wills, *Domain-specific development with visual studio dsl tools,* Editora Addison-Wesley Professional, 1ªEdição, 2007.

(Czarnecki e Helsen, 2006) K. Czarnecki e S. Helsen, *Feature-based survey of model transformation approaches,* em *IBM Systems Journal*, V. 45, Nº 3, 2006.

(Deusen *et* al., 2000) A. Deursen, P. Klint e J. Visser, *Domain-Specific Languages: An Annotated Bibliography*, *SIGPLAN Not.*, V.35, págs. 26-36, Amsterdão, Holanda, Junho de 2000.

(Dias, 2009) A. Dias, *Uma Linguagem Específica do Domínio para uma abordagem Orientada aos Objectivos baseada em KAOS*, Dissertação de Mestrado em Engenharia Informática, Departamento de Informática, Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, Quinta da Torre, Caparica, Portugal, 2009.

(Dinkelaker *et* al., 2010) T. Dinkelaker, R. Mitschke, K. Fetzer e M. Mezini, *A Dynamic Software Product Line Approach using Aspect Models at Runtime*, *AOSD '10*, Rennes e Saint Malo, França, 2010.

(Engelen, 2001) R. Engelen**,** *ATMOL: A Domain-Specific Language for Atmospheric Modeling*, Departamento de Computação Científica, *Florida State University*, EUA, 2001.

(EuGENia, 2008) EuGENia, http://www.eclipse.org/gmt/epsilon/doc/articles/eugenia-gmf-tutorial/, 2008.

(Faith *et* al., 1997) R. Faith, L. Nyland e J. Prins, *KHEPERA: A System for Rapid Implementation of Domain Specific Languages*, em *Proceedings of the Conference on Domain-Specific Languages*, Santa Bárbara, Califórnia, Outubro de 1997.

(Falbo *et* al., 2002) R. Falbo, G. Guizzardi e K. Duarte, *An Ontological Approach to Domain Engineering*, em *Proceedings of 14th Int. Conference on Software Engineering and Knowledge Engineering*, *SEKE'02*, Ischia, Itália, 2002.

(Fall e Fall, 2001) A. Fall e J. Fall, *A domain-specific language for models of landscape Dynamics, School of Resource and Environmental Management***,** Universidade de Simon Fraser, Burnaby, Canadá, Abril de 2001.

(Filman, ***et* al., 2005)** R. Filman, T. Elrad, S. Clarke, M. Aksit, *Aspect-Oriented Software Development*, Editora Addison-Wesley, Pearson Education, 2005.

(Fisher e Gruber, 2005) K. Fisher, R. Gruber, *PADS: A Domain-Specific Language for Processing Ad Hoc Data*, em *PLDI'05*, Chicago, Illinois, EUA, Junho de 2005.

(Frakes *et* al., 1998) W. Frakes, R. Prieto-Diaz e C. Fox, *DARE: Domain analysis and reuse environment*, em *Journal Annals of Software Engineering*, págs. 125-141, V. 5, Nº 1, Editora Springer, Holanda, Janeiro de 1998.

(Gabriel, 2010) P. Gabriel, *Software Languages Engineering: Experimental Evaluation*, Dissertação de Mestrado em Engenharia Informática, Departamento de Informática, Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia, Lisboa, 2010.

(Gabriel *et* al., 2010) P. Gabriel, M. Goulão e V. Amaral, *Do Software Languages Engineers Evaluate their Languages?*, em *Proceedings of the XIII Congreso Iberoamericano en Software Engineering (CIbSE'2010)*, Universidade de Azuay, ISBN-978-9978-325-10-0, Cuenca, Equador, Abril de 2010.

(Harel e Rumpe, 2000) D. Harel e B. Rumpe, *Modeling Languages: Syntax, Semantics and All That Stuff*, Agosto de 2000.

(Herndon e Berzins, 1998) R. Herndon e V. Berzins, *The realizable benefits of a language prototyping language*, em *IEEE Transactionson Software Engineering*, V.14, págs. 803-809, 1988.

(Jacobson, 1987) I. Jacobson, *Object-Oriented Development in as Industrial Environment Engineering*, em *Proceedings of OOPSLA'87*, págs. 183-191, Outubro de 1987.

(Jacobson, 1991) I. Jacobson, *Object-Oriented Software Engineering*, Editora Addisson-Wesley, 1991.

(Jacobson e NG, 2004) I. Jacobson e PW Ng, *Aspect-Oriented Software Development with Use Cases*, Editora Addison-Wesley Professional, 2004.

(Kang *et* al., **1990**) K. Kang, S. Cohen, J. Hess, W. Novak, A. Peterson, *Feature-Oriented Domain Analysis (FODA) Feasibility Study***,** Instituto de Engenharia de Software, Universidade de Cargegie Mellon, Pittsburg, Novembro de 1990.

(Kelly *et* al., 2006) S. Kelly, K. Lyytinen e M. Rossi, *MetaEdit+: A fully configurable Multi-User and Multi-Tool CASE and CAME Environment*, Departamento de Computação Científica e Sistemas de Informação, Universidade de Jyväskylä, Finlândia, 2006.

(Lamsweerde, 2001) A. Lamsweerde, *Goal-Oriented Requirements Enginnering: A Guided Tour*, Artigo Convidado para *RE'01 - 5th IEEE International Symposium on Requirements Engineering*, Toronto, Canadá, Agosto de 2001.

(Lamsweerde *et* al., 1991), A. Lamsweerde, A. Dardenne, B. Delcourt e F. Dubisy, *The KAOS Project: Knowledge Acquisition in Automated Specification of Software*, In *Proceedings AAAI Spring Symposium Series, Design of Composite Systems*, Universidade de Stanford, págs. 59-62, Março de 1991.

(Lara e Vangheluwe 2002) J. Lara e H. Vangheluwe**,** *Using AToM^3 as a Meta-CASE Tool*, Escola de Computação Científica, Universidade Autónoma de Madrid, 2002.

(Ledeczi *et* al., 2001) A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle e P. Volgyesi, *The Generic Modeling Environment,*em *Proceedings of WISP'2001*, Budapense, Hungria, Maio de 2001.

(Marot e Wuyts, 2008) A. Marot e R. Wuyts, *A DSL to declare aspect execution order*, *Workshop DSAL'08*, Bruxelas, Bélgica, Abril de 2008.

(Mernik *et* al., 2005) M. Mernik, J. Heering e A. Sloane, *When and How to Develop Domain-Specific Languages*, em *ACM Computing Surveys*, V.37, Nº4, págs. 316-344, Dezembro de 2005.

(Munnelly e Clarke, 2007) J. Munnelly, S. Clarke, *ALPH: A Domain-Specific Language for Crosscutting Pervasive Healthcare Concerns*, em *DSAL'07*, Vancouver, Canadá, Março de 2007.

(Nakatani e Jones, 1997) L. Nakatani e M. Jones, *Jargons and infocentrism*, em DSL '97 – 1º *Workshop ACM SIGPLAN* em Linguagens de Domínio Específico, em associação com *POPL '97*, Paris, França, Janeiro de 1997.

(Neighbors, 1984) J.Neighbors, *The Draco approach to constructing software from reusable components*, em *IEEE Transactions on Software Engineering*, págs. 564-74, Setembro de 1984.

(Nóbrega *et* al., 2006) L. Nóbrega, N. Nunes e H. Coelho, *The Meta Sketch Editor*, em *Proceedings CADUI*, págs. 201-204, Bucareste, Roménia, Junho de 2006.

(Nunes, 2009) C. Nunes, *Uma Linguagem de Domínio Específico para a Framework i **, Mestrado em Engenharia de Software, Departamento de Informática. (DI), Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, Quinta da Torre, Caparica, Portugal, 2009.

(Nunes *et* al., 2009) C. Nunes, J. Araújo, V. Amaral, C. Silva, *A Domain Specific Language for the I* Framework*, em *ICEIS2009*, págs. 158-163, Milão, Itália, Maio de 2009.

(Pelechano *et* al., 2006) V. Pelechano, M. Albert, J. Muñoz, C. Cetina, *Building Tools for Model Driven Development, Comparing Microsoft DSL Tools and Eclipse Modeling Plug-Ins*, Departamento de Informação, Sistemas e Computação, Universidade Técnica de Valência, Espanha, 2006.

(Rashid *et* al., 2002), A. Rashid, P. Sawyer, A. Moreira e J. Araújo, *Early Aspects: A Model for Aspect-Oriented Requirements Engineering*, em *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, págs. 199-202, Essen, Alemanha, Setembro de 2002.

(Rashid *et* al., 2003) A. Rashid, A. Moreira, e J. Araújo, *Modularization and composition of aspectual requirements*, em *Proceedings of the International Conference on Aspect-oriented Software Development*, págs. 11-20, Boston, EUA, 2003.

(Reisner, 1981) P. Reisner, *Human Factors Studies of Database Query Languages: A Survey and Assessment, Computing Surveys, ACM*, V.13, Nº 1, págs. 13-31, Março de 1981.

(Rumbaugh *et* al., 1991), J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen *Object Oriented Modeling and Design*, Editora West Prentice Hall, Nova Iorque, EUA, 1991.

(Rumbaugh *et* al., 1999) J. Rumbaugh, I. Jacobson e G. Booch**,** *The Unified Modeling Language Reference Manual*, C. 13, págs. 484-485, Editora Addison-Wesley, 1999.

(Sardinha *et* al., 2010) A. Sardinha, J. Araújo, A. Moreira, A. Rashid, *Conflict Management in Aspect-Oriented Requirements Engineering*, *Workshop* em *Early Aspects*, *AOSD 2010*, Rennes, França, Março de 2010.

(Sawyer *et* al., 1996) P. Sawyer, I. Sommerville, e S. Viller, *PREview: Tackling the Real Concerns of Requirements Engineering*, *Cooperative Systems Engineering Group*, Relatório Técnico, CSEG, Departamento de Computação da Universidade de Lancaster , Reino Unido, 1996.

(Simos, 1996) M. Simos, *Organization Domain Modeling (ODM): Domain Engineering as a Co-Methodology to Object-Oriented Techniques*, em *Fusion Newsletter*, V. 4, págs. 13-16, Laboratórios Hewlett-Packard, 1996.

(Sommerville, 2007) I. Sommerville, *Software Engineering*, Addison Wesley, 8ª Edição, 2007.

(Sommerville e Sawyer, 1997) I. Sommerville e P. Sawyer, *Viewpoints: principles, problems and a practical approach to requirements engineering*, A*nnals of Software Engineering*, págs. 101-130, 1997.

