Fábio Jorge Tavares Soares

Licenciado em Engenharia Informática

Uma Abordagem para Derivar Modelos de Requisitos a partir de Mecanismos de Reconhecimento de Voz

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Prof. Doutor João Baptista da Silva Araújo Júnior,
Professor Auxiliar,
Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia

Co-orientador : Fernando Wanderley, Doutorando,
Universidade Nova de Lisboa - Faculdade de Ciências e Tecnologia

Júri:

Presidente: Prof. Doutora Sofia Carmen Faria Cavaco

Arguente: Prof. Doutor José Alberto Rodrigues Pereira Sardinha

Vogal: Prof. Doutor João Baptista da Silva Araújo Júnior

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Novembro, 2014

Uma Abordagem para Derivar Modelos de Requisitos a partir de Mecanismos de Reconhecimento de Voz

Dedico este trabalho à minha mãe.
Obrigado por tudo.

Agradecimentos

Em primeiro lugar, quero agradecer ao meu orientador João Araújo por toda a sua dedicação, apoio, disponibilidade ao longo do desenvolvimento da dissertação e, sobretudo, da muita paciência que teve ao responder aos inúmeros *e-mails* que pediam conselhos e revisões. Em seguida, mas não menos importante, quero agradecer ao meu co-orientador Fernando Wanderley pelo interesse, auxílio e conselhos que ajudaram a melhorar o trabalho. A vossa contribuição e experiência fizeram-me aprender sempre algo novo ao longo do trabalho, um muito obrigado.

Também quero agradecer à Associação de Cegos e Amblíopes de Portugal, mais especificamente à Susana Venâncio pela toda colaboração prestada. Ao Prof. Vasco Amaral pelos conselhos dados ao longo do trabalho.

Às pessoas que colaboraram no processo de avaliação, nomeadamente os meus colegas, à Prof. Sofia Cavaco por ter direccionado às pessoas certas para a impressão de conteúdos em braile, ao Glen Shires pela atenção e ajuda na inclusão do serviço de reconhecimento de voz da Google no trabalho e ao Sérgio Neves e Fernando Santos por se terem incluído como participantes cegos, tornando o trabalho mais enriquecedor.

Ao Departamento de Informática da Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (DI - FCT/UNL), não só pela oportunidade e condições para desenvolver este trabalho, como também pelo apoio financeiro com a Bolsa de Investigação, através do Centro de Informática e Tecnologias de Informação (CITI) e FCT/MEC.

Aos meus amigos que estiveram sempre presentes ao longo do meu percurso académico, principalmente ao António Silva, Fernando Branco, João Furtado, João Rato, Paulo Sequeira, Francisco Inácio, Filipe Patrício, Filipe Lança, Rui Maciel, Marco Mendão, Ricardo Monteiro, Inês Oliveira, Filipa Belo, João Regateiro, Márcio Duarte, Ricardo Dias, Pedro Cardoso, Simão Santana, entre outros pelo apoio, incentivo e bons momentos.

A toda a minha família, nomeadamente à minha avó Fernanda Mateus, à minha irmã Andreia Soares e ao Batista pela motivação e ajuda ao longo deste tempo.

Como agradecimento final e especial, quero agradecer à minha mãe Ana Paula Tavares não só pelo incentivo dado ao longo da dissertação, mas também pelo apoio sempre presente ao longo da minha vida.

Resumo

A elicitação de requisitos é uma das primeiras actividades do processo de Engenharia de Requisitos. Através desta etapa é possível capturar e estruturar os requisitos dos *stakeholders* e do sistema a ser implementado. Tipicamente esta estruturação é representada através de notação gráfica com o auxílio de ferramentas CASE. Estas ferramentas tornam esta actividade exclusiva, em termos de acessibilidade, a engenheiros sem deficiências físicas devido à complexidade das funcionalidades oferecidas pelas mesmas que permitem construir estes modelos.

Nesta dissertação de mestrado é proposto desenvolver uma abordagem com suporte de uma ferramenta para melhorar a acessibilidade, e consequentemente, integrar um engenheiro de requisitos com limitações físicas na actividade de elicitação de requisitos. A ferramenta também possibilita uma alternativa para os *stakeholders* em geral para produzir modelos sem usar as mãos quando for mais conveniente. Esta abordagem propõe usar mecanismos de voz na geração de modelos de requisitos, mais concretamente modelos de requisitos orientados a objectivos, modelos de objectos e modelos de *features* usando técnicas de **M***odel*-**D***riven* **D***evelopment* (MDD) (e.g., metamodelos). O *stakeholder* assim irá capturar os seus requisitos através de mecanismos de reconhecimento de voz, sendo automaticamente realizada uma transformação do seu discurso para um modelo KAOS, para um modelo conceptual ou para um modelo de *features*.

Palavras-chave: Acessibilidade, Reconhecimento de voz, Modelos de Requisitos

Abstract

Requirements elicitation is one of the first activities of the Requirements Engineering process. Through this activity, it is possible to capture stakeholder's requirements for the system to be implemented and structure them along with the system's requirements. Typically, the structure of these requirements is represented by graphical notation with the aid of CASE tools. These tools make this activity exclusive to requirements engineers with unhampered accessibility due to the complexity of the functionalities of such tools.

In this master's dissertation, an approach with tool support is proposed to improve the accessibility of the elicitation process and integrate a requirements engineer with disabilities in the requirements elicitation activity. This tool also provides an alternative to the stakeholders in general to produce models without using their hands when it is most convenient. With this approach, we intend to use speech recognition mechanisms to generate requirements models, more specifically goal-oriented models, object models and feature models. Through speech recognition, the engineer is capable of voicing the requirements which are automatically used to generate a KAOS model, a conceptual model or a feature model using techniques of Model-Driven Development (MDD).

Keywords: Accessibility, Speech Recognition, Requirements Models

Lista de Acrónimos

ACAPO	**A**ssociação dos **C**egos e **A**mblíopes de **P**ortugal
API	**A***pplication* **P***rogramming* **I***nterface*
BNF	**B***ackus Naur Form*
CASE	**C***omputer-***A***ided* **S***oftware* **E***ngineering*
CIM	**C***omputation* **I***ndependent* **M***odel*
COM	**C***omponent* **O***bject* **M***odel*
CRUD	**C***reate*, **R***ead*, **U***pdate* e **D***elete*
DFD	**D***ata* **F***low* **D***iagrams*
DSL	**D***omain* **S***pecific* **L***anguage*
ECL	**E***psilon* **C***omparation* **L***anguage*
EGL	**E***psilon* **G***eneration* **L***anguage*
EMF	**E***clipse* **M***odeling* **F***ramework*
EML	**E***psilon* **M***erging* **L***anguage*
EOL	**E***psilon* **O***bject* **L***anguage*
Epsilon	**E***xtensible* **P***lataform of* **I***ntegrated* **L***anguages for m***O***del ma***N***agement*
ETL	**E***psilon* **T***ransformation* **L***anguage*
EVL	**E***psilon* **V***alidation* **L***anguage*
EWL	**E***psilon* **W***izard* **L***anguage*
GEF	**G***raphical* **E***diting* **F***ramework*
GME	**G***eneric* **M***odeling* **E***nvironment*
GMF	**G***raphical* **M***odeling* **F***ramework*
GQM	**G***oal-***Q***uestion-***M***etric*
GUI	**G***raphical* **U***ser* **I***nterface*
HMM	**H***idden* **M***arkov* **M***odel*
JSAPI	**J***ava* **S***peech* API
JSGF	**J***ava* **S***peech* **G***rammar* **F***ormat*
JSML	**J***ava* **S***peech* **M***arkup* **L***anguage*

JSON	**J***ava***S***cript* **O***bject* **N***otation*
KAOS	**K***nowledge* **A***cquisition in aut***O***mated* **S***pecification* **K***eep* **A***ll* **O***bjectives* **S***atisfied*
LDE	**L**inguagem para **D**omínio **E**specífico
LIFO	**L***ast* **I***n* **F***irst* **O***ut*
MDA	**M***odel-***D***riven* **A***rchitecture*
MDD	**M***odel-***D***riven* **D***evelopment*
MOF	**M***eta* **O***ject* **F***acility*
NFR	**N***on* **F***unctional* **R***equirements*
OCL	**O***bject* **C***onstraint* **M***odel*
OMG	**O***bject* **M***anagement* **G***roup*
PCM	**P***ulse-***C***ode* **M***odulation*
PICO	**P***opulation*, **I***ntervention*, **C***omparation* e **O***utcomes*
PIM	**P***lataform-***I***ndependent* **M***odel*
PLUMB	*ex***PL***oring graphs at* **UMB**
PSM	**P***lataform-***S***pecific* **M***odel*
SAPI	**M***icrosoft* **S***peech* API
SD	**S***trategic* **D***ependency*
SR	**S***trategic* **R***ationale*
TCP/IP	**T***ransmission* **C***ontrol* **P***rotocol*/**I***nternet* **P***rotocol*
TeDUB	**T***echnical* **D***iagram* **U***nderstanding for the* **B***lind*
UML	**U***nified* **M***odelling* **L***anguage*
VODER	**V***oice* **O***perating* **D***emonstrator*
Wi-Fi	**W***ireless* **F***idelity*
WIMP	**W***indow*, **I***con*, **M***enu* e **P***ointing device*
WLAN	**W***ireless* **L***ocal* **A***rea* **N***etwork*
XML	*e***X***tensible* **M***arkup* **L***anguage*
XMI	**X***ML* **M***etadata* **I***nterchange*

Conteúdo

Lista de Figuras

Lista de Tabelas

Listagens

1

Introdução

1.1 Contexto e Motivação

Os Sistemas de Informação têm vindo a ganhar um papel decisivo na sociedade onde nos encontramos. É cada vez mais importante que estes sejam entregues com qualidade, dentro do prazo delimitado e que não excedam o orçamento. Segundo estudos realizados pelo Standish Group em 2013 [Gro13], apenas 39% dos projectos de *software* foram entregues com todas as especificações pedidas pelo cliente e na data estipulada, 43% não cumpriram o prazo de entrega e não completaram a especificação pedida, e por fim, 18% foram cancelados numa instância do ciclo de vida do desenvolvimento do *software*. O relatório refere-se a factores críticos que contribuem para o sucesso de um produto, alguns dos quais relacionados com a **elicitação de requisitos**: envolvimento do cliente, definição explícita dos requisitos e objectivos de negócios claros.

A **elicitação de requisitos**, uma das primeiras actividades do processo de Engenharia de Requisitos [Pre09], consiste em realizar o levantamento e estruturação dos requisitos dos *stakeholders*[1] e do sistema. Para além da elicitação, é importante detalhar as restantes actividades tais como especificação, a validação e a gestão dos requisitos [KS98], tornando a Engenharia de Requisitos uma área crítica no desenvolvimento de um projecto de *software*.

Nesta fase de elicitação e análise, a estruturação dos requisitos é representada através de notação textual e de modelos, ou seja, gráfica. Estes artefactos são responsáveis por representar essencialmente as expectativas e objectivos dos *stakeholders*. No caso dos

[1] *Stakeholders*: Partes interessadas num projecto de desenvolvimento de *software*.

modelos, estes são estruturados graficamente com o objectivo de facilitar a comunicação e resolução de problemas, graças à eficácia cognitiva que uma representação de um elemento, pertencente a um gráfico, apresenta na compreensão humana [MHM10]. Esta representação de um elemento é associada a um padrão que, por sua vez, está ligado a um conceito. Considera-se um conceito como a semântica de uma propriedade e o padrão como a representação visual da semântica. Por exemplo, na abordagem KAOS [DLF93], cuja finalidade é a modelação de requisitos orientada a objectivos, o conceito de operacionalização é representado por uma oval.

Muitas das ferramentas CASE[2], como se focam na sua representação gráfica, tornam esta actividade exclusiva do ponto de vista de acessibilidade a engenheiros de requisitos sem limitações físicas, devido à complexidade das funcionalidades que as ferramentas oferecem para construir esses modelos. Portanto, é bastante difícil que engenheiros de requisitos com deficiências consigam integrar-se num projecto.

É neste contexto apresentado anteriormente que este trabalho propõe uma abordagem para melhorar a acessibilidade na actividade de elicitação, para engenheiros de requisitos com limitações físicas, envolvendo mecanismos de reconhecimento de voz para gerar modelos de requisitos. A voz do engenheiro poderia ser interpretada, através de uma aplicação, permitindo ao membro da equipa de desenvolvimento ter à sua disposição uma opção ágil para construir modelos de requisitos. A geração de modelos de análise de requisitos a partir do reconhecimento de voz será executada através de mecanismos do Desenvolvimento Orientado a Modelos, do inglês **M***odel*-**D***riven* **D***evelopment* (MDD). O MDD tem como principal foco o uso de modelos como artefactos principais durante o desenvolvimento de um sistema de *software*.

1.2 Objectivo do Trabalho

Com esta dissertação pretende-se desenvolver uma ferramenta acessível para engenheiros de requisitos com limitações físicas (especialmente cegos), para que possam colaborar com os vários membros da equipa de desenvolvimento na actividade de elicitação de requisitos. Durante o processo de elicitação, e através de mecanismos de reconhecimento de voz, o engenheiro utiliza a voz (expressando comandos em inglês), como meio de modelação dos requisitos em vez de utilizar os habituais periféricos, como o rato e o teclado, para manipular as ferramentas de modelação. A modelação pode ser feita através dos mecanismos de reconhecimento de voz que podem ser usados na construção de modelos de requisitos tradicionais, tais como os modelos de objectivos (e.g., abordagem KAOS), conceptual e de *features*.

Para que isto se torne possível recorremos à definição de gramáticas para poder definir os modelos através da voz e de mecanismos MDD, nomeadamente metamodelação, e de Linguagens para Domínio Específico (do inglês **D***omain* **S***pecific* **L***anguage* - DSL) para

[2] **C***omputer*-**A***ided* **S***oftware* **E***ngineering*: Ferramentas especificadas para apoiar no desenvolvimento de sistemas de *software*.

representar graficamente os modelos. Portanto, o engenheiro de requisitos descreve o que se pretende desenvolver no sistema utilizando reconhecimento de voz *speech-to-text*. Com a modificação da voz para texto, são realizadas transformações para gerar os modelos de objectivos KAOS, conceptual e de *features*. Depois de uma operação ter sido executada, o *stakeholder* é notificado do seu resultado através de mecanismos de sintetização de voz.

1.3 Principais Contribuições

Este trabalho apresenta o uso de reconhecimento de voz, através de uma API[3], melhorando a acessibilidade de ferramentas CASE na modelagem de requisitos e, consequentemente, a integração de um engenheiro de requisitos com limitações físicas no desenvolvimento de um projecto informático.

Esta abordagem também contribui como uma opção de desenvolvimento para ser utilizada numa abordagem colaborativa de requisitos, de modo a evitar a quebra de raciocínio ao registar os mesmos num documento. Ao usar a voz como mecanismo de *input* para a elicitação, os membros da equipa de desenvolvimento não despendem tanto tempo a manipular a ferramenta para construir os modelos, quando o mais importante é especificar correctamente os requisitos com os *stakeholders*.

Contudo o objectivo não é só para melhorar a acessibilidade de *stakeholders* com deficiências, mas também para fornecer uma alternativa para os interessados em geral, para produzir modelos sem mãos quando for mais conveniente.

Por fim, contribuímos para o estado da arte com o artigo em [SAW15], a ser publicado na conferência SAC ACM.

1.4 Organização do Documento

Este documento está estruturado da seguinte forma:

- **Capítulo 2 - Engenharia de Requisitos**: é introduzido o processo de Engenharia de Requisitos, com maior ênfase na elicitação de requisitos. De forma a reforçar essa actividade, são apresentadas e descritas algumas abordagens de requisitos: modelo de objectivos, objectos e modelação da variabilidade de domínio.
- **Capítulo 3 - Reconhecimento e Sintetização de Voz**: breve descrição dos conceitos principais sobre o reconhecimento e sintetização de voz, os seus processos, tal como a descrição dos conceitos de gramática usando o Java Speech API.
- **Capítulo 4 - Desenvolvimento Orientado a Modelos**: descrição dos seus conceitos principais e das DSLs que também é um dos aspectos mais relevantes para a dissertação.

[3] **A***pplication* **P***rogramming* **I***nterface*: Conjunto de serviços disponibilizados por uma aplicação.

- **Capítulo 5 - Trabalhos Relacionados**: é feito um estudo sobre os trabalhos relacionados nesta área, na qual são discutidos alguns trabalhos sobre acessibilidade em ferramentas CASE.
- **Capítulo 6 - Abordagem VoiceToModel**: é realizada uma descrição pormenorizada sobre a ferramenta VoiceToModel, isto é, como é composta a sua arquitectura, qual o modelo de interacção, os comandos específicos para cada modelo e a descrição de técnicas que permitem o melhor funcionamento do programa.
- **Capítulo 7 - Avaliação**: aborda o capítulo de avaliação, expondo o processo de planeamento, as métricas estabelecidas para avaliar alguns aspectos da implementação e a análise do *feedback* dado pelos utilizadores.
- **Capítulo 8 - Conclusão**: no último capítulo são tiradas as conclusões do trabalho efectuado, as suas limitações e a descrição de possíveis trabalhos futuros.
- **Anexo A - Comandos**: descrição completa de todos os comandos implementados dado o seu comando, qual o tipo de *feedback*, nota adicional e a complexidade temporal que demora até retornar um resultado.
- **Anexo B - Anotações**: notações textuais de cada metamodelo, destacando as notações criadas para gerar um comando associado a uma classe do metamodelo.
- **Anexo C - Questionário**: exemplar do questionário fornecido a todos os utilizadores que usaram a ferramenta VoiceToModel.
- **Anexo D - Comandos em braile**: descrição das tarefas e dos comandos implementados em formato braile exclusivamente para os utilizadores cegos.

2

Engenharia de Requisitos

2.1 Conceitos

Engenharia de Requisitos é a etapa inicial do desenvolvimento de um projecto de *software*, que se não for devidamente especificada pode comprometer o resto do desenvolvimento de um projecto [RKR13]. Segundo Kotonya e Sommerville [KS98], a Engenharia de Requisitos consiste nas actividades de extracção, documentação, validação e manutenção de um conjunto de requisitos. Zave apresenta a sua perspectiva da seguinte maneira [Zav97]: "*A engenharia de requisitos é o ramo da engenharia de software que se preocupa com os objectivos do mundo real, as suas funcionalidades e restrições em sistemas de software. Também está preocupada com a relação desses factores com as especificações precisas do comportamento do software e sua evolução ao longo do tempo e entre as famílias de software.*".

A IEEE [Sep90] define um requisito como: 1) uma condição ou capacidade de que um utilizador necessita para resolver um problema ou atingir um objectivo; 2) uma condição ou capacidade que deve ser alcançada por um sistema ou componente do sistema para satisfazer um contracto, padrão, especificação ou outro documento formalmente imposto; 3) uma representação documentada de uma condição ou capacidade referida em 1) ou 2).

No entanto, os requisitos não possuem todos a mesma natureza, visto que podem abordar vários aspectos de um projecto, havendo a necessidade de classificar cada requisito. Um **requisito funcional** descreve uma funcionalidade que o sistema tem para oferecer ao utilizador e o seu conjunto define o comportamento do sistema. O registo de uma pessoa com os seus dados num sistema de informação é um exemplo de um requisito funcional. Um **requisito não funcional** define restrições que o sistema deve cumprir. Uma aplicação que seja executada em qualquer ambiente é vista como um exemplo

de um requisito não funcional (i.e., portabilidade). Porém, esta designação não é linear dado que existem requisitos não funcionais de diversos tipos. A abordagem de Kotonya e Sommerville refere três tipos [KS98]: **requisitos do produto** (e.g., segurança, eficiência, usabilidade), **requisitos externos** (e.g., requisitos legais), e **requisitos do processo** (e.g., normas, padrões).

Os **requisitos de domínio** são características, funcionais e não funcionais, provenientes do domínio da aplicação. Através destas características pode ser gerada uma família de várias aplicações do mesmo domínio. Por fim, os **requisitos do utilizador** descrevem os requisitos que o sistema tem para oferecer, funcionais e não funcionais, de forma clara e explícita a um utilizador sem recorrer a detalhes técnicos de desenho.

2.2 Processo em Engenharia de Requisitos

Aqui a interacção dos engenheiros de requisitos com os demais *stakeholders* é fundamental. Um bom engenheiro de requisitos deve ter a capacidade de compreensão e comunicação sobre um problema modo a que o sistema fique consistente com as especificações dos clientes. Christensen et al. relatam quatro características que defendem serem fulcrais [Chr+96]:

- **Aptidões**: deve-se ter uma experiência do ciclo de vida inteiro no desenvolvimento de *software*. É importante evitar o risco de focar em requisitos que não sejam tão importantes que os restantes, pois caso aconteça, implica num esforço adicional para dedicar mais atenção aos requisitos mais prioritários;

- **Processo**: o engenheiro de requisitos deve controlar a evolução dos requisitos, pois a constante mudança pode implicar o incumprimento dos prazos de entrega. De modo que as mudanças agradem tanto ao cliente, como ao desenvolvedor, é crucial assegurar o processo até ao fim do desenvolvimento do sistema;

- **Comunicação**: um dos aspectos fundamentais é a comunicação - saber ouvir tanto as partes que conhecem os termos técnicos, como quem não conhece. É essencial negociar da melhor forma com o objectivo de implementar o produto como foi solicitado.

- **Tecnologia**: um engenheiro de requisitos deve compreender o domínio sobre o problema que está a ser resolvido. É necessário ter o conhecimento técnico de formar e aplicar os requisitos numa estrutura formal. Essa estruturação é representada numa linguagem que pode ser expressa em texto ou notação gráfica.

As actividades de extracção, documentação, validação e manutenção fazem parte do **processo** de Engenharia de Requisitos. Na Figura 2.1 é apresentada uma perspectiva do processo.

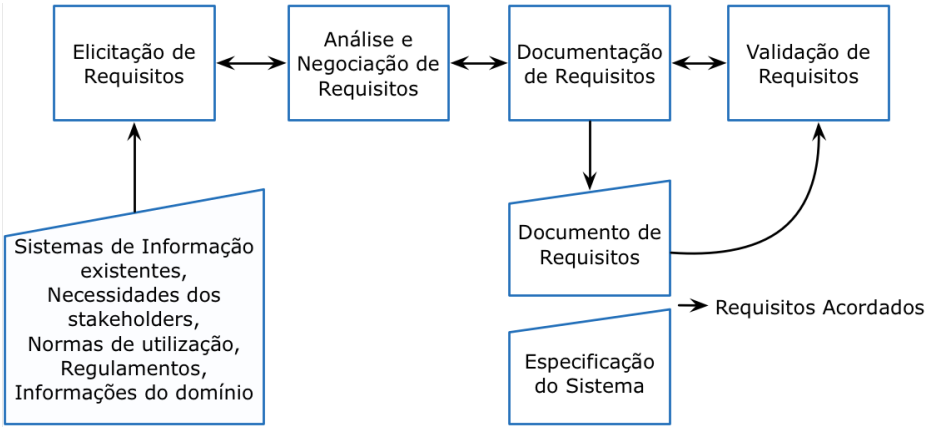


Figura 2.1: Modelo de actividades *Coarse-grain* do processo de Engenharia de Requisitos [KS98].

O processo de Engenharia de Requisitos inicia-se com a **elicitação de requisitos** onde os engenheiros da área questionam às partes interessadas sobre o que pretendem do produto, a sua finalidade no mercado e os seus requisitos. Para tal é necessário que exista uma conversação entre as duas entidades. Para além do levantamento dos requisitos, a elicitação estrutura os requisitos dos *stakeholders* e do sistema. É uma fase bastante importante visto que vai ser o reflexo do comportamento do *software*. Roger S. Pressman [Pre09] separa esta actividade em duas: a **concepção da ideia** que tem como objectivo identificar o domínio do problema; quem são os *stakeholders*; qual é a solução para o produto e a **elicitação de requisitos**, actividade idêntica ao que foi descrito pela perspectiva de Kotonya e Sommerville.

Na **análise e negociação** os requisitos são analisados em detalhe, podendo haver a necessidade de negociar com os *stakeholders* quando estes têm que ser modificados. Por norma esta negociação ocorre quando não há recursos disponíveis para implementar algum dos requisitos, quando os clientes solicitam mais funcionalidades que excedam o prazo de entrega do produto, ou então quando os clientes requerem requisitos que entrem em conflito entre si. Roger S. Pressman [Pre09] segue a mesma ideia da negociação referida por Kotonya e Sommerville. No entanto, refere que deve haver uma **elaboração**, onde todo o trabalho gerado através da iniciação da ideia e elicitação é posto em prática. Toda a informação obtida é mapeada para um modelo de requisitos refinado, identificando os aspectos funcionais do sistema e, posteriormente, criando cenários com actores que vão interagir com o sistema. No fim da concepção dos cenários é gerado o modelo de domínio, isto é, as classes que identificam o desenho do sistema e as relações entre elas.