(Sommerville *et* al., 1998) I. Sommerville, P. Sawyer e S. Viller, *Viewpoints for requirements elicitation: a practical approach*, Departamento de Computação da Universidade de Lancaster, Reino Unido, 1998.

(Sousa e Sobral, 2010) E. Sousa, J. L. Sobral, *JPPAL: Java Parallel Programming Annotation Library*, *AOSD '10*, Rennes e Saint Malo, França, 2010.

(Stahl e Volter, 2006) T. Stahl e M. Volter, *Model-Driven Software Development: technology, engineering, management*, Editora *John Wiley & Sons Ltd*, Hoboken, EUA, 2006.

(Taylor *et* al., 1995) R. Taylor, W. Tracz e L. Coglianese, *Software Development Using Domain-Specific Software Architectures*, em *Software Engineering Notes*, V. 20, Nº 5, pág. 27, Dezembro de 1995.

(Thibault, 1998) S. Thibault, *DomainSpecific Languages: Conception, Implementation and Application*, Tese de Doutoramento, Universidade de Rennes, França, 1998.

(Thibault *et* al., 1997) S. Thibault, R. Marlet e C. Consel, *A Domain Specific Language for Video Device Drivers: from Design to Implementation*, em *Proceedings of the Conference on Domain-Specific Languages*, Santa Bárbara, Califórnia, EUA, Outubro de 1997.

(Thomas e Hunt, 2000) D. Thomas e A. Hunt, *Programming Ruby: the pragmatic programmer's guide*, Addison-Wesley, 2000.

(Warmer e Kleppe, 1998) J. Warmer e A. Kleppe**,** *The object constraint language: precise modeling with UML*, Editora Addison-Wesley, Boston, EUA, 1998.

(Weiss e Lay, 1999) D. Weiss e C. Lay, *Software ProductLine Engineering*, Editora Addison-Wesley, 1999.

(Whittle e Jayaraman, 2007) J. Whittle e P. Jayaraman, "*MATA: A Tool for Aspect-Oriented Modeling based on Graph Transformation*", *Workshop* em *Aspect Oriented Modeling*, *MODELS*, Nashville, TN, EUA, 2007.

(YU, 1995) E. YU, *Modelling Strategic Relationships for Business Process Reengineering*, Tese (Doutorado), Departamento de Ciências Computacionais, Universidade de Toronto, Canadá, 1995.

Anexo A. Tabelas dos Construtores da Linguagem de Composição

A Tabela A.1 contém as *Constraint Actions* da linguagem de composição. Nesta tabela pode ler-se uma descrição de cada tipo de *Constraint Action*, e os aspectos aos quais cada tipo se pode aplicar.

Tabela A.1 - Descrição das *Constraint actions*

Constraint Action		**Aspectos a que se pode aplicar**
Tipo	Descrição	
enforce	Usado para impor uma condição adicional sobre um conjunto de requisitos de um *viewpoint*.	Tempo de Resposta
ensure	Usado para afirmar que uma condição que deve existir para um conjunto de requisitos de um *viewpoint* já existe.	Disponibilidade, compatibilidade, Correccção
provide	Usado para especificar características adicionais a serem incorporadas num conjunto de requisitos de um *viewpoint*.	Segurança, Acesso Múltiplo
applied	Usado para descrever as regras que se aplicam a um conjunto de requisitos de um *viewpoint* e que podem alterar o seu resultado.	Assuntos Legais
exclude	Usado para excluir alguns *viewpoints* ou requisitos se o valor *all* foe especificado.	Qualquer um

Os elementos *Constraint Operators* podem ser observados na Tabela A.2, onde é efectuada uma descrição de cada tipo de *Constraint Operator*, são focadas as *Constraint Actions* com que estes podem ser utilizados e os aspectos com que estas combinações podem ser usadas.

Tabela A.2 - Descrição dos *Constraint Operators*

Constraint Operator		***Action***	**Aspectos Válidos: combinações *action-operator***
Tipo	Descrição		
during	Descreve o intervalo de tempo em que um conjunto de requisitos está a ser satisfeito.	*ensure*	Disponibilidade: *ensure-during*
between	Descreve o intervalo de tempo compreendido entre a satisfação de dois requisitos. O intervalo começa quando o primeiro requisito é satisfeito, e termina quando o segundo está a começar a ser satisfeito.	*enforce*	Tempo de Resposta: *enforce-between*
on	Descreve o ponto no tempo depois de um conjunto de requisitos ter sido satisfeito.	*enforce*	Tempo de Resposta: *enforce-on*
for	Descreve que características adicionais vão complementar os requisitos do *viewpoint*.	*applied, provide*	Assuntos Legais: *applied-for* Segurança: *provide-for* Acesso Múltiplo: *provide-for*
with	Descreve que uma condição vai ser suportada por dois conjuntos de requisitos em relação um ao outro.	*ensure*	Compatibilidade: *ensure-with*
in	Descreve que uma condição vai ser assegurada por um conjunto de requisitos que foram satisfeitos.	*ensure*	Correcção: *ensure-in*
XOR	Ou-Exclusivo (quando cada requisito é satisfeito, mas não ambos).	Qualquer um	Qualquer um

A Tabela A.3 contém a descrição das Outcome Actions e os aspectos a que se podem aplicar.

Tabela A.3. Descrição das *Outcome Actions*.

Outcome Action		**Aspectos a que se pode aplicar**
Tipo	Descrição	
satisfied	Usado para afirmar que um conjunto de requisitos de um *viewpoint* vão ser satisfeitas após serem aplicadas as restrições do requisito aspectual.	Qualquer um
fulfilled	Usado para afirmar que as restrições de um requisito aspectual foram aplicadas com êxito.	Qualquer um