Na **documentação de requisitos** [KS98], ou **especificação** [Pre09], os requisitos são registados num documento escrito em linguagem natural, modelo gráfico, fórmulas matemáticas, cenários ou protótipos. Para a **validação de requisitos** são usadas técnicas que validam a especificação do projecto com os requisitos estabelecidos (e.g., revisão de requisitos, prototipagem). Através desta técnica é garantido que os requisitos não são ambíguos, e que possíveis omissões e/ou erros possam ser detectados e corrigidos a tempo,

evitando que ocorram consequências maiores a nível de custo de produção e incumprimento de prazos.

Para além destas actividades, há a **gestão de requisitos** [KS98; Pre09] que está presente em todas as técnicas, ajudando a equipa de projecto a identificar, controlar e detectar mudanças de requisitos ao longo da vida de um projecto.

As actividades descritas acima, numa perspectiva de Kotonya e Sommerville e Roger S. Pressman, são defendidas como as principais e as que devem ser implementadas pelas instituições. No entanto, nenhuma destas técnicas refere explicitamente o uso de estratégias de acessibilidade para membros da equipa de desenvolvimento com deficiências.

2.3 Abordagens de requisitos

O processo de Engenharia de Requisitos nem sempre é fácil de ser executado, especialmente se for no contexto de sistemas em grande escala. Um dos problemas encontra-se na elicitação de requisitos [Pre09; NE00]. É referido que os clientes, por vezes, omitem especificações do sistema que consideram óbvias, solicitam funcionalidades que não são computacionalmente possíveis, ou pretendem modificar os requisitos do sistema durante as iterações do desenvolvimento do projecto, criando problemas de volatilidade.

As próximas subsecções irão referir em detalhe os modelos de análise de requisitos usados na presente dissertação tais como a **Engenharia de Requisitos Orientados a Objectivos** (subsecção 2.3.1), com o objectivo de estruturar requisitos usando a noção de objectivos; modelar os requisitos tendo como base os objectos com a **Engenharia de Requisitos Orientados a Objectos** (subsecção 2.3.2) e modelar as variabilidades de um domínio utilizando o **Modelo de Variabilidade** (subsecção 2.3.3).

Existem outras abordagens que não irão ser discutidas em detalhe, tais como os ***viewpoints*** e **aspectos**. Os *viewpoints* permitem estruturar e gerir a fase de análise e elicitação de requisitos para o sistema a partir de pontos de vista diferentes (e.g., cliente, utilizador) [SE05]. Com os aspectos pretende-se identificar e especificar características transversais[1], de forma separada, com o objectivo de modularizar melhor um sistema [Ara+05].

2.3.1 Engenharia de Requisitos Orientada a Objectivos

Um objectivo é uma técnica aplicada em Engenharia de Requisitos Orientada a Objectivos que o sistema deve alcançar [LA01]. Para definir como este objectivo deve ser alcançado, podem-se estabelecer mais objectivos que decompõem o objectivo inicial, até chegar a uma sequência de requisitos que descreve de que forma é que esse objectivo pode ser usado. Este paradigma é usado durante o processo de Engenharia de Requisitos, nas seguintes actividades [LA01]: elicitação, elaboração, estruturação, análise, especificação, negociação, documentação e modificação.

A Engenharia de Requisitos Orientada a Objectivos deve ser aplicada no processo

[1] Aspecto, a nível de requisitos, que se encontra em vários pontos do sistema.

devido às seguintes razões:

- **completude**: os objectivos caracterizam os requisitos de uma forma precisa para garantir a qualidade do sistema;
- **requisitos precisos**: definem um critério preciso para justificar a existência desse requisito dado um conjunto de objectivos que são necessários no sistema;
- **aumento da legibilidade**: o refinamento dos objectivos torna os requisitos documentados com elevada complexidade numa estrutura para melhorar a legibilidade;
- **conflitos**: ao desenvolver dois ou mais objectivos é possível detectar conflitos entre os mesmos e prover uma possível solução.

Nas próximas subsubsecções vão ser apresentadas as principais metodologias que permitem utilizar a Engenharia de Requisitos Orientada a Objectivos: **KAOS**, **i* framework** e **NFR framework**. A abordagem KAOS terá um foco principal em relação às restantes dado que é um dos modelos de análise que irá ser integrado na dissertação. A razão pela qual o KAOS foi escolhido deve-se ao facto de ser uma abordagem muito usada no contexto académico e industrial auxiliado pela ferramenta comercial Objectiver [IT14]. Por sua vez o *i* framework* foca-se no aspecto organizacional e o NFR *framework* em requisitos não funcionais, que não são o foco desta dissertação.

KAOS

Knowledge **A**cquisition in aut**O**mated **S**pecification ou **K**eep **A**ll **O**bjectives **S**atisfied [DLF93] é uma metodologia que tem como origem a cooperação entre a Universidade de Oregon e a Universidade de Louvain (Bélgica) em 1990 [IT07], com o objectivo de criar modelos de requisitos orientados a objectivos dentro do contexto de Engenharia de Requisitos.

O KAOS contém quatro modelos que ajudam a especificar a sua estrutura [IT07]: objectivos, objectos, operações e agentes. Os quatro modelos são representados através de notação gráfica, em formato de grafo, com o objectivo de ajudar a resolver problemas do domínio, organizar os requisitos de forma a atribuir responsabilidades, identificar novos requisitos, os conflitos entre eles, entre outros. No âmbito da dissertação apenas vão ser detalhados os elementos principais do modelo de objectivos (i.e., agentes e objectivos).

Com o modelo de objectivos é possível identificar os *goals* através dos refinamentos AND e OR que satisfaçam o objectivo principal. Os elementos principais são: requisitos, expectativas, obstáculos, objectos, operacionalizações, agentes e propriedade de domínio. Um **requisito** especifica de forma concreta um objectivo, atribuído a um agente do sistema, localizado por norma como folha de um modelo KAOS. Uma **expectativa** é um objectivo atribuído a um agente do ambiente. Os **obstáculos** ocorrem quando um objectivo não pode ser alcançado numa dada circunstância. Estes obstáculos devem ser decompostos de modo arranjar uma solução através da criação de novos requisitos. As **operacionalizações** são baseadas em requisitos e nas responsabilidades atribuídas aos

agentes, de forma a satisfazer o objectivo. Os **agentes** podem ser categorizados através de dois tipos: **agente de ambiente**, que se responsabiliza por expectativas, e **agente de sistema**, que se responsabiliza por requisitos. As **propriedades do domínio** representam características que o objectivo contém do domínio da aplicação.

As relações que existem entre elementos são definidas através do refinamento OR, refinamento AND, ligação de responsabilidade, ligação de operacionalização, ligação de obstáculo, resolução, *concern*, conflito e execução (performance). O **refinamento OR** tem como propósito refinar um objectivo em requisitos ou expectativas que só serão satisfeitos se pelo menos um dos elementos que o decompõem for satisfeito. O **refinamento AND** funciona da mesma forma, excepto que todos os elementos têm que ser satisfeitos para que o objectivo principal possa ser cumprido. A **ligação de responsabilidade** atribui a responsabilidade de um requisito ou expectativa a um agente do sistema ou de ambiente. A **ligação de operacionalização** atribui uma operação a um requisito. A **ligação de obstáculo** faz a ligação de um obstáculo com um objectivo, requisito ou expectativa e uma **ligação de resolução** corresponde à ligação de um obstáculo que é resolvido por um objectivo/requisito, sendo representado como resolução. Uma **ligação concern** conecta um requisito a uma entidade para que o requisito seja satisfeito e um **conflito** ocorre quando dois ou mais objectivos não podem ser satisfeitos em simultâneo. Por norma são aplicáveis em requisitos não funcionais tais como segurança, desempenho e fiabilidade.

Na Figura 2.2 é apresentado um exemplo de um ginásio estruturado num modelo orientado a objectivos.

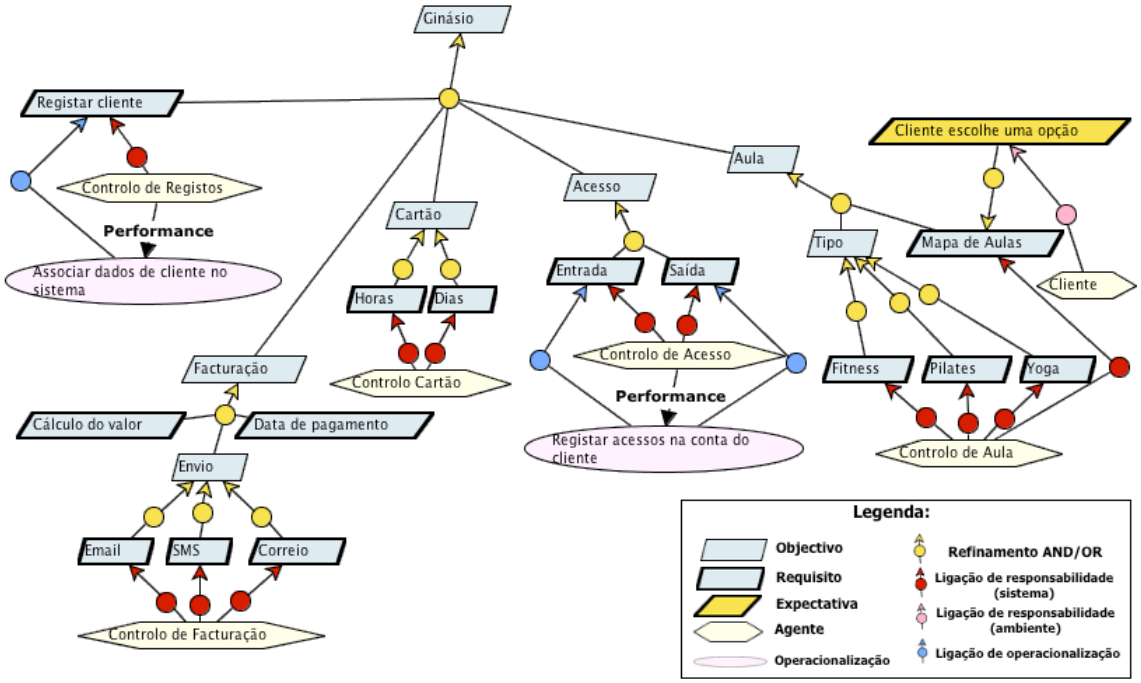


Figura 2.2: Representação do sistema de um Ginásio em KAOS.

O **objectivo** "Ginásio" descreve o comportamento de forma generalizada com o intuito de alcançar a requisitos do sistema. O "registar cliente" é um **requisito** que especifica de forma concreta o **objectivo** "Ginásio". A **expectativa** é um requisito que o agente assume que seja executado, identificado a amarelo na figura como "Cliente escolhe uma opção". O "Cliente" é representado como um **agente de ambiente** e o "Controlo de facturação" como **agente de sistema**. As **operacionalizações** são baseadas em requisitos e nas responsabilidades atribuídas aos agentes de forma a satisfazer o objectivo. O elemento "Registar acessos na conta do cliente" é uma operacionalização para os requisitos "Entrada" e "Saída".

i* framework

O **i*** é uma ferramenta de modelação orientada a agentes [Lap05] constituída por dois modelos: **S***trategic* **D***ependency* (SD) e **S***trategic* **R***ationale* (SR). Esta abordagem não vai ser integrada nos modelos de análise a ser utilizados nesta dissertação.

O modelo SD é composto por uma rede cujos nós são os actores e as relações são as dependências, dado um contexto organizacional. O objectivo é capturar os objectivos que cada processo tem dentro da organização e definir a importância dos participantes de uma forma abstracta. Os **actores** podem ser definidos como seres humanos ou sistemas de *hardware/software*. As dependências são relações que ajudam a conectar actores com **objectivos**, **requisitos não funcionais**, **tarefas** - acções a serem executadas pelos actores e **recursos** - entidades a serem usadas no sistema.

O modelo SR detalha o modelo SD, em cada actor, com mais objectivos, tarefas, requisitos não funcionais e recursos. Neste modelo são usadas relações adicionais, comparativamente ao SD, tais como: ***means-end***, que estabelece uma ligação de uma tarefa com um objectivo, e a **decomposição** que particulariza uma tarefa em mais estados, mais concretamente em subtarefas, sub-objectivos, recursos e requisitos não funcionais.

NFR Framework

N*on* **F***unctional* **R***equirements Framework* é um modelo de análise de requisitos não funcionais [Lap05], representado em forma de grafo. Por ser um modelo *goal-oriented*, o objectivo da modelação é realizar refinamentos sucessivos para chegar a uma operacionalização do sistema e realizar uma análise do impacto das escolhas feitas.

Os *softgoals* presentes no grafo são apresentados como requisitos não funcionais (e.g., segurança, desempenho, concorrência); *softgoals* de operacionalização que definem possíveis soluções ou alternativas ao requisito não funcional a ser satisfeito; e *softgoals* de argumentação que explicam o contexto de um *softgoal* ou interdependência.

As interdependências, que podem ser explícitas ou implícitas, refinam *softgoals* através de condições lógicas AND e OR ou então através de contribuições parcialmente positivas (**+**), positivas (**++**), parcialmente negativas (**-**) e negativas (**- -**).

2.3.2 Engenharia de Requisitos Orientada a Objectos

Outra metodologia que visa a colaboração na elicitação de requisitos é a Engenharia de Requisitos Orientada a Objectos. Esta abordagem contribui com muitos benefícios para a implementação do sistema a ser desenvolvido usando o paradigma orientado a objectos, tais como [DS99]: auxiliar a compreensão de um problema através do uso de objectos analisando o seu comportamento, as suas propriedades, a relação entre objectos, a facilidade de modificar o modelo, a modularização e reutilização.

A modelação conceptual especifica o domínio de um dado problema através da representação dos objectos/entidades. Num contexto orientado a objectos, um objecto/entidade de um sistema é associado a um objecto/ideia do mundo real. O Modelo Conceptual é constituído por entidades, atributos e o relacionamento que existe entre as entidades. Uma **entidade** representa um objecto, como instância de uma classe, através de um conjunto de **atributos**, que definem o estado da classe, e o seu comportamento dado as suas operações. As **relações** definem conexões entre os objectos visto que num sistema os objectos nunca ficam isolados.

Nesta metodologia é usado a **U***nified* **M***odelling* **L***anguage* (UML). O UML é uma linguagem de modelação orientada para o desenvolvimento de *software*, criada por Grady Booch, Ivar Jacobson e Jim Rumbaugh. Mais tarde começou a ser mantida pela **O***bject* **M***anagement* **G***roup* (OMG). Existem vários diagramas definidos pela OMG, mas neste contexto, o Modelo Conceptual é representado através do diagrama de classes com o auxílio da ferramenta MagicDraw [Mag14] para criar modelos. Através deste diagrama é possível visualizar como está estruturado o sistema num paradigma orientado a objectos [LBD02]. Tem como objectivo detalhar cada objecto com os seus atributos e comportamento e também as relações entre os objectos.

A Figura 2.3 ilustra um exemplo de um modelo conceptual de um ginásio, seguindo o exemplo ginásio do modelo KAOS (Figura 2.2).

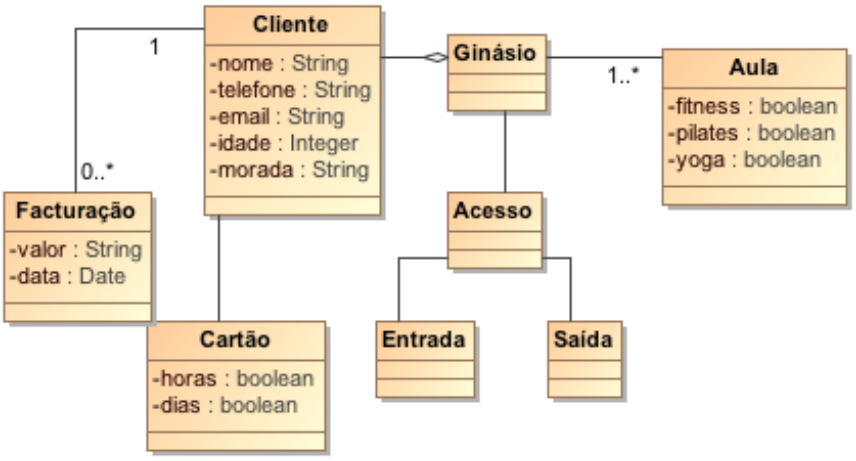


Figura 2.3: Modelo Conceptual do sistema de um Ginásio.

O "Cliente" é representado através de uma entidade, composta por vários atributos: "nome", "telefone", "*email*", "idade" e "morada". A associação e as suas multiplicidades são representadas como está especificado entre as entidades "Cliente" e "Facturação". A agregação está definida entre as entidades "Ginásio" e "Cliente".

2.3.3 Modelo de Variabilidade

O Modelo de Variabilidade tem como objectivo modelar as variabilidades de um domínio que podem gerar uma linha de produtos. Nesta secção é discutido o Modelo de *Features* [Kan+90], que pode ser construído através da ferramenta FeatureIDE [Kas+05]. Uma *feature* é definida como um aspecto, uma característica do domínio ou uma qualidade num sistema de *software* [Asa+12; Kan+90].

As *features* estão organizadas num diagrama, mais concretamente através de uma árvore, onde na raiz há o produto que se pretende desenvolver e os seus descendentes são constituídos por *features*. É possível refinar *features* através de outras *features*, estabelecendo, desta forma, uma hierarquia. Na Figura 2.4 é apresentado um exemplo de um ginásio através de um Modelo de *Features*, com as mesmas funcionalidades do exemplo do modelo KAOS (Figura 2.2) e modelo conceptual (Figura 2.3).

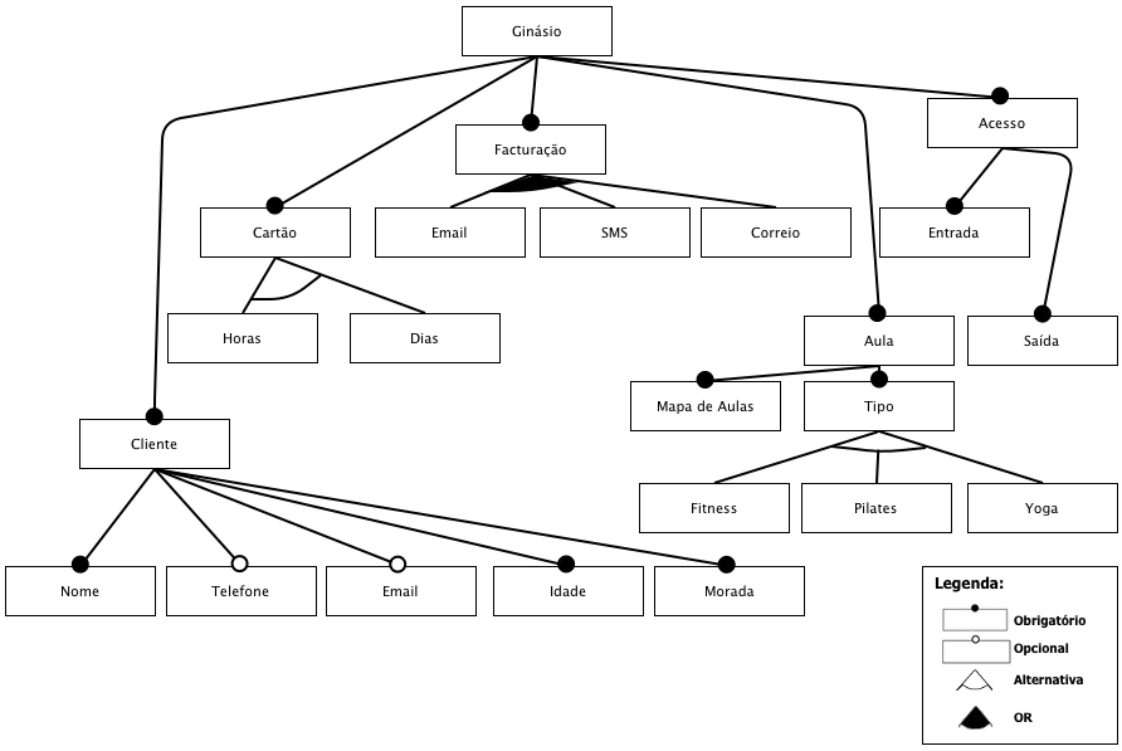


Figura 2.4: Modelo de Features do Ginásio

A *feature* "Cliente" é representada como **obrigatória**, ou seja, um modelo que tenha essa representação significa que a *feature* tem que ser aplicada. A propriedade da *feature* "Telefone" é definida como **opcional**, podendo ou não, ser aplicada no sistema. Na *feature* "Facturação" está representada a decomposição **OR**, cuja semântica refere que pelo

menos uma das *sub-features* precisa de ser seleccionada caso a *feature* raiz seja aplicada. A decomposição **XOR**, utilizada pela *feature* "Cartão", indica que uma das *sub-features* tem que ser seleccionada caso a *feature* raiz seja aplicada.

2.4 Sumário

Neste capítulo foi discutido o conceito de Engenharia de Requisitos onde, em geral, nenhuma delas usa ferramentas ou estratégias de acessibilidade a membros da equipa de desenvolvimento com deficiências. Foram apresentados a abordagem KAOS, *i* framework*, NFR *framework*, o modelo conceptual e o modelo de *features*. As *frameworks* de modelação Objectiver, MagicDraw e FeatureIDE são ferramentas CASE muito populares para cada modelo, mas que não usam estratégias de acessibilidade. O próximo capítulo apresenta os conceitos de reconhecimento e sintetização de voz, as tecnologias a serem aplicadas na dissertação e os conceitos de gramática que vão permitir derivar os modelos de requisitos.

3

Reconhecimento e Sintetização de Voz

3.1 Contexto

A fala é a forma mais fácil e comum de comunicação verbal entre humanos. No mundo da ciência tem havido, durante os últimos quarenta anos, pesquisas com o intuito de melhorar a interacção homem-máquina, desenvolvendo sistemas automáticos de reconhecimento de voz [End01]. As primeiras tentativas para desenvolver estes sistemas automáticos, apontam para 1950, focadas na acústica da voz. Em 1952, K. H. Davis et al. [DBB52] desenvolveram nos laboratórios Bell, um sistema para o reconhecimento de um dígito isolado. Os investigadores Fry e Denes da University College of England nos finais da década de 50 tentaram construir um reconhecimento de fonemas[1] para vogais e consoantes, usando um analisador de espectro e uma combinação de padrões. Desde então que foram realizados vários melhoramentos e, graças ao desenvolvimento de algoritmos mais sofisticados, já é possível a interpretação de palavras isoladas e o reconhecimento de fala espontânea contínua [Vaz+12].

Em relação à sintetização de voz, os primeiros esforços foram feitos com sistemas mecânicos em 1779 [Lem99], quando Christian Kratzenstein explicou as diferenças fisiológicas entre as cinco vogais longas (“a”, “e”, “i”, “o” e “u”) e construiu ressonadores acústicos baseados no trato vocal humano para produzir as cinco vogais. Em 1791, Wolfgang von Kempelen criou um mecanismo capaz de produzir palavras e até frases inteiras, denominada “Acoustic-Mechanical Speech Machine” [Lem99; Von91]. Em meados

[1] Fonema: É o menor elemento de fala de uma língua.

de 1800, Charles Wheatstone construiu uma versão mais complexa em relação à máquina de von Kempelen, sendo que esta produzia mais combinações de sons [Lem99].

Homer Dudley desenvolveu o **V***oice* **O***perating* **D***emonstrator* (VODER), em 1939, que foi considerado o primeiro dispositivo de sintetização de fala electrónico [Lem99]. Este aparelho foi desenvolvido nos laboratórios Bell. O primeiro sistema completo *text-to-speech* para a língua inglesa foi criado em 1968 por Noriko Umeda. O som era bastante inteligível e distante dos sistemas actuais, que progrediram bastante [Lem99].

3.2 Processos dos mecanismos

Nas próximas subsecções vão ser descritas, muito sucintamente, os processos que permitem transformar as expressões orais em texto, na subsecção 3.2.1, e texto em som, apresentado na subsecção 3.2.2.