Anexo B. Modelo Emfatic do editor base

```
@gmf(foo="bar")
@namespace(uri="AORE", prefix="AORE")
package aore;

@gmf.diagram(foo="bar")
class AORE {
  val Weighs[*] hasWeight;
  val HasImpact[*] hasImpact;
  val Viewpoint[*] hasViewpoint;
  val Concern[*] hasRNFConcern;
  val CompositionRule[*] hasCompositeAggRule;
  val ConcernsAggregationModule[+] hasConcernsAggregationModule;
  val ViewpointsAggregationModule[+] hasViewpointsAggregationModule;
  val CompositionRulesModule[+] hasCompositionRulesAggregationModule;
}

@gmf.node(figure="rounded", label="name", border.width="3", border.color="187,0,0",
color="255,206,206", tool.name="Viewpoints Aggregation Module",
tool.description="Create a new Viewpoints Aggregation Module node.")
class ViewpointsAggregationModule {
  attr String name;
  attr String description;

  @gmf.compartment(foo="bar")
  val Viewpoint[+] hasVP;

  @gmf.compartment(foo="bar")
  val ViewpointsAggregationModule[*] hasVM;
}

@gmf.node(label="name", label.pattern="{0}", color="240,240,240",
border.color="215,0,0", border.width="3", tool.name="Viewpoint",
tool.description="Create a new Viewpoint node.")
class Viewpoint {

  @gmf.compartment(layout="list")
  val FRequirement[+] contains;
  attr int Id = 1;
  attr String name;
  attr String focus;
  attr String source;

  @gmf.compartment(foo="bar")
  val FRAggregationModule[*] hasCM;
}

@gmf.node(figure="rectangle", label="Id, description", label.pattern="{0}. {1}",
label.icon="false", border.color="240,240,240", color="240,240,240",
tool.name="Functional Requirement", tool.description="Create a new Funcional
Requirement node.")
class FRequirement {
  attr String Id = "1";
  attr String description;
}

@gmf.node(figure="rounded", label="Id, name", border.width="2", label.pattern="{0}.
{1}", color="240,240,240", border.color="0,0,160", tool.name="Functional
Requirements Aggregation Module", tool.description="Create a new Funcional
Requirements Aggregation Module node.")
class FRAggregationModule {
  attr int Id = 1;
  attr String name;
  attr String description;

  @gmf.compartment(layout="list")
```

```
  val FRequirement[+] containsRF;
}

@gmf.node(figure="rounded", label="name", border.width="3",
border.color="14,105,3", color="205,250,190", tool.name="Concerns Aggregation
Module", tool.description="Create a new Concerns Aggregation Module node .")
class ConcernsAggregationModule {
  attr String name;
  attr String description;

  @gmf.compartment(foo="bar")
  val Concern[+] hasConc;

  @gmf.compartment(foo="bar")
  val ConcernsAggregationModule[*] hasCM;
  val Contributes[*] hasC;
}

@gmf.node(label="name", label.pattern="{0}", color="240,240,240", border.width="3",
border.color="57,183,15", tool.name="Concern", tool.description="Create a new
Concern node.")
class Concern {

  @gmf.compartment(layout="list")
  val NFRequirement[+] CcontainsNFR;
  attr String name;
  attr int Id = 1;
}

@gmf.node(figure="rectangle", label="Id, description", label.pattern="{0}. {1}",
label.icon="false", border.color="240,240,240", color="240,240,240",
tool.name="Non-Functional Requirement", tool.description="Create a new Non
Funcional Requirement node.")
class NFRequirement {
  attr String Id = "1";
  attr String description;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/NFRequirementTOCAction.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/NFRequirementTOCAction.gif", style="dot",
width="2", target.decoration="arrow", color="0,140,0", tool.name="NFRequirement-
>CAction", tool.description="Create a new link from a Non Functional Requirement to
a Constraint Action")
  ref ConstraintAction[*]#NFReqToCActionOp NFRequirementTOCAction;
}

@gmf.node(color="240,240,240", border.color="0,140,0", tool.name="Aspect",
tool.description="Create a new Aspect node.")
class Aspect extends Concern {
  ref Viewpoint[2..*] crosscutsVP;
}

enum Type {
  PosPos = 0;
  NegNeg = 1;
  PosNeg = 2;
  NegPos = 3;
  PosNull = 4;
  NullPos = 5;
  NegNull = 6;
  NullNeg = 7;
}

@gmf.link(label="type", source="in", target="out", style="solid", width="1",
color="50,50,50", source.decoration="filledclosedarrow",
target.decoration="filledclosedarrow", tool.name="Contribution",
tool.description="Create a new Contribution link.")
class Contributes {
  attr Type type;
```

```
  ref Concern[1] in;
  ref Concern[1] out;
}

@gmf.node(figure="rounded", label="name", border.width="3",
border.color="111,0,221", color="234,213,255", tool.name="Composition Rules
Aggregation Module", tool.description="Create a new Composition Rules Aggregation
Module node.")
class CompositionRulesModule {
  attr String name;
  attr String description;

  @gmf.compartment(foo="bar")
  val CompositionRule[+] hasCR;

  @gmf.compartment(foo="bar")
  val CompositionRulesModule[*] hasCRM;
}

@gmf.node(figure="rounded", label="Id, name", label.pattern="{0}. {1}",
color="240,240,240", border.color="128,128,192", border.width="3",
tool.name="Composition Rule", tool.description="Create a new Composition Rule
node.")
class CompositionRule {
  attr int Id = 1;
  attr String name;
  attr String description;

  @gmf.compartment(foo="bar")
  val Constraint[1] hasConstraint;

  @gmf.compartment(foo="bar")
  val OutcomeAction[1] hasOutcAct;

  @gmf.compartment(foo="bar")
  val ConstraintOperator[+] hasCO;
}

@gmf.node(figure="rounded", label="name", color="240,240,240",
border.color="64,0,128", border.width="2", tool.name="Constraint",
tool.description="Create a new Constraint node.")
class Constraint {
  attr String name;

  @gmf.compartment(foo="bar")
  val ConstraintAction[1] hasConstAction;

  @gmf.compartment(foo="bar")
  val ConstraintOperator[1] hasConsOper;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/ConstraintTOOAction.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/ConstraintTOOAction.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Constraint-
>OAction", tool.description="Create a new link from a Constraint node to an Outcome
Action node")
  ref OutcomeAction[1] ConstraintTOOAction;
}

@gmf.node(label="name", label.icon="false", label.placement="internal")
abstract class OutcomeAction {
  attr String name;
}

@gmf.node(figure="figures.FulfilledFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Fulfilled Outcome Action
node.")
class Fulfilled extends OutcomeAction {
}
```

```
@gmf.node(figure="figures.SatisfiedFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Satisfied Outcome Action
node.")
class Satisfied extends OutcomeAction {

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/OActionTOViewpoint.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/OActionTOViewpoint.gif", style="solid",
width="2", target.decoration="arrow", color="215,0,0", tool.name="OAction-
>Viewpoint", tool.description="Create a new link from an Outcome Action to a
Viewpoint.")
  ref Viewpoint[*] OActionTOViewpoint;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/OActionTOFRAggModule.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/OActionTOFRAggModule.gif", style="solid",
width="2", target.decoration="arrow", color="215,0,0", tool.name="OAction-
>FRAggModule", tool.description="Create a new link from an Outcome Action to a
Functional Requirements Aggregation Module Node.")
  ref FRAggregationModule[*] OActionTOFRModule;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/OActionTOFRequirement.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/OActionTOFRequirement.gif", style="solid",
width="2", target.decoration="arrow", color="215,0,0", tool.name="OAction-
>FRequirement", tool.description="Create a new link from an Outcome Action to
Functional Requirement.")
  ref FRequirement[*] OActionTOFRequirement;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/SatisfiedTOCOperator.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/SatisfiedTOCOperator.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Satisfied-
>COperator", tool.description="Create a new link from a Constraint node to an
Outcome Action node")
  ref ConstraintOperator[*] SatisfiedTOCOperator;
}
abstract class ConstraintAction {

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/CActionTOXor.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/CActionTOXor.gif", style="solid", width="1",
target.decoration="arrow", color="0,0,0", tool.name="CAction->Xor",
tool.description="Create a new link from a Constraint Action node to the Constraint
Operator Xor")
  ref Xor[1] CActionTOXor;
  attr String name;
  ref NFRequirement#NFRequirementTOCAction NFReqToCActionOp;
}

abstract class ConstraintOperator {
  attr String name;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/COperatorTOFRequirement.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/COperatorTOFRequirement.gif", style="dash",
width="2", target.decoration="arrow", color="215,0,0", tool.name="COperator-
>FRequirement", tool.description="Create a new link from a Constraint Operator node
to a Functional Requirement node.")
  ref FRequirement[*] COperatorTOFRequirement;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/COperatorTOFRAggModule.gif",
tool.large.bundle="Aore.edit",
```

```
tool.large.path="/icons/full/obj16/COperatorTOFRAggModule.gif", style="dash",
width="2", target.decoration="arrow", color="215,0,0", tool.name="COperator-
>FRModule", tool.description="Create a new link from a Constraint Operator node to
a Functional Requirements Aggregation Module node.")
  ref FRAggregationModule[*] COperatorTOFRModule;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/COperatorTOViewpoint.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/COperatorTOViewpoint.gif", style="dash",
width="2", target.decoration="arrow", color="215,0,0", tool.name="COperator-
>Viewpoint", tool.description="Create a new link from a Constraint Operator node to
a Viewpoint node.")
  ref Viewpoint[*] COperatorTOViewpoint;
}

abstract class AppliedProvide extends ConstraintAction {

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/AppliedORProvideTOFor.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/AppliedORProvideTOFor.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Applied|Provide-
>For", tool.description="Create a new link from the Constraint Action Applied or
Provide node to the Constraint Operator For")
  ref For[1]#ForTOAppliedProvide AppliedProvideTOFor;
}

abstract class OnBetween extends ConstraintOperator {
  ref Enforce[1]#EnforceTOOnBetween OnBetweenTOEnforce;
}

abstract class DuringWithIn extends ConstraintOperator {
  ref Ensure[1]#EnsureTODuringWithIn DuringWithInTOEnsure;
}

@gmf.node(figure="figures.AppliedFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Applied Constraint
Action node.")
class Applied extends AppliedProvide {
}

@gmf.node(figure="figures.EnforceFigure", label="name", label.icon="false",
label.placement="exinternal", tool.description="Create a new Enforce Constraint
Action node.")
class Enforce extends ConstraintAction {

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/EnforceToOnOrBetween.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/EnforceToOnOrBetween.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Enforce-
>On|Between", tool.description="Create a new link from the Constraint Action
Enforce node to the Constraint Operator node On or Between.")
  ref OnBetween[1]#OnBetweenTOEnforce EnforceTOOnBetween;
}

@gmf.node(figure="figures.EnsureFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Ensure Constraint Action
node.")
class Ensure extends ConstraintAction {

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/EnsureTODuringORWithORIn.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/EnsureTODuringORWithORIn.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Ensure-
>During|With|In", tool.description="Create a new link from the Constraint Action
Ensure node to the Constraint Operator node During, With or In.")
  ref DuringWithIn[1]#DuringWithInTOEnsure EnsureTODuringWithIn;
```

```
}

@gmf.node(figure="figures.ExcludeFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Exclude Constraint
Action node.")
class Exclude extends ConstraintAction {

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/ExcludeToCOperator.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/ExcludeToCOperator.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Exclude-
>COperator", tool.description="Create a new link from the Constraint Action Exclude
node to any Constraint Operator node.")
  ref ConstraintOperator[1] ExcludeToCOperator;
}

@gmf.node(figure="figures.ProvideFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Provide Constraint
Action node.")
class Provide extends AppliedProvide {
}

@gmf.node(figure="figures.BetweenFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Between Constraint
Operator node.")
class Between extends OnBetween {
}

@gmf.node(figure="figures.DuringFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new During Constraint
Operator node.")
class During extends DuringWithIn {
}

@gmf.node(figure="figures.ForFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new For Constraint Operator
node.")
class For extends ConstraintOperator {
  ref AppliedProvide#AppliedProvideTOFor ForTOAppliedProvide;
}

@gmf.node(figure="figures.InFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new In Constraint Operator
node.")
class In extends DuringWithIn {
}

@gmf.node(figure="figures.OnFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new On Constraint Operator
node.")
class On extends OnBetween {
}

@gmf.node(figure="figures.WithFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new With Constraint Operator
node.")
class With extends DuringWithIn {
}

@gmf.node(figure="figures.XorFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Xor Constraint Operator
node.")
class Xor extends ConstraintOperator {
}

enum Weight {
  Zero = 0;
  ZeroUm = 1;
  ZeroDois = 2;
```

```
  ZeroTres = 3;
  ZeroQuatro = 4;
  ZeroCinco = 5;
  ZeroSeis = 6;
  ZeroSete = 7;
  ZeroOito = 8;
  ZeroNove = 9;
  Um = 10;
}

@gmf.link(label="weight", source="inW", target="outW", style="dash",
color="100,100,100", width="2", target.decoration="filledclosedarrow",
tool.name="Weigh", tool.description="Create a new Weight link.")
class Weighs {
  attr Weight weight = "0";
  ref Viewpoint[1] outW;
  ref Concern[1] inW;
}

@gmf.link(label="name", source="inH", target="outH", tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/HasImpact.gif", tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/HasImpact.gif", width="2", style="dot",
target.decoration="filledclosedarrow", color="0,0,0", tool.name="Has Impact In",
tool.description="Create a new link from a Concern node to a Viewpoint node.")
class HasImpact {
  attr String name;
  ref Viewpoint[1] outH;
  ref Concern[1] inH;
}
```