3.2.1 Processo de Reconhecimento de Voz

O processo de reconhecimento de voz envolve três passos principais: captura da voz, o seu processamento e a tradução para texto. Na Figura 3.1 é mostrado um diagrama em bloco do respectivo processo da tradução de voz para texto.

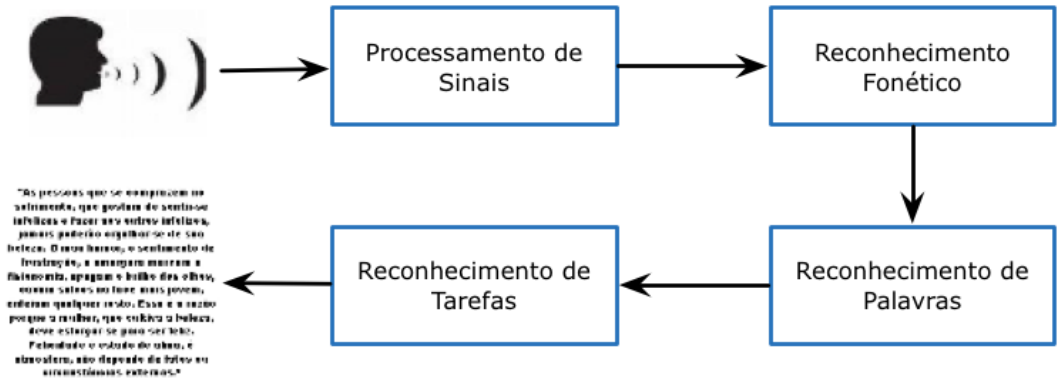


Figura 3.1: Diagrama em bloco de um reconhecedor de voz adaptado [PR96].

A voz é mapeada para uma sequência de *feature vectors* para analisar as frequências auditivas. De seguida, como exemplo, através de **H***idden* **M***arkov* **M***odel* (HMM) ou Redes Neuronais, os padrões do espectro são comparados com os padrões dos fonemas da língua que está a ser reconhecida.

Com uma estrutura lexical e reconhecimento de palavras a sequência de fonemas é comparada a uma lista de sequência de hipótese de palavras. Na última etapa, através de uma gramática, as palavras reconhecidas na fase anterior são transformadas em texto, completando as fases para transformar a voz em texto [PR96].

3.2.2 Processo de Sintetização de Voz

Na Figura 3.2 é apresentado um diagrama muito geral do mecanismo usado para a sintetização da voz. Em primeiro lugar é apresentado o módulo de processamento de linguagem natural que, com base no texto que recebe como *input*, produz uma transcrição fonética, a entonação e o ritmo pretendido do respectivo texto. Com o módulo de processamento de sinal digital transforma-se a informação recebida pelo *output* oferecido pelo módulo anterior em discurso em som [Dut97].

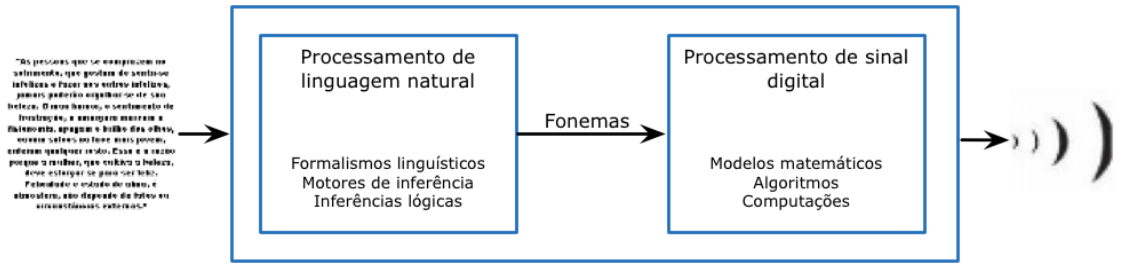


Figura 3.2: Diagrama funcional de um sistema de sintetização de voz [Dut97].

3.3 Vantagens e Desvantagens

Os mecanismos de reconhecimento de voz proporcionam inúmeras vantagens. Um dos grandes benefícios é possibilitar que estudantes e trabalhadores com deficiências físicas possam realizar as suas tarefas em dispositivos electrónicos. Os computadores desempenham cada vez mais um papel decisivo na educação, e graças ao reconhecimento de voz, os alunos com limitações podem interagir com estes dispositivos com um menor grau de dificuldade [wis14]. Também permite oferecer mais oportunidades na área do trabalho [wis14], pois existem muitos indivíduos talentosos mas que são afectados por limitações físicas. O reconhecimento da fala pode ajudar a contornar esse tipo de situações, permitindo que estes indivíduos possam dar a sua contribuição. Para além de ajudar a que pessoas com limitações possam aprender e contribuir para o mundo do trabalho e educação, a utilização das técnicas de reconhecimento de voz automatiza funções, de forma a proporcionar maior rapidez nas tarefas das aplicações, a comunicação com dispositivos quando não há a possibilidade de o usar com as mãos (por exemplo, no acto de conduzir), ou pelo controlo remoto de sistemas através das telecomunicações.

Todavia a tecnologia ainda não é muito utilizada devido a uma série de limitações. O ruído pode interferir com o processo de reconhecimento de voz, não produzindo o resultado expectável. Também pode ocorrer uma tradução errada de palavras que são muito semelhantes foneticamente como por exemplo as palavras *right* e *write*. O vocabulário é um componente muito importante no reconhecimento de voz, influenciando a *performance* do mecanismo de reconhecimento de voz e na ocorrência de falsos positivos na tradução para texto [Uni96]. Se um vocabulário estiver bem definido, o risco de ocorrerem falsos positivos diminui, embora muitas vezes implique um grande registo de

entradas na base de dados do vocabulário, podendo provocar uma diminuição no tempo de resposta da aplicação.

Os sistemas de sintetização de voz permitem escalabilidade porque cada palavra é gerada em tempo real e não necessita de ter o discurso previamente gravado em disco [Dir04], permitem extensibilidade com a possibilidade de acrescentar novas palavras com base na adição de uma transcrição fonética e personalização [Dir04] e a costumização possibilita ao utilizador a definição nas preferências da voz, tais como o género, *pitch*, velocidade da fala ou se pretendem que o modo de fala seja formal ou informal [Dir04]. No entanto, muitos dos mecanismos contêm falas com som pouco natural (robóticas) e, para melhorar esse processo, são necessários ter à disposição um armazenamento de dados e processamento mais consideráveis [Pro14].

3.4 Aplicações

O uso de mecanismo de reconhecimento de voz permite ao utilizador implementar outras formas de interacção com dispositivos para que não seja necessário usar as mãos, e por sua vez, melhorar a eficiência da transição de tarefas em relação ao método tradicional. Uma das aplicações foi a criação de uma interface de reconhecimento de voz para manipular o dispositivo móvel enquanto o utilizador está a conduzir desenvolvido por Zhanh Hua e Wei Leih Ng [HN10]. O sistema permite realizar chamadas, aceder a contactos e ouvir mensagens, melhorando a acessibilidade da aplicação. Tony Ayres e Brian Nolan [AN04] realizaram uma pesquisa sobre um dispositivo robótico remoto controlado através do reconhecimento de voz via rede Wi-Fi[2], WLAN[3] e TCP/IP[4], usando a linguagem de programação Java.

No contexto da Engenharia de *Software* existe um trabalho desenvolvido por Tavis Rudd que permite programar em Python via voz [Rud13]. Este trabalho foi implementado em *Dragon NaturallySpeaking*[5] em ambiente Windows no qual criou comandos de voz para melhorar a acessibilidade da escrita de código, criação de atalhos, situar o cursor para uma posição específica do código, entre outros. Com este trabalho é possível na fase de implementação de um sistema de *software* programar sem necessidade de escrever no teclado, contribuindo para uma maior produtividade visto que em situações normais o ser humano é capaz de ditar textos mais rápido do que escreve.

No capítulo de Trabalhos Relacionados (capítulo 5) estão relatados mais trabalhos dentro do contexto de Engenharia de *Software* que utilizam mecanismos de reconhecimento de voz.

[2] **W***ireless* **F***idelity*: Tecnologia que permite aceder a uma rede sem acesso a fios.
[3] **W***ireless* **L***ocal* **A***rea* **N***etwork*: Acesso a uma rede sem fios usando ondas de rádio para transmitir os dados.
[4] **T***ransmission* **C***ontrol* **P***rotocol*/**I***nternet* **P***rotocol*: Conjunto de protocolos de comunicação entre computadores.
[5] *Dragon NaturallySpeaking*: `http://www.nuance.com/dragon/index.htm` (Acesso feito em Setembro de 2014)

3.5 Tecnologias

Actualmente as tecnologias desenvolvidas permitem capturar a voz através do microfone como dispositivo de entrada de dados e sintetizar voz para os periféricos de som. Existem inúmeras API's (**A***pplication* **P***rogramming* **I***nterface*) de reconhecimento e sintetização de voz como por exemplo a **M***icrosoft* **S***peech* API[6] (SAPI) , mas nesta dissertação vai ser utilizado o **J***ava* **S***peech* API (JSAPI). A escolha foi atribuída para o JSAPI porque a SAPI utiliza **C***omponent* **O***bject* **M***odel*[7] (COM), não oferecendo a possibilidade de desenvolvimento em qualquer ambiente. O JSAPI vai ser a interface usada na aplicação porque permite a portabilidade dado que foi implementada na linguagem de programação Java.

Também vai ser integrada a ferramenta Google *Speech* API[8]. É uma aplicação desenvolvida em Javascript pela Google que permite reconhecer voz. Através desta aplicação é possível traduzir voz para texto através de uma grande variedade de idiomas com uma boa taxa de *accuracy* na tradução do texto com um tempo de resposta bastante bom. Esta API vai ser usada em conjunto com a tecnologia que usa os serviços do JSAPI, de forma remota, para ter acesso à sua base de dados no que diz respeito ao vocabulário e taxa de *accuracy* bastante elevada.

3.5.1 Java Speech API

J*ava* **S***peech* API (JSAPI) permite criar aplicações com reconhecimento e sintetizadores de voz através de uma interface fornecida pela JSAPI [Ora14]. Houve várias empresas envolvidas neste desenvolvimento, tais como: Sun Microsystems; Apple Computer, Inc.; AT&T; Dragon Systems, Inc.; IBM Corporation; Novell, Inc.; Philips; Speech Processing and Texas Instruments Inc. Existem vários tipos de implementação que utilizam esta API, sendo os mais relevantes:

- **IBM's Speech for Java [Ora14]**: interpreta fala contínua, por comando e controlo, e sintetizador de voz suportando os idiomas Inglês, Francês, Alemão, Italiano, Espanhol e Japonês. É baseado no IBM's Via Voice, produto comercial lançado em 2005 nas plataformas Windows e OS X;
- **Sphinx-4 [Sph14]**: ferramenta desenvolvida pela Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Electric Research Labs (MERL) e Hewlett Packard (HP). Implementado em Java, é capaz de reconhecer voz discreta e contínua usando HMM. Esta ferramenta será a API de reconhecimento de voz em conjunto com o Java Speech API utilizada na dissertação porque não necessita de uma licença para utilizar a aplicação;

[6] **M***icrosoft* **S***peech* API: http://msdn.microsoft.com/en-us/library/ee125663(v=vs.85).aspx (Acesso feito em Setembro de 2014)

[7] **C***omponent* **O***bject* **M***odel*: Permite desenvolver componentes usando a plataforma Windows.

[8] Google *Speech* API: https://www.google.com/intl/pt/chrome/demos/speech.html (Acesso feito em Setembro de 2014)

- **FreeTTS [WK05]**: sintetizador de voz desenvolvido pelo grupo Speech Integration da Sun Microsystems Laboratories. É um sistema desenvolvido em Java com suporte apenas para a língua inglesa. Com o FreeTTS é possível: usar a especificação fornecida pelo sintetizador da JSAPI; serviço automático de sistema de telefone (e.g., apoio ao cliente de uma operadora móvel); ou para o uso em uma *workstation/desktop*. Será incluído na ferramenta com o objectivo de fornecer *feedback*.

As ferramentas Sphinx-4 e FreeTTS vão ser descritas com mais detalhe nas subsecções 3.5.2 e 3.5.3. Estas *frameworks* vão possibilitar a integração de reconhecimento e sintetização de voz no sistema interno, sendo que o Google Speech API vai ser usado remotamente.