Anexo C. Modelo Emfatic do sub-editor do elemento *ConcernsAggregationModule*

```
@gmf(foo="bar")
@namespace(uri="AORE", prefix="AORE")
package aore;

class AORE {
  val Weighs[*] hasWeight;
  val HasImpact[*] hasImpact;
  val Viewpoint[*] hasViewpoint;
  val Concern[*] hasRNFConcern;
  val CompositionRule[*] hasCompositeRule;
  val ConcernsAggregationModule[+] hasConcernsModule;
  val ViewpointsModule[+] hasViewpointsModule;
  val CompositionRulesModule[+] hasCompositionRulesModule;
}

class ViewpointsModule {
  attr String name;
  val Viewpoint[+] hasVP;
  val ViewpointsModule[*] hasVM;
}

class Viewpoint {
  val FRequirement[+] contains;
  attr int Id = 1;
  attr String name;
  attr String focus;
  attr String source;
  val FRModule[*] hasCM;
}

class FRequirement {
  attr String Id = "1";
  attr String description;
}

class FRModule {
  attr int Id = 1;
  attr String name;
  val FRequirement[+] containsRF;
}
```

```
@gmf.diagram(foo="bar")
class ConcernsAggregationModule {
  attr String name;
  val Concern[+] hasConc;
  val ConcernsAggregationModule[*] hasCAggM;
  val Contributes[*] hasC;
}

@gmf.node(label="name", label.pattern="{0}", color="240,240,240", border.width="3",
border.color="57,183,15", tool.name="Concern", tool.description="Create a new
Concern node.")
class Concern {

  @gmf.compartment(layout="list")
  val NFRequirement[+] CcontainsNFR;
  attr String name;
  attr int Id = 1;
 }

@gmf.node(figure="rectangle", label="Id, description", label.pattern="{0}. {1}",
label.icon="false", border.color="240,240,240", color="240,240,240",
tool.name="Non-Functional Requirement", tool.description="Create a new Non
Funcional Requirement node.")
class NFRequirement {
```

```
  attr String Id = "1";
  attr String description;
  ref ConstraintAction[*]#NFReqToCActionOp NFRequirementTOCAction;
}

@gmf.node(color="240,240,240", border.color="0,140,0", tool.name="Aspect",
tool.description="Create a new Aspect node.")
class Aspect extends Concern {
  ref Viewpoint[2..*] crosscutsVP;
}

enum Type {
  PosPos = 0;
  NegNeg = 1;
  PosNeg = 2;
  NegPos = 3;
  NegNull = 4;
  NullNeg = 5;
  PosNull = 6;
  NullPos = 7;
}

@gmf.link(label="type", source="in", target="out", style="solid", width="1",
color="50,50,50", source.decoration="filledclosedarrow",
target.decoration="filledclosedarrow", tool.name="Contribution",
tool.description="Create a new Contribution link.")
class Contributes {
  attr Type type;
  ref Concern[1] in;
  ref Concern[1] out;
}

class CompositionRulesModule {
  attr String name;
  val CompositionRule[+] hasCR;
  val CompositionRulesModule[*] hasCRM;

}

class CompositionRule {
  attr int Id = 1;
  attr String name;
  val Constraint[1] hasConstraint;
  val OutcomeAction[1] hasOutcAct;
  val ConstraintOperator[*] hasCO;
}

class Constraint {
  attr String name;
  val ConstraintAction[1] hasConstAction;
  val ConstraintOperator[1] hasConsOper;
  ref OutcomeAction[1] ConstraintTOOAction;
}

abstract class OutcomeAction {
  attr String name;
}

class Fulfilled extends OutcomeAction {
}

class Satisfied extends OutcomeAction {
  ref Viewpoint[1] OActionTOViewpoint;
  ref FRModule OActionTOFRModule;
  ref FRequirement OActionTOFRequirement;
  ref ConstraintOperator[*] SatisfiedTOCOperator;
}

abstract class ConstraintAction {
  ref Xor[1] CActionTOXor;
```

```
  attr String name;
  ref Concern#ConcernTOCAction ConcernToCActionOP;
  ref NFRequirement#NFRequirementTOCAction NFReqToCActionOp;
}

abstract class ConstraintOperator {
  attr String name;
  ref FRequirement COperatorTOFRequirement;
  ref FRModule COperatorTOFRAggModule;
  ref Viewpoint COperatorTOViewpoint;
}

abstract class AppliedProvide extends ConstraintAction {
  ref For[1]#ForTOAppliedProvide AppliedProvideTOFor;
}

abstract class OnBetween extends ConstraintOperator {
  ref Enforce[1]#EnforceTOOnBetween OnBetweenTOEnforce;
}

abstract class DuringWithIn extends ConstraintOperator {
  ref Ensure[1]#EnsureTODuringWithIn DuringWithInTOEnsure;
}

class Applied extends AppliedProvide {
}

class Enforce extends ConstraintAction {
  ref OnBetween[1]#OnBetweenTOEnforce EnforceTOOnBetween;
}

class Ensure extends ConstraintAction {
  ref DuringWithIn[1]#DuringWithInTOEnsure EnsureTODuringWithIn;
}

class Exclude extends ConstraintAction {
    ref ConstraintOperator[1] ExcludeToCOperator;
}

class Provide extends AppliedProvide {}
class Between extends OnBetween {
}

class During extends DuringWithIn {
}

class For extends ConstraintOperator {
  ref AppliedProvide#AppliedProvideTOFor ForTOAppliedProvide;
}

class In extends DuringWithIn {
}

class On extends OnBetween {
}

class With extends DuringWithIn {
}

class Xor extends ConstraintOperator {
}

enum Weight {
  Zero = 0;
  ZeroUm = 1;
  ZeroDois = 2;
  ZeroTres = 3;
  ZeroQuatro = 4;
  ZeroCinco = 5;
  ZeroSeis = 6;
```

```
  ZeroSete = 7;
  ZeroOito = 8;
  ZeroNove = 9;
  Um = 10;
}

class Weighs {
  attr Weight weight = "0";
  ref Viewpoint[1] outW;
  ref Concern[1] inW;
}

class HasImpact {
  attr String name;
  ref Viewpoint[1] outH;
  ref Concern[1] inH;
}
```

Anexo D. Modelo Emfatic do sub-editor do elemento *CompositioRulesAggregationModule*:

```
@gmf(foo="bar")
@namespace(uri="AORE", prefix="AORE")
package aore;

class AORE {

  val Weighs[*] hasWeight;
  val HasImpact[*] hasImpact;
  val Viewpoint[*] hasViewpoint;
  val Concern[*] hasRNFConcern;
  val CompositionRule[*] hasCompositeRule;
  val ConcernsModule[+] hasConcernsModule;
  val ViewpointsModule[+] hasViewpointsModule;
  val CompositionRulesAggregationModule[+] hasCompositionRulesModule;
}

class ViewpointsModule {
  attr String name;
  val Viewpoint[+] hasVP;
  val ViewpointsModule[*] hasVM;
}

class Viewpoint {
  val FRequirement[+] contains;
  attr int Id = 1;
  attr String name;
  attr String focus;
  attr String source;
  val FRModule[*] hasCM;
}

class FRequirement {
  attr String Id = "1";
  attr String description;
}

class FRModule {
  attr int Id = 1;
  attr String name;
  val FRequirement[+] containsRF;
}

class ConcernsModule {
  attr String name;
  val Concern[+] hasConc;
  val ConcernsModule[*] hasCM;
  val Contributes[*] hasC;
}

class Concern {
  val NFRequirement[+] CcontainsNFR;
  attr String name;
  attr int Id = 1;

}
class NFRequirement {
  attr String Id = "1";
  attr String description;
  ref ConstraintAction[*]#NFReqToCActionOp NFRequirementTOCAction;
}

class Aspect extends Concern {
  ref Viewpoint[2..*] crosscutsVP;
}
```

```
enum Type {
  PosPos = 0;
  NegNeg = 1;
  PosNeg = 2;
  NegPos = 3;
}

class Contributes {
  attr Type type;
  ref Concern[1] in;
  ref Concern[1] out;
}
```

```
@gmf.diagram(foo="bar")
class CompositionRulesAggregationModule {
  attr String name;
  attr String description;
  val CompositionRule[+] hasCR;
  val CompositionRulesAggregationModule[*] hasCRM;
}

@gmf.node(figure="rounded", label="Id, name", label.pattern="{0}. {1}",
color="240,240,240", border.color="128,128,192", border.width="3",
tool.name="Composition Rule", tool.description="Create a new Composition Rule
node.")
class CompositionRule {
  attr int Id = 1;
  attr String name;
  attr String description;

  @gmf.compartment(foo="bar")
  val Constraint[1] hasConstraint;

  @gmf.compartment(foo="bar")
  val OutcomeAction[1] hasOutcAct;
   @gmf.compartment(foo="bar")
  val ConstraintOperator[+] hasCO;
}

@gmf.node(figure="rounded", label="name", color="240,240,240",
border.color="64,0,128", border.width="2", tool.name="Constraint",
tool.description="Create a new Constraint node.")
class Constraint {
  attr String name;

  @gmf.compartment(foo="bar")
  val ConstraintAction[1] hasConstAction;

  @gmf.compartment(foo="bar")
  val ConstraintOperator[1] hasConsOper;

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/ConstraintTOOAction.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/ConstraintTOOAction.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Constraint-
>OAction", tool.description="Create a new link from a Constraint node to an Outcome
Action node")
  ref OutcomeAction[1] ConstraintTOOAction;
}

abstract class OutcomeAction {
  attr String name;
}

@gmf.node(figure="figures.FulfilledFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Fulfilled Outcome Action
node.")
class Fulfilled extends OutcomeAction {
}
```

```
@gmf.node(figure="figures.SatisfiedFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Satisfied Outcome Action
node.")
class Satisfied extends OutcomeAction {
  ref Viewpoint[*] OActionTOViewpoint;
  ref FRModule[*] OActionTOFRAggModule;
  ref FRequirement[*] OActionTOFRequirement;
  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/SatisfiedTOCOperator.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/SatisfiedTOCOperator.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Satisfied-
>COperator", tool.description="Create a new link from a Constraint node to an
Outcome Action node")
  ref ConstraintOperator[*] SatisfiedTOCOperator;
}

abstract class ConstraintAction {

  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/CActionTOXor.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/CActionTOXor.gif", style="solid", width="1",
target.decoration="arrow", color="0,0,0", tool.name="CAction->Xor",
tool.description="Create a new link from a Constraint Action node to the Constraint
Operator Xor")
  ref Xor[1] CActionTOXor;
  attr String name;
  ref NFRequirement#NFRequirementTOCAction NFReqToCActionOp;
}
abstract class ConstraintOperator {
  attr String name;
  ref FRequirement[*] COperatorTOFRequirement;
  ref FRModule[*] COperatorTOFRAggModule;

  ref Viewpoint[*] COperatorTOViewpoint;
}
abstract class AppliedProvide extends ConstraintAction {
  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/AppliedORProvideTOFor.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/AppliedORProvideTOFor.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Applied|Provide-
>For", tool.description="Create a new link from the Constraint Action Applied or
Provide node to the Constraint Operator For")
  ref For[1]#ForTOAppliedProvide AppliedProvideTOFor;
}

abstract class OnBetween extends ConstraintOperator {
  ref Enforce[1]#EnforceTOOnBetween OnBetweenTOEnforce;
}

abstract class DuringWithIn extends ConstraintOperator {
  ref Ensure[1]#EnsureTODuringWithIn DuringWithInTOEnsure;
}

@gmf.node(figure="figures.AppliedFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Applied Constraint
Action node.")
class Applied extends AppliedProvide {
}

@gmf.node(figure="figures.EnforceFigure", label="name", label.icon="false",
label.placement="exinternal", tool.description="Create a new Enforce Constraint
Action node.")
class Enforce extends ConstraintAction {
  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/EnforceToOnOrBetween.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/EnforceToOnOrBetween.gif", style="solid",
```

```
width="1", target.decoration="arrow", color="0,0,0", tool.name="Enforce-
>On|Between", tool.description="Create a new link from the Constraint Action
Enforce node to the Constraint Operator node On or Between.")
  ref OnBetween[1]#OnBetweenTOEnforce EnforceTOOnBetween;
}

@gmf.node(figure="figures.EnsureFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Ensure Constraint Action
node.")
class Ensure extends ConstraintAction {
  @gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/EnsureTODuringORWithORIn.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/EnsureTODuringORWithORIn.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Ensure-
>During|With|In", tool.description="Create a new link from the Constraint Action
Ensure node to the Constraint Operator node During, With or In.")
  ref DuringWithIn[1]#DuringWithInTOEnsure EnsureTODuringWithIn;
}

@gmf.node(figure="figures.ExcludeFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Exclude Constraint
Action node.")
class Exclude extends ConstraintAction {
@gmf.link(tool.small.bundle="Aore.edit",
tool.small.path="/icons/full/obj16/ExcludeToCOperator.gif",
tool.large.bundle="Aore.edit",
tool.large.path="/icons/full/obj16/ExcludeToCOperator.gif", style="solid",
width="1", target.decoration="arrow", color="0,0,0", tool.name="Exclude-
>COperator", tool.description="Create a new link from the Constraint Action Exclude
node to any Constraint Operator node.")
  ref ConstraintOperator[1] ExcludeToCOperator;
}

@gmf.node(figure="figures.ProvideFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Provide Constraint
Action node.")
class Provide extends AppliedProvide {
}

@gmf.node(figure="figures.BetweenFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Between Constraint
Operator node.")
class Between extends OnBetween {
}

@gmf.node(figure="figures.DuringFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new During Constraint
Operator node.")
class During extends DuringWithIn {
}

@gmf.node(figure="figures.ForFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new For Constraint Operator
node.")
class For extends ConstraintOperator {
  ref AppliedProvide#AppliedProvideTOFor ForTOAppliedProvide;
}

@gmf.node(figure="figures.InFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new In Constraint Operator
node.")
class In extends DuringWithIn {
}

@gmf.node(figure="figures.OnFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new On Constraint Operator
node.")
class On extends OnBetween {
}
```

```
@gmf.node(figure="figures.WithFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new With Constraint Operator
node.")
class With extends DuringWithIn {
}

@gmf.node(figure="figures.XorFigure", label="name", label.icon="false",
label.placement="external", tool.description="Create a new Xor Constraint Operator
node.")
class Xor extends ConstraintOperator {
}
```

```
enum Weight {
  Zero = 0;
  ZeroUm = 1;
  ZeroDois = 2;
  ZeroTres = 3;
  ZeroQuatro = 4;
  ZeroCinco = 5;
  ZeroSeis = 6;
  ZeroSete = 7;
  ZeroOito = 8;
  ZeroNove = 9;
  Um = 10;
}

class Weighs {
  attr Weight weight = "0";
  ref Viewpoint[1] outW;
  ref Concern[1] inW;
}

class HasImpact {
  attr String name;
  ref Viewpoint[1] outH;
  ref Concern[1] inH;
}
```