3.5.2 Sphinx-4

Sphinx-4 [Wal+04] é uma ferramenta com uma estrutura modelar e que permite conectar de forma simples os componentes, incorporando normas de sistemas já existentes, como por exemplo Java Speech API, com flexibilidade suficiente para suportar áreas emergentes de pesquisa. A modulação permite que os módulos se dediquem a tarefas muito específicas e facilmente substituíveis em tempo de execução. A Figura 3.3 mostra os componentes do sistema Sphinx-4.

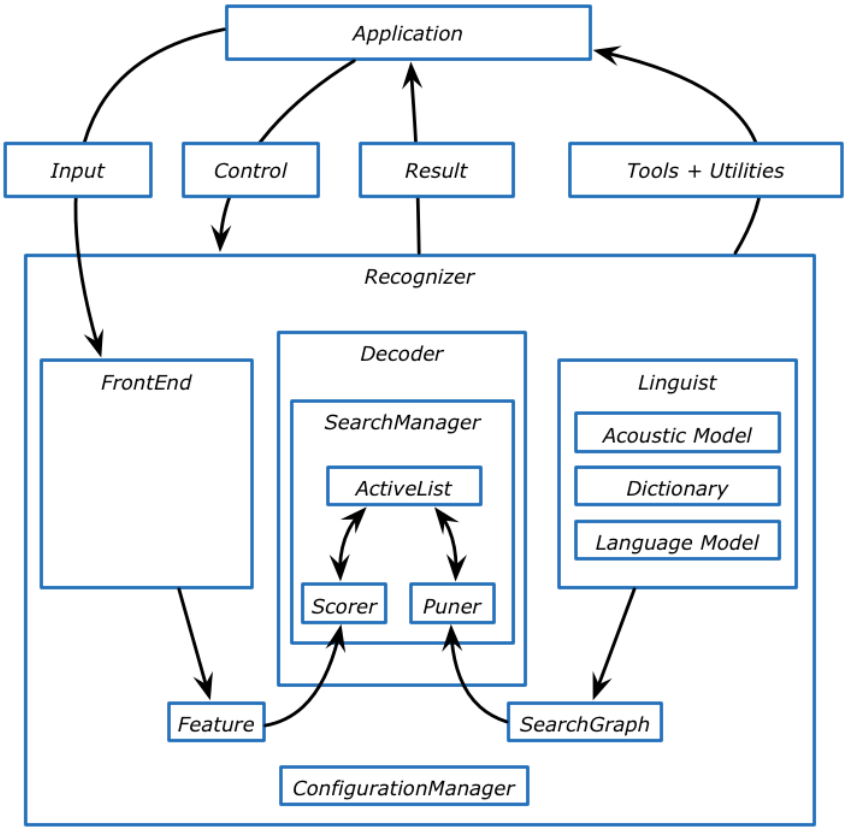


Figura 3.3: Diagrama dos componentes Sphinx-4 [Wal+04].

O Sphinx-4 é composto por três componentes principais: *FrontEnd*, Linguística e Descodificador. O **FrontEnd** tem como objectivo fazer a parametrização de sinais, como por exemplo áudio, num conjunto de caracteres. Este módulo possibilita a computação de várias parametrizações do mesmo ou de diferentes sinais de entrada. O módulo **linguístico** tem como objectivos principais a geração do *SearchGraph* e abstracção da complexidade da geração do grafo, com a possibilidade de configurar dinamicamente os subcomponentes **Modelo Acústico**, **Dicionário** e **Modelo de Linguagem**. O Modelo Acústico mapeia um conjunto de unidades de fala e o HMM, com a finalidade de construir o *SearchGraph* usando uma estrutura definida pelo Modelo de Linguagem. Ao usar o componente Dicionário, é possível transformar palavras do Modelo de Linguagem em sequências de elementos do Modelo Acústico. O Modelo de Linguagem permite definir uma estrutura de uma linguagem através de gramáticas[9], tais como *SimpleWordListGrammar*, *JSGFGrammar*, *LMGrammar*, *FSTGrammar*, *SimpleNGramModel*, *LargeTrigramModel*. No contexto deste trabalho vai ser usado o *JSGFGrammar* para estruturar a linguagem de cada modelo de análise porque tem um bom suporte em relação à documentação e é recomendada pela W3C. Com o conjunto de caracteres provenientes do *FrontEnd* e em conjunto com o *SearchGraph* é gerado um resultado provável oriundo da voz graças ao componente **Descodificador**.

3.5.3 FreeTTS

FreeTTS [WLK02] é uma ferramenta que recebe como entrada um bloco de texto e divide num conjunto de fonemas, realizando posteriormente transformações de modo a que esses fonemas sejam convertidos em som. A Figura 3.4 apresenta a arquitectura da ferramenta FreeTTS.

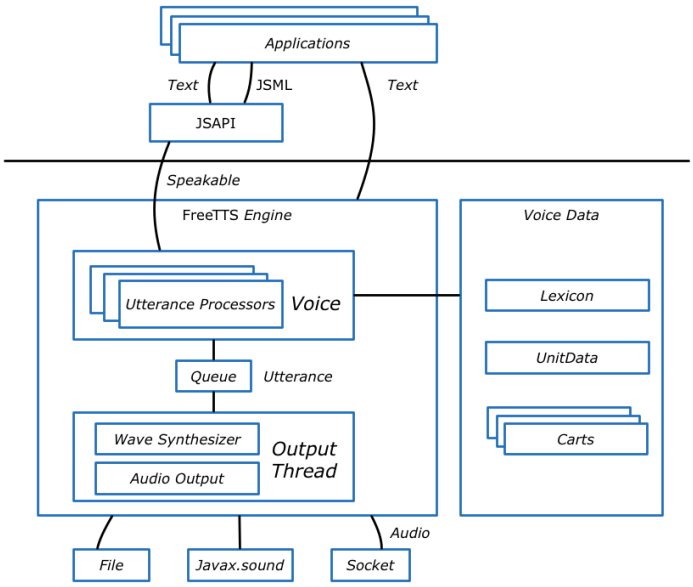


Figura 3.4: Arquitectura FreeTTS [WLK02].

[9]Gramática: Conjunto de regras baseadas em palavras-chave de forma a desencadear uma acção.

Os módulos principais da arquitectura FreeTTS são *voice* e *output thread*. O módulo **voice** é composto por um conjunto de *utterance*[10] *processors*, que é responsável por criar, processar e anotar uma estrutura de *utterance*. Também mantém a informação global do processo, tais como o local, a pronúncia lexical, o banco de dados da unidade e o sintetizador da onda. Por cada bloco de áudio que vai ser produzido, é criada uma estrutura de *utterance*. Estas estruturas são inicializadas quando existe um bloco de texto para ser reproduzido e depois são transferidas para um conjunto de *utterance processors*. Depois de o texto ter sido reproduzido para um canal de áudio, o sistema elimina o objecto das estruturas. Os *utterance processors* permitem adicionar itens para uma estrutura *utterance* para que fique especificada de forma hierárquica e relacional. Também possibilita a criação da relação de itens que descrevem as sílabas das palavras, para cada sílaba, apontando para palavras individuais que tenham sido criadas por outros *utterance processors*.

O módulo **output thread** é responsável por sintetizar uma *utterance* para áudio e direccionar essa informação para o canal apropriado, seja um ficheiro de áudio, para a biblioteca *Javax.sound* ou para um *socket* de áudio. No contexto da dissertação os blocos de textos são automaticamente reproduzidos para o canal de áudio que está ligado directamente à saída do som, sem necessitar de usar ficheiros de áudio como intermediários.

3.5.4 Gramática

Existem dois tipos de gramáticas usados no JASPI para realizar o reconhecimento de voz [End01]: regras, para usar como comando de modo a desencadear uma acção, e ditado, para especificar o conteúdo de um elemento. As gramáticas de regras permitem restringir as palavras, provenientes da fala, que vão ser processadas de acordo com as regras que são definidas. Por outro lado, as gramáticas de ditado permitem que esse tipo de limitação sejam um pouco mais abrangente transformando em texto tudo o que foi ditado no discurso do utilizador. Este tipo de gramática traduz-se numa maior complexidade em relação às gramáticas de regras visto que é menos precisa devido ao constante processamento das palavras que são ditadas.

A aplicação Sphinx-4 suporta os dois tipos definidos, mas no modo de ditado pode-se usar o vocabulário por defeito da ferramenta, ou então, restringir o domínio da linguagem com o objectivo de obter melhores resultados. Essa restrição passa por definir um modelo de linguagem específico [CMU13] escrevendo no ficheiro de texto as palavras do domínio e com a ferramenta Sphinx Knowledge Base Tool[11] é gerado um ficheiro com o novo dicionário e modelo de linguagem. Na próxima subsecção são descritas as regras de uma gramática que vai permitir estruturar um modelo de requisitos.

[10] *Utterance*: representa o que o utilizador diz, designado como expressão oral.

[11] Sphinx Knowledge Base Tool: http://www.speech.cs.cmu.edu/tools/lmtool-new.html (Acesso feito em Setembro de 2014)

3.5.5 Java Speech Grammar Format

J*ava* **S***peech* **G***rammar* **F***ormat* (JSGF) [End01] é uma gramática que permite definir regras usando os padrões da JSGF. O formato usado é o **B***ackus* **N***aur* **F***orm*[12] (BNF), definindo, desta forma, uma gramática com um conjunto de regras usando uma representação textual. Em seguida vão ser apresentadas as propriedades das regras do JSGF disponibilizado pela W3C [W3C00].

Definição. As regras são combinações de texto que podem apontar para outras regras. Cada regra tem um nome único e a sua sintaxe é definida através dos caracteres "⟨⟩". Um exemplo de uma regra é ⟨*ola*⟩ ou ⟨*apanhar*⟩.

Para definir na gramática o que é que um utilizador falou é usado o conceito símbolo que corresponde muitas vezes a uma palavra ou a uma sequência de palavras. É possível definir um símbolo como "*ola*" ou "*abrir a porta*".

Composição. Uma regra tipicamente é composta por regra(s) e/ou símbolo(s), delimitada por espaços brancos e terminada com um ponto e vírgula. Apesar de serem os componentes base, existem várias possibilidades de definir a composição de uma regra. Uma forma de compor uma regra é através de **sequências** de expansões. Uma aplicação para este contexto é:

⟨curso⟩ = Eu estou a estudar Engenharia Informática;
⟨objecto⟩ = Aquele objecto é para ⟨accao⟩;

Para activar esta sequência é necessário que seja ditada pela ordem definida na regra. Deve-se dizer primeiro a regra "curso", através das palavras, "*Eu estou a estudar Engenharia Informática*". Na regra "objecto" é necessário dizer exactamente "*Aquele objecto é para*" seguido por algo que respeite a regra ⟨accao⟩.

Outro tipo de composição são as **alternativas** entre símbolos e regras dentro de uma regra separadas pelo símbolo "|". Um conjunto de alternativa pode ser exemplificado da seguinte forma:

⟨chamada⟩ = atender | rejeitar | silêncio;

Para satisfazer a regra "chamada" deve-se dizer exactamente uma das seguintes palavras: "*atender*", "*rejeitar*" ou "*silêncio*".

A composição por sequência tem maior precedência em relação às alternativas. No topo das precedências há a composição por **agrupamento** que tem como objectivo garantir que numa regra é possível definir um conjunto possível de resultados. A sua sintaxe é definida através dos parênteses:

[12] **B***ackus Naur Form*: sintaxe usada para expressar gramáticas livres de contexto.

⟨execucao⟩ = (abrir | fechar) (porta | janela | cancela | computador);

Uma instância correcta desta regra é "*abrir porta*" ou então "*fechar cancela*", entre muitas outras combinações possíveis. Outro tipo de agrupamentos são os **opcionais** com a sintaxe "*[⟨regra⟩ ou símbolo]*".

Cada regra/símbolo pode ter um peso dentro da composição quando é conhecido a probabilidade de pronunciar uma determinada palavra numa regra. A atribuição de pesos é feita através da sintaxe "*/número/*" (sendo o número pertencente ao conjunto dos reais). Um exemplo de atribuição de pesos numa regra é:

⟨execucao⟩ = abrir (/1/ porta | /4/ janela | /7 /cancela | /20/ computador);

Operadores unários. A definição de multiplicidades é outra das características que podem ser implementadas no corpo das regras. Através do operador "*" é dito que uma regra ou símbolo é mencionada zero ou mais vezes e através do operador "+" essa regra/símbolo é accionada uma ou mais vezes.

Tags. O uso de *tags* permite associar informações específicas a uma regra na gramática de modo a simplificar ou melhorar o processamento dos resultados, tendo maior precedência em relação às sequências e alternativas. A sua sintaxe é expressa através das chavetas "{}" como mostra o seguinte exemplo:

⟨cumprimento⟩ = ola {hello};

Recursividade. Outro conceito fundamental na construção de uma gramática é através de **recursividade**. No exemplo a seguir, as instâncias "*porta*" ou "*porta e cancela e janela*" são válidas para as regras definidas em cima:

⟨comando⟩ = ⟨objecto⟩ | (⟨objecto⟩ e ⟨comando⟩);
⟨objecto⟩ = porta | janela | cancela | computador;

3.6 Sumário

Neste capítulo foram introduzidos alguns conceitos sobre reconhecimento e sintetização de voz. A ferramenta Sphinx-4 e FreeTTS foram aprofundadas em maior detalhe devido à sua importância no contexto da dissertação. Por fim, foram apresentadas as propriedades que permitem construir uma gramática. O próximo capítulo apresenta os conceitos no Desenvolvimento Orientado a Modelos, que vai permitir integrar o sistema de reconhecimento de voz para derivar modelos de requisitos explicados no capítulo 2.

4

Desenvolvimento Orientado a Modelos

4.1 Conceitos

M*odel*-**D***riven* **D***evelopment* (MDD) é uma abordagem onde os modelos são os artefactos principais durante o desenvolvimento do *software*, com o intuito de aumentar o nível de abstracção e simplificar os detalhes da resolução do domínio de um problema. Mellor et al. [Sch06] definem o MDD como a maneira de construir um modelo de um sistema e transformá-lo em algo real. Pode-se definir um modelo [BG01] como uma representação abstracta da construção de um *software*, com o objectivo de responder a perguntas no lugar do sistema. É composto por elementos que representam conceitos do sistema e ligações entre esses elementos.

Um subconjunto do MDD é a Arquitectura Orientada por Modelos (**M***odel*-**D***riven* **A***rchitecture* - MDA) que foi a primeira investida no que diz respeito a uma abordagem orientada a modelos proposta pela OMG. Os modelos do MDA são [Mel+02]: modelo de computação independente que descreve como é que um sistema se deve comportar (**C***omputation* **I***ndependent* **M***odel* - CIM), modelo independente da plataforma em que é construído (**P***lataform*-**I***ndependent* **M***odel* - PIM), modelos específicos da plataforma (**P***lataform*-**S***pecific* **M***odel* - PSM) e, através de cada PSM, gerar código.

A definição dos elementos de um dado domínio é feita com base numa linguagem de modelação. Para a linguagem estar bem especificada, necessita de uma sintaxe e semântica bem definidas. A sintaxe de uma linguagem é dividida em duas características: sintaxe abstracta e concreta. A **sintaxe abstracta** define a estrutura de uma linguagem,

nomeadamente os seus conceitos e como eles se relacionam (e.g., subconjunto do modelo Ecore da Figura 4.3). A **sintaxe concreta** define como vai ser a estrutura externa de uma linguagem, transformando-a numa sintaxe textual e/ou gráfica. A sintaxe textual é estruturada através de uma gramática (e.g, Emfatic ou Xtext) e a sintaxe gráfica usando, por exemplo, o **G***raphical* **M***odeling* **F***ramework* (GMF) (citado na subsecção 4.5.3).

A **semântica estática** é representada como um conjunto de regras para que os modelos sejam válidos (e.g., EVL - citado na subsecção 4.5.2) e a **semântica dinâmica** como a composição de dois componentes: o **mapeamento semântico** estabelece a ponte entre a sintaxe abstracta e o significado de uma linguagem, denominado por **domínio semântico**.

4.2 MetaModelo e Meta-MetaModelo

Um metamodelo é um modelo que define uma linguagem de modelagem, isto é, representa os conceitos definidos pela linguagem e também a relação entre eles. Pode-se afirmar que, um metamodelo é a representação da sintaxe abstracta em notação gráfica e também da semântica estática, definindo restrições e removendo ambiguidades que um metamodelo com a sintaxe abstracta não consegue especificar.

Um meta-metamodelo tem como propósito definir um metamodelo tal como o modelo é definido pelo metamodelo. Na Figura 4.1 é apresentada a arquitectura de camadas da **M***eta* **O***ject* **F***acility* (MOF).

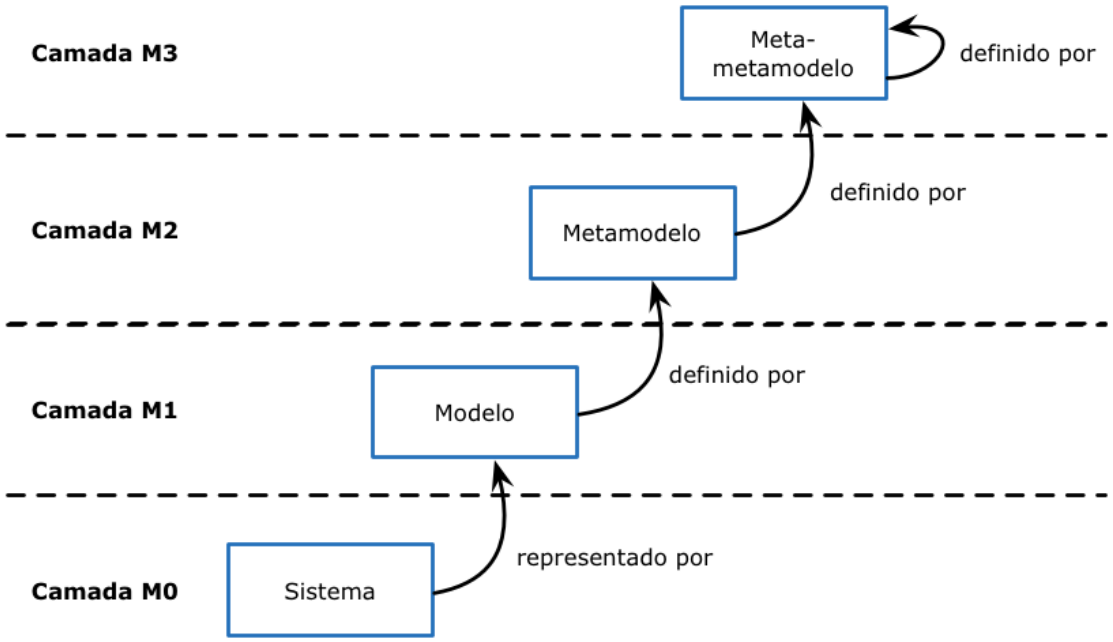


Figura 4.1: Arquitectura de camadas da MOF.

É possível ver na figura a arquitectura com quatro camadas padronizada pela **O***bject* **M***anagement* **G***roup* (OMG) [OMG11]. A camada **M0** representa o sistema em tempo de execução. Na camada **M1** o modelo representa os elementos do sistema da camada M0. Na camada **M2** o metamodelo define o modelo na camada M1, sendo o metamodelo definido pela camada **M3**. Nesta última camada o meta-metamodelo define-se a si próprio.

Para especificar restrições em modelos é usada a linguagem de especificação formal **O***bject* **C***onstraint* **M***odel* (OCL) [WK03]. Essas restrições são definidas em expressões que podem manipular elementos de um modelo (e.g., classes). As expressões não têm efeitos colaterais, ou seja, retornam um valor e não modificam o estado.

As restrições podem ser definidas em invariantes, pré-condições e pós-condições. Uma **invariante** está associada a um elemento de um modelo que tem como objectivo definir uma condição que deve ser sempre válida. As **pré-condições** estabelecem restrições que devem ser verdadeiras antes de executar a operação e as **pós-condições** impõem restrições depois de a operação ter sido executada.

4.3 Linguagem para Domínio Específico

Uma **L**inguagem para **D**omínio **E**specífico (LDE) é uma linguagem a ser aplicada num determinado domínio a fim de solucionar um dado problema com maior facilidade. É frequentemente usada no contexto MDD, pois são usualmente definidas através de metamodelos e usadas em transformações. Por oposição às LDEs, que se focam num domínio específico de um problema, existem as Linguagens de Propósito Geral, como por exemplo o Java, que tem como objectivo a produção de aplicações a uma grande diversidade de domínios. Um exemplo de uma LDE é a linguagem JavaScript cujo propósito serve para fornecer operações de forma a transformar páginas estáticas em dinâmicas.

As Linguagens para Domínios Específicos permitem gerar uma solução ao nível da abstracção do domínio do problema e, desta forma, permitir aos especialistas do domínio perceberem, validarem ou criarem uma LDE [VKV00]. No início da construção de uma LDE é realizado um estudo do domínio em que a linguagem se vai especializar, permitindo fazer uma análise cuidadosa do mesmo. Como o propósito de uma LDE é solucionar um dado problema de um domínio, isso significa que é possível reutilizar a LDE criada para resolver outro problema desse mesmo domínio, aumentando desta forma a sua produtividade. Existem outras características relevantes tal como a fiabilidade, facilidade de manutenção, portabilidade e geração de código [VKV00].

Contudo a criação de uma linguagem que seja produtiva leva a custos no que diz respeito a concepção, implementação e a sua manutenção [VKV00]. É necessário investir na formação dos utilizadores que vão utilizar a LDE, o que traduz num custo em relação ao tempo despendido [VKV00]. É importante referir que se durante o desenvolvimento, ou até mesmo quando já está terminada, de uma linguagem, o domínio for alterado, pode ocorrer um elevado custo de reparação.

4.4 Transformações de Modelos

Transformações de modelos são o método de converter um modelo em outro tipo de modelo usando regras de transformações. Define-se um modelo alvo através de um modelo fonte, realizando uma série de transformações (T: $MM_{alvo} \rightarrow MM_{fonte}$), mediante de condições preestabelecidas [KWB03]. Uma das consequências das transformações de modelos é o facto das regras de transformação permitirem que o modelo alvo tenha um domínio diferente do modelo fonte.

Na Figura 4.2 apresenta-se um esquema do processo de transformação. As regras de transformações passam por associar um elemento do metamodelo fonte a um elemento do metamodelo alvo através de um conjunto de regras. Com a definição das regras de transformação estabelecidas, é possível definir um modelo definido pelo metamodelo alvo através de transformações automáticas.

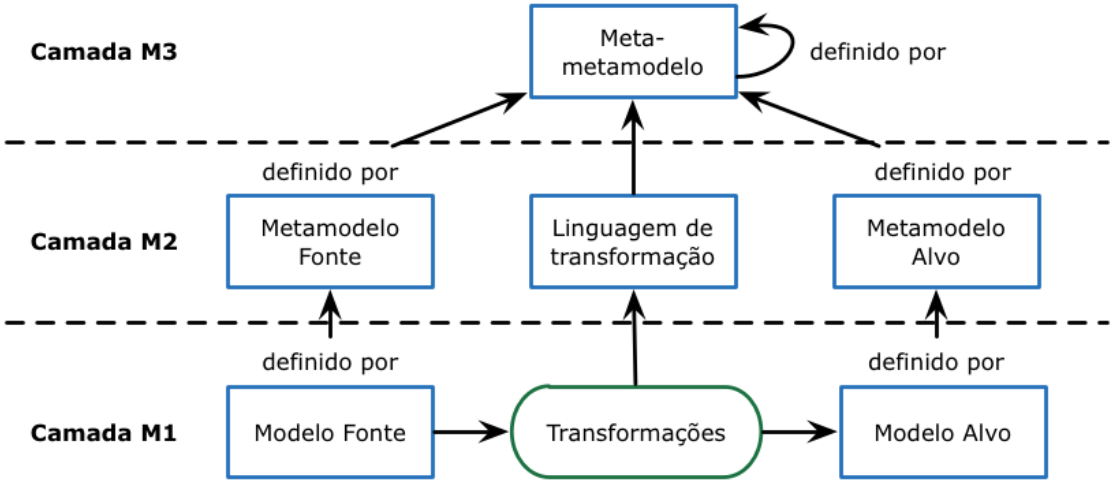


Figura 4.2: Esquema adaptado do processo de transformação [JK06].