Anexo E. Modelo Emfatic do sub-editor do elemento *ViewpointsAggregationModule*

```
@gmf(foo="bar")
@namespace(uri="AORE", prefix="AORE")
package aore;

class AORE {
  val Weighs[*] hasWeight;
  val HasImpact[*] hasImpact;
  val Viewpoint[*] hasViewpoint;
  val Concern[*] hasRNFConcern;
  val CompositionRule[*] hasCompositeRule;
  val ConcernsModule[+] hasConcernsModule;
  val ViewpointsAggregationModule[+] hasViewpointsModule;
  val CompositionRulesAggregationModule[+] hasCompositionRulesModule;
}
```

```
@gmf.diagram(foo="bar")
class ViewpointsAggregationModule {
  attr String name;
  attr String description;
  val Viewpoint[+] hasVP;
  val ViewpointsAggregationModule[*] hasVM;
}

@gmf.node(label="name", label.pattern="{0}", color="240,240,240",
border.color="215,0,0", border.width="3", tool.name="Viewpoint",
tool.description="Create a new Viewpoint node.")
class Viewpoint {

  @gmf.compartment(layout="list")
  val FRequirement[+] contains;
  attr int Id = 1;
  attr String name;
  attr String focus;
  attr String source;

  @gmf.compartment(foo="bar")
  val FRAggregationModule[*] hasCM;
}

@gmf.node(figure="rectangle", label="Id, description", label.pattern="{0}. {1}",
label.icon="false", border.color="240,240,240", color="240,240,240",
tool.name="Functional Requirement", tool.description="Create a new Funcional
Requirement node.")
class FRequirement {
  attr String Id = "1";
  attr String description;
}

@gmf.node(figure="rounded", label="Id, name", border.width="2", label.pattern="{0}.
{1}", color="240,240,240", border.color="0,0,160", tool.name="Functional
Requirements Aggregation Module", tool.description="Create a new Funcional
Requirements Aggregation Module node.")
class FRAggregationModule {
  attr int Id = 1;
  attr String name;
  attr String description;

  @gmf.compartment(layout="list")
  val FRequirement[+] containsRF;
}
```

```
class ConcernsModule {
  attr String name;
  val Concern[+] hasConc;
  val ConcernsModule[*] hasCM;
```

```
  val Contributes[*] hasC;
}

class Concern {
  val NFRequirement[+] CcontainsNFR;
  attr String name;
  attr int Id = 1;
   }

class NFRequirement {
  attr String Id = "1";
  attr String description;
  ref ConstraintAction[*]#NFReqToCActionOp NFRequirementTOCAction;
}

class Aspect extends Concern {
  ref Viewpoint[2..*] crosscutsVP;
}

enum Type {
  PosPos = 0;
  NegNeg = 1;
  PosNeg = 2;
  NegPos = 3;
}

class Contributes {
  attr Type type;
  ref Concern[1] in;
  ref Concern[1] out;
}

class CompositionRulesAggregationModule {
  attr String name;
  val CompositionRule[+] hasCR;
  val CompositionRulesAggregationModule[*] hasCRM;
}

class CompositionRule {
  attr int Id = 1;
  attr String name;
  val Constraint[1] hasConstraint;
  val OutcomeAction[1] hasOutcAct;
  val ConstraintOperator[*] hasCO;
}

class Constraint {
  attr String name;
  val ConstraintAction[1] hasConstAction;
  val ConstraintOperator[1] hasConsOper;
  ref OutcomeAction[1] ConstraintTOOAction;
}

abstract class OutcomeAction {
  attr String name;}

class Fulfilled extends OutcomeAction {
}

class Satisfied extends OutcomeAction {
  ref Viewpoint[1] OActionTOViewpoint;
  ref FRAggregationModule OActionTOFRModule;
  ref FRequirement OActionTOFRequirement;
  ref ConstraintOperator[*] SatisfiedTOCOperator;
}

abstract class ConstraintAction {
  ref Xor[1] CActionTOXor;
   attr String name;
  ref Concern#ConcernTOCAction ConcernToCActionOP;
```

```
    ref NFRequirement#NFRequirementTOCAction NFReqToCActionOp;
}

abstract class ConstraintOperator {
  attr String name;
  ref FRequirement COperatorTOFRequirement;
  ref FRAggregationModule COperatorTOFRModule;
  ref Viewpoint COperatorTOViewpoint;
}

abstract class AppliedProvide extends ConstraintAction {
  ref For[1]#ForTOAppliedProvide AppliedProvideTOFor;
}

abstract class OnBetween extends ConstraintOperator {
  ref Enforce[1]#EnforceTOOnBetween OnBetweenTOEnforce;
}

abstract class DuringWithIn extends ConstraintOperator {
  ref Ensure[1]#EnsureTODuringWithIn DuringWithInTOEnsure;
}

class Applied extends AppliedProvide {
}

class Enforce extends ConstraintAction {
  ref OnBetween[1]#OnBetweenTOEnforce EnforceTOOnBetween;
}

class Ensure extends ConstraintAction {
  ref DuringWithIn[1]#DuringWithInTOEnsure EnsureTODuringWithIn;
}

class Exclude extends ConstraintAction {
  ref ConstraintOperator[1] ExcludeToCOperator;
}

class Provide extends AppliedProvide {
}

class Between extends OnBetween {
}

class During extends DuringWithIn {
}

class For extends ConstraintOperator {
  ref AppliedProvide#AppliedProvideTOFor ForTOAppliedProvide;
}

class In extends DuringWithIn {
}

class On extends OnBetween {
}

class With extends DuringWithIn {
}

class Xor extends ConstraintOperator {
}

enum Weight {
  Zero = 0;
  ZeroUm = 1;
  ZeroDois = 2;
  ZeroTres = 3;
  ZeroQuatro = 4;
  ZeroCinco = 5;
  ZeroSeis = 6;
```

```
  ZeroSete = 7;
  ZeroOito = 8;
  ZeroNove = 9;
  Um = 10;
}

class Weighs {
  attr Weight weight = "0";
  ref Viewpoint[1] outW;
  ref Concern[1] inW;
}

class HasImpact {
  attr String name;
  ref Viewpoint[1] outH;
  ref Concern[1] inH
```

Anexo F Especificação Textual

Viewpoints

Habitante

1. O administrador escolhe uma opção de configuração:
2. Se o habitante escolhe a opção de configurar o AC:
    2.1. Se pretende configurar o sistema de energia:
        2.1.1. Insere data e hora a que pretende ligar o AC, assim como as divisões da casa em que deve ser ligado;
    2.2. Se pretende configurar o sistema de climatização:
        2.2.1. Activa a opção de regulação pela temperatura exterior e insere as divisões da casa em que deve ser ligado ou define a temperatura mínima e máxima e as divisões pretendidas;
3. Se o habitante escolhe a opção de ajustar as janelas:
    3.1. Se pretende configurar o sistema de fumo e fogo:
        3.1.1. Insere as divisões da casa em que deve ser ligado e activa a abertura de janelas em caso de fumo/fogo;
    3.2. Se pretende configurar o sistema de climatização:
        3.2.1. Activa a opção de regulação pela temperatura exterior e insere as divisões da casa em que deve ser ligado ou define a temperatura mínima e máxima e as divisões pretendidas;
4. Se o habitante escolhe a opção de ajustar os estores:
    4.1. Se pretende configurar o sistema de fumo e fogo:
        4.1.1. Insere as divisões da casa em que deve ser ligado e activa a abertura de janelas em caso de fumo/fogo;
    4.2. Se pretende configurar o sistema de climatização:
        4.2.1. Activa a opção de regulação pela temperatura exterior e insere as divisões da casa em que deve ser ligado ou define a temperatura mínima e máxima e as divisões pretendidas;
5. Se o habitante escolhe a opção de ajustar as luzes:
    5.2. Se pretende configurar o sistema de iluminação:
        5.2.1. Insere a data e hora em que ligam e desligam as luzes, assim como as divisões da casa onde se deseja esse efeito;
    5.3. Se pretende configurar o sistema de simulação de presença:
        5.3.1. Insere as divisões da casa, insere data e hora de início e fim da simulação, a duração e frequência de cada simulação, e selecciona a opção de activar serviço de simulação;
6. Se o habitante escolhe a opção de ajustar as portas:
    6.2. Se pretende configurar o sistema de fumo e fogo:
        6.2.1. Insere as divisões da casa em que deve ser ligado e activa/desactiva a opção de portas anti-fogo em caso de fumo/fogo;
    6.3. Se pretende configurar o sistema de segurança:
        6.3.1. Activa/desactiva sistema de alarme para detecção de entrada de intrusos na casa, através das portas;
7. Se o habitante escolhe a opção de ajustar alarme:

7.2. Se pretende configurar o sistema de fumo e fogo:

7.2.1. Activar/desactivar sistema de alarme em caso de fumo e/ou fogo;

7.2.2. Activar/desactivar sistema de chamadas de emergência para bombeiros;

7.2.3. Inserir Divisões da casa pretendidas;

7.3. Se pretende configurar o sistema de segurança:

7.3.1. Escolher o tipo de alarme;

7.3.2. Activar/desactivar sistema de alarme para detecção de movimento;

7.3.3. Activar/Desactivar alarme para detecção de vidros quebrados;

7.3.4. Inserir Divisões da casa pretendidas;

7.3.5. Activar/desactivar sistema de chamadas de emergência para polícia;

8. Se o habitante escolhe a opção de ajustar *sprinkler*:

8.2. Se pretende configurar o sistema de fumo e fogo:

8.2.1. Activar/desactivar *sprinkler* activar/desactivar sistema de chamadas de emergência para bombeiros;

8.2.2. Inserir divisões da casa pretendidas.

Telemóvel

1. Se o utilizador pretender configurar o sistema por telemóvel, efectua uma ligação ao sistema central.

Polícia

1. Se o sistema central efectuar chamada de emergência para a polícia, ela dirige-se para o local da chamada e presta auxílio.

Bombeiros

1. Se o sistema central efectuar uma chamada de emergência para os bombeiros, eles dirigem-se ao local da chamada, prestando auxílio.

Actuadores

Actuador de Janelas

1. Se o sistema central receber informação das condições necessárias para que as janelas sejam abertas, o actuador de janelas abre as janelas;
2. Se o sistema central receber informação das condições necessárias para que as janelas sejam fechadas, o actuador de janelas fecha as janelas.

Interruptores de Luzes

1. Se o sistema central receber informação das condições necessárias para que as luzes sejam ligadas, os interruptores de luzes ligam as luzes;
2. Se o sistema central receber informação das condições necessárias para que as luzes sejam desligadas, o interruptor de luzes desliga as luzes.

Actuador de Portas

1. Se o sistema central receber informação das condições necessárias para que as portas sejam fechadas, o actuador de portas fecha as portas;

2. Se o sistema central receber informação das condições necessárias para que as portas sejam abertas, o actuador de portas abre as portas.

Actuador de Persianas

1. Se o sistema central receber informação das condições necessárias para que os estores sejam abertos, o actuador de estores abre os estores;
2. Se o sistema central receber informação das condições necessárias para que os estores sejam fechados, o actuador de estores fecha os estores.

Dispositivos

Ar Condicionado

1. Se o sistema central receber informação das condições necessárias para que o AC seja ligado, ou o seu termóstato indicar que atingiu o limite de temperatura dado pelo utilizador, o AC liga-se;
2. Se o sistema central receber informação das condições necessárias para que o AC seja desligado, ou o seu termóstato indicar que atingiu o limite de temperatura dado pelo utilizador, o AC desliga-se.

Sensores

Sensor de alarme

1. Detecta estado do alarme;
2. Envia informação do estado para o sistema central.

Sensor de fumo e fogo

3. Detecta presença/ausência de fumo e fogo;
4. Envia informação para o sistema central.

Sensor de janelas

1. Detecta estado das janelas;
2. Envia informação do estado para o sistema central.

Sensor de persianas

1. Detecta estado das persianas;
2. Envia informação do estado para o sistema central.

Sensor de portas

1. Detecta estado das portas;
2. Envia informação do estado para o sistema central.

Sensor de luzes

1. Detecta estado das luzes;
2. Envia informação do estado para o sistema central

Sensor de *Sprinkler*

1. Detecta estado do *sprinkler*;
2. Envia informação do estado do *sprinkler* para o sistema central.

Sensor de temperatura exterior

1. Detecta temperatura no exterior da casa;
2. Envia informação para o sistema central.

Sensor de temperatura interior

1. Detecta temperatura no interior da casa;
2. Envia informação para o sistema central.

Concerns

Compatibilidade

1. O sistema deve ser compatível com os dispositivos, actuadores e sensores;
2. O telemóvel deve ser compatível com a operadora;
3. O sistema deve ser compatível com a operadora.

Disponibilidade

1. O sistema deve garantir a disponibilidade dos sensores;
2. O sistema deve garantir a disponibilidade dos dispositivos;
3. O sistema deve garantir a disponibilidade dos actuadores;
4. O sistema deve garantir a disponibilidade do sistema central;
5. O sistema deve garantir a disponibilidade da polícia;
6. O sistema deve garantir a disponibilidade dos bombeiros;
7. O sistema deve garantir a disponibilidade nas comunicações.

Safety

1. O sistema deve garantir a segurança dos sensores;
2. O sistema deve garantir a segurança na activação/desactivação do sistema de extinção de incêndios;
3. O sistema deve garantir a segurança na activação/desactivação do alarme;
4. O sistema deve garantir a segurança na abertura/fecho de portas, janelas e estores;
5. O sistema deve garantir segurança quando efectua chamadas de emergência;

Usabilidade

1. O sistema deve possuir uma interface fácil de usar e compreender.

Regras de Composição

Composição de Usabilidade

1. O requisito 1 de usabilidade deve restringir todos os requisitos do utilizador e do administrador.

Composição de Compatibilidade

1. O requisito 1 de compatibilidade deve restringir todos os requisitos dos actuadores, sensores e dispositivos;
2. O requisito 2 de compatibilidade deve restringir o requisito 2 da operadora e todos os do telemóvel;
3. O requisito 3 de compatibilidade deve restringir o requisito 1 da operadora.

Composição de *Safety*

1. O requisito 1 de *safety* deve restringir todos os requisitos dos sensores;

2. O requisito 2 de *safety* deve restringir todos os requisitos do *sprinkler*;
3. O requisito 3 de *safety* deve restringir todos os requisitos do alarme;
4. O requisito 4 de *safety* deve restringir todos os requisitos de portas, janelas e estores;
5. O requisito 4 de *safety* deve restringir todos os requisitos de operadora.

Composição de Disponibilidade

1. O requisito 1 de disponibilidade deve restringir todos os requisitos dos sensores;
2. O requisito 2 de disponibilidade deve restringir todos os requisitos dos dispositivos;
3. O requisito 3 de disponibilidade deve restringir todos os requisitos dos actuadores;
4. O requisito 5 de disponibilidade deve restringir o requisito 1 da polícia;
5. O requisito 6 de disponibilidade deve restringir o requisito 1 dos bombeiros;
6. requisito 7 de disponibilidade deve restringir todos os requisitos da operadora.

Anexo G. Instruções para Teste

Introduction

The purpose of this test is to assess how easy is to learn and use the **VisualAore LDE.** This evaluation is done in two ways:

- **Concept Validation** – how adequate are the concepts and the easiness to learn them;
- **Usability Validation** - easy to understand and use the plugin;

To perform this test it's used the ***Via Verde*** case study, specified next. To facilitate the testing process, the case study is given with a reduced solution. The testers are only asked to use this solution to get the final model in the **VisualAore** plugin.

To start the test, the file **VisualAore.zip** must be unzipped into the plugins folder of Eclipse. To start with the tool, the following steps should be followed:

1. **File -> New -> Project -> General -> Project;**
2. **Right click on created project -> New -> Example -> AoreDiagram.**

Thanks for your colaboration!

Case Study – *Via Verde*

“In a road traffic pricing system, drivers of authorized vehicles are charged at toll gates automatically. The gates are placed at special lanes called green lanes. A driver has to install a device (a gizmo) in his/her vehicle. The registration of authorized vehicles includes the owner's personal data, bank account number and vehicle details. The gizmo is sent to the client to be activated using an ATM that informs the system upon gizmo activation. A gizmo is read by the toll gate sensors. The information read is stored by the system and used to debit the respective account. When an authorized vehicle passes through a green lane, a green light is turned on, and the amount being debited is displayed. If an unauthorized vehicle passes through it, a yellow light is turned on and a camera takes a photo of the plate (used to fine the owner of the vehicle). There are three types of toll gates: single toll, where the same type of vehicles pay a fixed amount, entry toll to enter a motorway and exit toll to leave it. The amount paid on motorways depends on the type of the vehicle and the distance travelled."

Viewpoints

ATM

1. The ATM sends the customer's card number, account number and gizmo identifier to the system for activation and reactivation;
    1.1. The ATM is notified if the activation or reactivation was successful or not;
        1.1.1. ln case of unsuccessful activation or reactivation the ATM is notified of the reasons for the failure.

Vehicle

1. The vehicle enters the system when it is within ten meters of the toll gate;
2. The vehicle enters the toll gate;
3. The vehicle leaves the toll gate;
4. The vehicle leaves the system when it is twenty meters away from the toll gate.

Unauthorised Vehicle

1. The vehicle number plate will be photographed.

Gizmo

1. The gizmo identifier is read by the system;
    1.1. The gizmo identifier is validated by the system;
    1.2. The gizmo is checked by the system for being active or not.

Tool Gate

Paying Toll

1. A green light is turned on if the gizmo is valid;
2. A yellow light is turned on if the gizmo is not present or invalid;
3. An alarm is sounded if the gizmo is not present or invalid;
4. The amount being debited is displayed if the gizmo is valid;
    4.1. The amount being debited depends on the class of the vehicle.

Single Toll

1. The amount being displayed is fixed.

Exit Toll

1. A yellow light is shown if the vehicle did not enter using a green lane;
2. The amount being debited depends upon the entry point.

Entry Toll

1. No signals are shown on passing an entry point.

Concerns

Correctness

1. The system must ensure correctness of the data:
    1.1. calculated within the system;
    1.2. exchanged with the environment;

Aspects

Compatibility

1. The system must be compatible with systems used to:
    1.1. activate and reactivate gizmos;
    1.2. deal with infraction incidents;
    1.3. charge for usage.

ResponseTime

1. The system needs to react in-time in order to:
    1.1. the gizmo identifier;
    1.2. turn on the light (to green or yellow);
    1.3. display the amount to be paid;
    1.4. photograph the plate number from the rear;
    1.5. sound the alarm;
    1.6. gizmo activation and reactivation.

Table 1. Impacts.

Vp / Concern	Gizmo	ATM	Paying Tool	Single Tool	Exit Tool	Entry Tool	Vehicle	Unauth. Vehicle
Response Time	√	√	√	√	√	√	√	√
Correctness	√		√	√	√	√		
Compatibility		√						

Table 2. Conflicts.

Apects / Aspects	Response Time	Correctness	Compatibility
Response Time		-	
Correctness			
Compatibility			

Table 3. Conflicts Resolution.

Vp / Concern	Gizmo	ATM	Paying Tool	Single Tool	Exit Tool	Entry Tool	Vehicle	Unauth. Vehicle
Response Time	1,0	√	1,0	1,0	1,0	1,0	√	√
Correctness	0,8		1,0	1,0	1,0	1,0		
Compatibility		√						

Composition

Compatibility

1st Rule:

Aspect: Compatibility, **NFRequirement:** 1.1

Constraint
- **Constraint Action:** ensure
- **Constraint Operator:** with

Requirement Viewpoint: ATM, **FRequirement:** all

Outcome Action: fulfilled

Tempo de Resposta

1st Rule:

Aspect: Response Time, **NFRequirement:** 1.1

Constraint
- **Constraint Action:** enforce
- **Constraint Operator:** between

Requirement Viewpoint: Vehicle, **FRequirement:** 1, 2

Outcome Action: Satisfied

Requirement Viewpoint: Gizmo, **FRequirement:** 1 **(include children)**

2nd Rule:

Aspect: Response Time, **NFRequirement:** 1.2

Constraint
- **Constraint Action:** enforce
- **Constraint Operator:** between

Requirement Viewpoint: Gizmo, **FRequirement:** 1 **(include children)**

Outcome Action: Satisfied

Constraint Operator: xor

Requirement Viewpoint: PayingToll, **FRequirement:** 1, 2

3rd Rule

Aspect: Response Time, **NFRequirement:** 1.3

Constraint
- **Constraint Action:** enforce
- **Constraint Operator:** between

Requirement Viewpoint: Paying Toll, **FRequirement:** 1

Requirement Viewpoint: Vehicle **FRequirement:** 3

Outcome Action: Satisfied

Requirement Viewpoint: PayingToll, **FRequirement:** 4 **(include children)**

4th Rule

Aspect: Response Time, **NFRequirement:** 1.4

Constraint
- **Constraint Action:** enforce
- **Constraint Operator:** between

Requirement Viewpoint: Paying Toll, **FRequirement:** 2

Requirement Viewpoint: Vehicle, **FRequirement:** 4

Outcome Action: Satisfied

Requirement Viewpointy: Unhautorized, **FRequirement:** 1

5th Rule

Aspect: Response Time, **NFRequirement:** 1.5

Constraint
- **Constraint Action:** enforce
- **Constraint Operator:** between

Requirement Viewpoint: Paying Toll, **FRequirement:** 2

Requirement Viewpoint: Vehicle, **FRequirement:** 4

Outcome Action: Satisfied

Requirement Viewpoint: Paying Tool, **FRequirement:** 3

6th Rule

Aspect: Response Time, **NFRequirement:** 1.6

Constraint
- **Constraint Action:** enforce
- **Constraint Operator:** on

Requirement Viewpoint: ATM, **FRequirement:** 1 **(include children)**

Outcome Action: fullfiled

Help

General:

- To view/edit the properties of the elements, right-click on canvas (editable surface) and select Show Properties View;
- To distribute the element in canvas, righ-click on canvas and select Arrange All (useful when returning to the editor, after editing on a sub-editor).

Requirements:

- To delete FRequirements or NFRequirements, right-click on it and click delete;
- To add FRequirements or NFRequirements through the pallete, click below the requirements already entered and above the separator bar compartment.

FRAggModules:

- Used to aggregate some viewpoint requirements (used in the composition rules).

Sub-Editors

- To acess the external editor, left-click on the border of the modules (Composition Rule Module, Viewpoint Module or Concern Module);
- Sub-editors must be saved before exit.

Composition Rules

- The elements of the Composition Rules must all be connected.

Anexo H. Questionário

A1. How often did you use a DSL workbench tool? If your answer is "1", please go to question B1.

Never ◡ 1 ◡ 2 ◡ 3 ◡ 4 ◡ 5 **Always**

A1.1. How long have you used it?

A1.2. Have you enjoyed it?

Not Really ◡ 1 ◡ 2 ◡ 3 ◡ 4 ◡ 5 **A lot**

A1.3. What for?

Conceptual Validation

B1. Did you understand the VisualAore language?

Very Bad ◡ 1 ◡ 2 ◡ 3 ◡ 4 ◡ 5 **Very Well**

B1.1. If not, what was missing?

B2. How easy did you find learning the concepts?

Very Difficult ◡ 1 ◡ 2 ◡ 3 ◡ 4 ◡ 5 **Very Easy**

B3. How do you identify the symbols representing the concepts?

Very Bad ◡ 1 ◡ 2 ◡ 3 ◡ 4 ◡ 5 **Very Good**

B3.1. Which one did you find inadequate?

B4. How do you identify the text representing the concepts?

Very Bad ◡ 1 ◡ 2 ◡ 3 ◡ 4 ◡ 5 **Very Good**

B5. How often did you find committing errors due to symbols similarity?

Never ◯1 ◯2 ◯3 ◯4 ◯5 **Always**

B6. How often did you find committing errors due to ambiguous vocabulary?

Never ◯1 ◯2 ◯3 ◯4 ◯5 **Always**

B7. What changes or additions do you propose to the language?

Usability Validation

C1. What time did you spent in developing the model through the VisualAore plug-in?

C2. How easily did you create the Aore model?

Very Difficult ◯1 ◯2 ◯3 ◯4 ◯5 **Very Easy**

C3. How do you evaluate the process of creating a model in the VisualAore plug-in?

Very Bad ◯1 ◯2 ◯3 ◯4 ◯5 **Very Good**

C4. Did you have difficulty using the VisualAore plug-in?

◯Yes ◯No

C4.1. If yes, what's the biggest difficulty you found?

C5. How did you feel about performing changes?

Very Difficult ◯1 ◯2 ◯3 ◯4 ◯5 **Very Easy**

C6. How physically demanding was performing the case study?

Very Difficult ◯1 ◯2 ◯3 ◯4 ◯5 **Very Easy**

C7. The outcome reflects what you were expecting?

Definitely Not ◯1 ◯2 ◯3 ◯4 ◯5 **Absolutely**

C8. How often did you feel unable to express what you intended?

Never ◯1 ◯2 ◯3 ◯4 ◯5 **Always**

C9. What are the advantages of the model obtained in the plug-in relatively to the specification done textual and manually?

C10. And the disadvantages?

C11. When creating the model in the plug-in, did you kept all the information specified textual and manually?

◯ Yes ◯ No ◯ Don't Know

C12. How often did you perform questions to the supervisor?

Never ◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 **Always**

C13. How confident did you feel during case study execution?

Not Confident ◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 **Very Confident**

C14. How often did you find trapped or confused during the case study?

Never ◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 **Always**

C15. How mentally demanding was the case study?

Very Difficult ◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 **Very Easy**

C16. What did you feel more difficult to reason/perform?

C17. How do you feel about the correctness of the performed case study?

Not Correct ◯ 1 ◯ 2 ◯ 3 ◯ 4 ◯ 5 **Very Correct**

C18. Have you ever used the Arcade tool? If your answer is "No", please go to question C19.

◯ Yes ◯ No ◯ Don't Know

C18.1. How do you rate de VisualAore plug-in comparing with the Arcade tool?

Very Worse ◯1 ◯2 ◯3 ◯4 ◯5 **Very Better**

C18.2. The VisualAore plug-in contains more information than the one that is provided by the Arcade?

◯Yes ◯No ◯Don't Know

C18.3. The VisualAore tool contains less information than the one provided by the Arcade?

◯Yes ◯No ◯Don't Know

C18.4. Comparing VisualAore with Arcade, what's the tool you think that offers a faster and better understanding of the models?

C18.5. Do you feel the VisualAore tool is a value-added compared to Arcade?

◯Yes ◯No ◯Don't Know

C18.6. Why?

C19. Do you consider the tool helpful?

Not Helpful ◯1 ◯2 ◯3 ◯4 ◯5 **Very Helpful**

C20. What is your overall appreciation of the VisualAore DSL?

Very Bad ◯1 ◯2 ◯3 ◯4 ◯5 **Very Good**

C21. What are, in your opinion, the weaknesses/difficulties of the VisualAore plug-in?

C22. What are, in your opinion, the strengths/facilities of the VisualAore plug-in?

C23. What are the innovations brought by the VisualAore plug-in?

C24. What do you suggest that can improve the VisualAore plug-in?

C25. Suggestions / global comments:

Anexo I. Manual do utilizador da LDE VisualAORE

1. Introdução

Este manual tem o objectivo de auxiliar o utilizador da LDE VisualAORE em duas vertentes: na vertente da instalação e na vertente da utilização.

Na secção 1.2 deste manual é explicado o procedimento para a instalação do *plug-in* VisualAORE, e na secção 1.3 são pormenorizadas as instruções de utilização da ferramenta.

1.1. Instruções de instalação

Para realizar a instalação do *plug-in* VisualAORE é necessário possuir o ambiente de desenvolvimento Eclipse, instalado no computador.

Preferencialmente, o utilizador deve efectuar o download da versão Eclipse Modeling Tools (Eclipse Galileo Sr2 Packages) em: http://www.eclipse.org/downloads/packages/release/galileo/sr2.

A versão do Eclipse acima mencionada já é provida dos plugins GMF e EMF, pelo que não é necessário efectuar qualquer actualização do mesmo.

Seguidamente, o utilizador deve descompactar o ficheiro VisualAORE.zip e colocar os seus ficheiros na pasta *plug-ins* do Eclipse (p.e. C:\galileo\eclipse\plugins). Nesta fase, deve-se reiniciar o Eclipse para que os *plug-ins* sejam carregados.

O ficheiro VisualAORE.zip deve conter os seguintes ficheiros:

- Aore.jar
- AoreC.jar
- AoreCR.jar
- AoreV.jar
- AoreEdit.jar
- AoreEditor.jar
- AoreDiagram.jar
- FiguresAore.jar

Neste momento estão reunidas as condições necessárias para utilizar a ferramenta VisualAORE. Assim, para abrir um editor VisualAORE, o utilizador deve iniciar o Eclipse, dirigir-se ao tab File e escolher New → Project → General →Project, como ilustra a Figura J.1. Em seguida deve clicar em Next, fornecer um nome ao projecto e clicar em Finish. Seguidamente deve clicar com o botão direito do rato no projecto criado, e escolher New → Example →AoreDiagram, tal como ilustra a Figura J.2. Deve posteriormente fornecer um nome ao ficheiro e clicar em Finish. Encontra-se neste momento pronto para dar iniciar o processo de modelação na ferramenta.

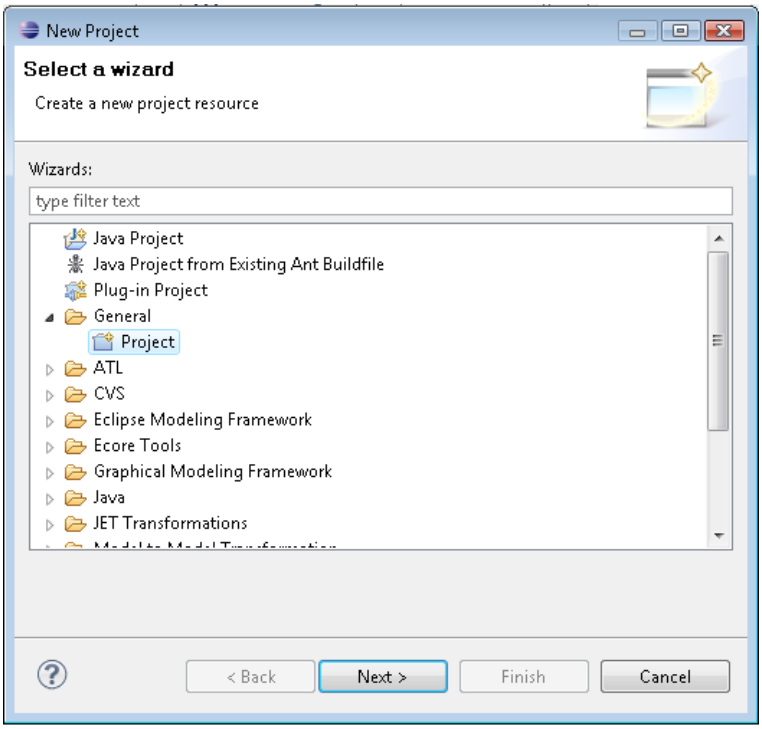

Figura I.1. Criação de um novo projecto.

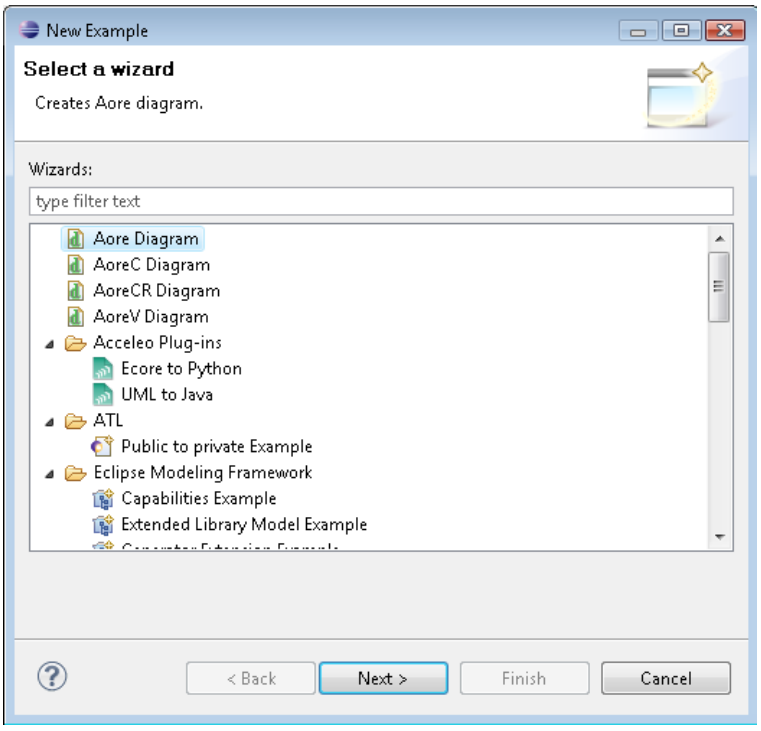

Figura I.2. Criação do diagrama AORE.

1.2. Instruções de utilização

Para iniciar a utilização do *plug-in* VisualAORE, o utilizador deve começar por observar o menu do editor base. Este menu é composto por 5 separadores, tal como ilustra a Figura J.3:

No separador *Modules* o utilizador escolhe o módulo de agregação de elementos que deseja, consoante os elementos que pretende agregar. No separador *Concern*, o utilizador possui os

elementos relacionados com *concerns*, tal como nos separadores *Composition Rule* e *Viewpoint*. O separador *Connections* possui todas as ligações possíveis de efectuar num modelo.

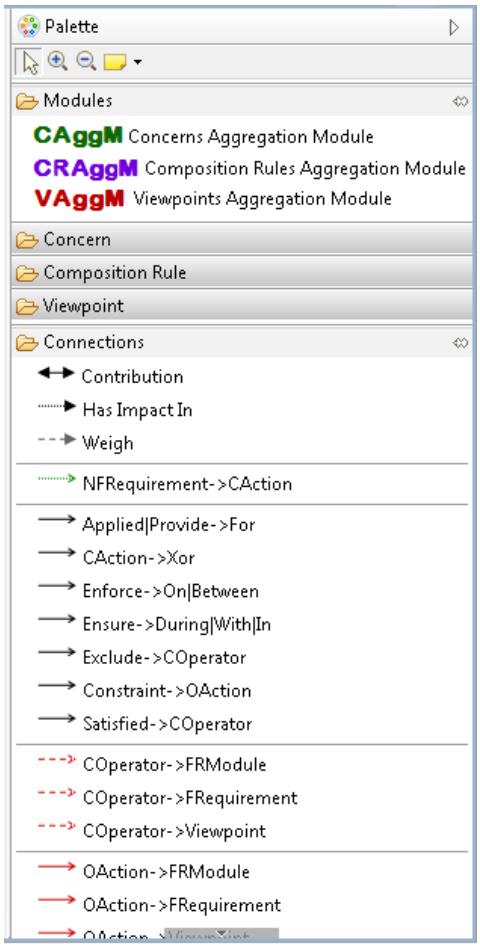

Figura I.3. Menu do editor base.

Para dar inicio à construção de um modelo, o utilizador deve colocar os módulos de agregação no editor base, uma vez que são os elementos que suportam o modelo.

Os elementos podem ser colocados no editor por duplo clique no elemento desejado no menu ou por arrasto para o editor. De notar que elementos que são postos dentro de outros elementos apenas podem ser colocados por arrasto.

Após colocar os módulos no editor, o utilizador pode então colocar os elementos dentro dos respectivos módulos.

Cada módulo de agregação presente no editor base possui um sub-editor associado. Deste modo, o utilizador pode editar um módulo no editor base, ou editar num editor em separado, onde apenas é visível a informação que lhe pertence. Para editar um módulo em separado, é necessário efectuar um duplo.clique com o botão direito do rato no rebordo do módulo em questão. Feito isto, é disponibilizado um novoeditor automaticamente, com um menu que contém os elementos que podem ser utilizados nesse editor. Sempre que o utilizador desejar voltar ao editor base para visualizar/editar o modelo global, deve fechar e guardar as edições efectuadas nos sub-editores, para que estas sejam carregadas.

Para saber quais os elementos que podem ser colocados num editor ou dento de um elemento, basta clicar com o botão esquerdo do rato no local onde se deseja colocar o novo elemento, e clicar na opção desejada, dentro das opções que são disponibilizadas. A título de exemplo, a imagem abaixo ilustra os elementos que podem ser colocados no sub-editor do módulo de agregação de *concerns*, sendo estes Aspectos e *Concerns*, tal como os símbolos indicam.

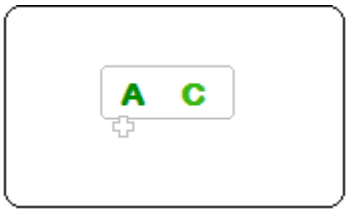

Figura I.4. Elementos a colocar no sub-editor do módulo de agregação de *concerns*.

Ao clicar com o botão direito do rato num editor, o utilizador acede a várias opções. Algumas delas são descritas em seguida:

***File**:* permite que o utilizador exporte o conteúdo do editor para uma imagem. Caso o utilizador seleccione um elemento, e em seguida clique nele com o botão direito do rato, é possível exportar apenas o elemento seleccionado.

***Delete from Model**:* permite eliminar todo o conteúdo do editor, ou caso se seleccione previamente um elemento, é eliminado apenas o elemento seleccionado. Também é possível eliminar um elemento efectuando a sua selecção e pressionando a tecla Delete.

***Select**:* permite seleccionar todos os elementos do editor.

***Arrange All**:* permite dispor todos os elementos ao longo do editor.

Show Property Views: mostra o separador que contém as propriedades dos elementos do editor. Estas propriedades podem ser nome, id, atributos onde se pode escolher um valor de um tipo enumerado, entre outros.

Com vista a efectuar ligações entre os elementos, deve-se seleccionar a ligação desejada no menu, em seguida deve-se clicar no elemento de origem e por fim no elemento de destino. A zona dos elementos onde se deve clicar é no cabeçalho ou ao longo do rebordo.