As Linguagens de Transformação definem, de forma funcional, a transformação de um modelo fonte num modelo alvo através da manipulação dos seus elementos, tendo em consideração os metamodelos usados na representação dos modelos fonte e alvo [KWB03]. Existem vários tipos de transformações de modelos tais como texto para modelo, modelo para modelo e modelo para texto. Nesta dissertação de mestrado não será dada ênfase a nenhum tipo de transformação.

4.5 Ferramentas

Existem várias ferramentas que permitem desenvolver uma Linguagem para Domínio Específico. Na lista apresentada em baixo vai ser descrito brevemente as ferramentas que são mais utilizadas actualmente:

- **DSL Tools [Béz+05]**: é uma ferramenta criada pela Microsoft que permite criar, editar, visualizar e usar domínios específicos para automatizar o processo de desenvolvimento de *software*. Permite desenvolver um modelo de domínio e um gerador de artefactos textuais; e disponibiliza uma interface gráfica para editar modelos de domínio. A partir do modelo é possível gerar o código e a sua validação;
- **GME [Led+01]**: **G***eneric* **M***odeling* **E***nvironment* (GME) é uma ferramenta criada para a modelação de domínios específicos desenvolvida pelo Instituto para Sistemas Integrados de Software. A configuração da ferramenta é feita através de metamodelos descrevendo a natureza da aplicação. A linguagem do metamodelo é baseada no diagrama de classes UML com restrições em OCL. Estes metamodelos são usados para gerar automaticamente o ambiente específico do domínio, sendo que estes dados ficam armazenados numa base de dados ou em formato XML;
- **EMF/GMF [Béz+05]**: **E***clipse* **M***odeling* **F***ramework* (EMF) é uma ferramenta de modelação e geração de código para a construção de ferramentas ou outro género de aplicações baseada em modelos de dados estruturados. Com base no metamodelo criado, o EMF permite, em tempo de execução, a geração e validação do código, produzindo um conjunto de classes Java associado ao modelo, a visualização e edição do modelo. O EMF é constituído por uma interface gráfica para criar e editar metamodelos (Ecore) e geração de código. O editor gráfico é gerado através do **G***raphical* **M***odeling* **F***ramework* (GMF) [Hun10], usando o metamodelo (Ecore) para definir o sistema através de um componente gráfico. As próximas subsecções vão detalhar estas duas ferramentas que vão ser integradas na dissertação.

4.5.1 Eclipse Modeling Framework

E*clipse* **M***odeling* **F***ramework* (EMF) é um programa de modelação para ser usada em ambiente Eclipse[1] [Bud04]. O seu foco é o desenvolvimento orientado a modelos e a realização de transformações, seja para código ou para outro tipo de modelos.

Modelo Ecore

O Ecore é o modelo usado para representar modelos em EMF [Bud04]. A sua notação é baseada num subconjunto do diagrama de classes da linguagem de modelação UML, como mostra a Figura 4.3.

A figura contém o elemento **EClass**, que é usado para representar uma classe, dado o seu nome, zero ou mais atributos e zero ou mais referências. O **EAttribute** é utilizado para representar um atributo dentro de uma *Eclass* com base no seu nome, sendo o seu tipo especificado no elemento **EDataType** (e.g., *int*, *string*, *float*). Com o elemento **EReference** é estabelecida uma associação/composição entre duas classes e uma referência para o tipo do *target*.

[1] Eclipse: `http://www.eclipse.org/home/index.php` (Acesso feito em Setembro de 2014)

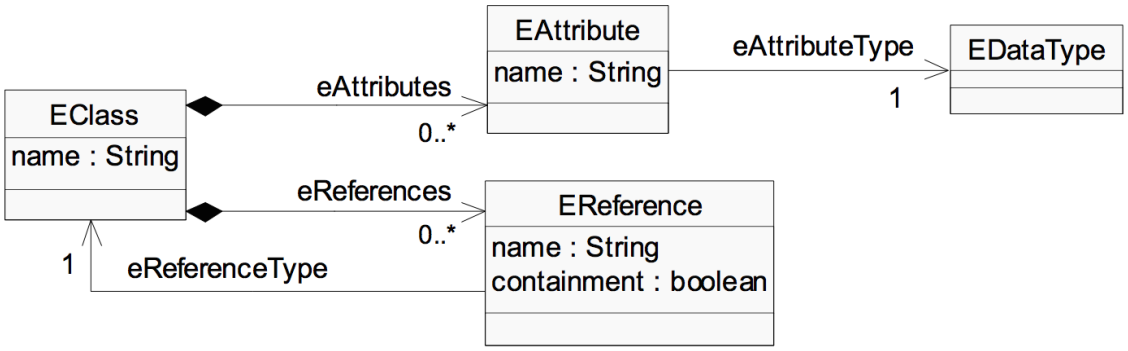


Figura 4.3: Subconjunto simplificado do modelo Ecore [Bud04].

Na Figura 4.4 está representado o modelo Ecore de uma estrutura de um empréstimo a um livro. Um empréstimo tem uma data de início e fim, associado a um livro com os seus atributos autor e título.

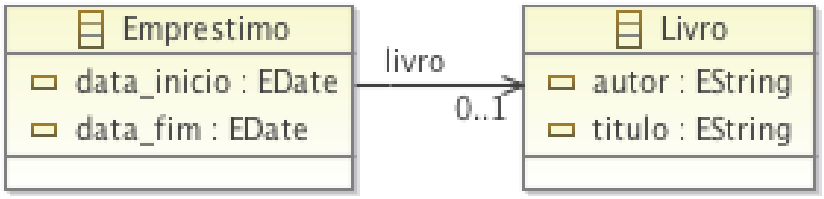


Figura 4.4: Modelo de empréstimo de um livro.

Geração de código

Depois de definir o modelo Ecore, a *framework* Eclipse faz a geração de código Java que possibilita, posteriormente, a manipulação do modelo através do código gerado. A Listagem 4.1 e Listagem 4.2 apresentam o código Java que o EMF gera com base no modelo da Figura 4.4.

Listagem 4.1: Código Java da interface Emprestimo.

```
1  public interface Emprestimo extends EObject
2  {
3
4    Date getData_inicio();
5    Date getData_fim();
6    Livro getLivro();
7
8    void setData_inicio(Date value);
9    void setLivro(Livro value);
10   void setData_fim(Date value);
11
12 }
```

Listagem 4.2: Código Java da interface Livro.

```
1 public interface Livro extends EObject
2 {
3   String getAutor();
4   String getTitulo();
5   void setAutor(String value);
6   void setTitulo(String value);
7 }
```

Na Listagem 4.1 estão especificados os *getters* e *setters* dos atributos "data_inicio", "data_fim" e a referência para a classe Livro. O mesmo acontece na Listagem 4.2 para os atributos "autor" e "título". Através dos métodos gerados pelo EMF é possível criar instâncias dos modelos em código Java.

4.5.2 Epsilon

Epsilon - **E***xtensible* **P***lataform of* **I***ntegrated* **L***anguages for m***O***del ma***N***agement* [Kol+13] é uma plataforma para a construção de linguagens específicas de tarefas estáveis e capazes de comunicar com outros sistemas de forma transparente, para tarefas que façam gestão de modelos (e.g., transformação de modelo, validação, geração de código).

Epsilon suporta actualmente as linguagens **E***psilon* **O***bject* **L***anguage* (EOL), **E***psilon* **V***alidation* **L***anguage* (EVL), **E***psilon* **T***ransformation* **L***anguage* (ETL), **E***psilon* **C***omparation* **L***anguage* (ECL), **E***psilon* **M***erging* **L***anguage* (EML), **E***psilon* **W***izard* **L***anguage* (EWL) e **E***psilon* **G***eneration* **L***anguage* (EGL). No contexto da dissertação apenas irá ser usado o EVL, permitindo a construção de regras de validação para cada modelo de requisitos.

Epsilon Validation Language

E*psilon* **V***alidation* **L***anguage* (EVL) [Kol+13] possibilita a especificação e validação de restrições em modelos, que são especificados de acordo com um metamodelo. O EVL surgiu devido a algumas limitações que o OCL contém tais como: incapacidade de fornecer *feedback* específico ao utilizador caso uma invariante não é satisfeita; não há diferenciação entre erros e *warnings*, logo o OCL assume que todas as restrições são erros; não suporta restrições dependentes; pouca flexibilidade na definição do contexto; não resolve inconsistências; e não tem suportes para restrições inter-modelo.

As especificações da validação estão organizadas em módulos, contendo operações definidas na linguagem EOL e um conjunto de invariantes associados pelo contexto [Kol+13]. Um **contexto** especifica que tipos de elementos do metamodelo vão ser avaliados. Dentro de um contexto é possível definir uma guarda, para que caso este falhe, não seja necessário avaliar a invariante. Uma **invariante** é definida pelo seu nome, a regra de validação, guarda (opcional), pré e pós condição e o *feedback*. É possível definir instruções dentro da invariante para corrigir automaticamente um erro/*warning*. A invariante pode ser do tipo *constraint* ou *critique*. A Listagem 4.3 mostra um exemplo de uma validação de um livro sem título definido.

Listagem 4.3: Restrição EVL de um livro sem título definido.

```
context Livro {
  constraint TituloSemNome {
    check     :  self.titulo.isDefined()
    message   :  "Titulo do livro sem nome."
  }
}
```

4.5.3 Graphical Modeling Framework

G*raphical* **M***odeling* **F***ramework* (GMF) é uma ferramenta que permite criar notações gráficas com base na sintaxe abstracta, definido no modelo Ecore [Hun10]. Este *plugin* é usado em ambiente Eclipse e interliga as ferramentas EMF e **G***raphical* **E***diting* **F***ramework* (GEF) para que o utilizador possa abstrair das configurações de baixo nível que eram necessárias realizar ao usar o GEF. Na Figura 4.5 são apresentadas as actividades do GMF. Os componentes principais são: **modelo de domínio** (*genmodel*), baseado no metamodelo ecore; **desenvolvimento da definição gráfica** (*gmfgraph*); **desenvolvimento das ferramentas de definição** (*gmftool*) para mostrar as classes do modelo de domínio na ferramenta; **modelo de mapeamento** (*gmfmap*) e **geração do diagrama** (*gmfgen*).

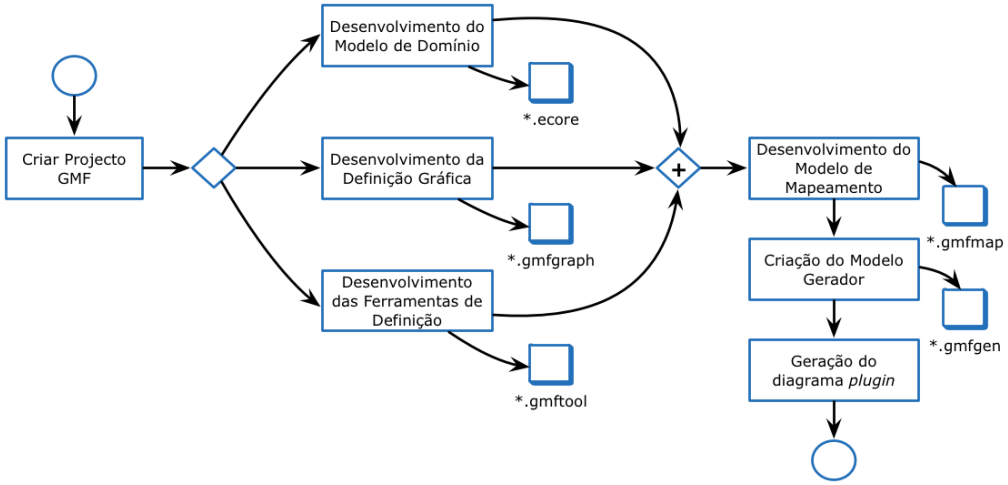


Figura 4.5: Fluxo de actividades do GMF [Hun10].

4.6 Sumário

Neste capítulo foram introduzidos os conceitos de MDD. Foram feitas referências à definição de metamodelos e meta-metamodelos, às LDEs e das linguagens de transformação. O EMF e GMF são as ferramentas a serem usadas nesta dissertação para criar LDEs que vão permitir a criação de modelos de requisitos através do reconhecimento de voz. O EVL vai permitir a especificação de restrições para cada modelo de requisitos. No próximo capítulo é descrito o estudo de trabalhos relacionados da dissertação.

5

Trabalhos Relacionados

De forma a encontrar e analisar trabalhos relacionados foi feita uma adaptação do processo de uma revisão sistemática da literatura [Kee07]. Uma revisão sistemática da literatura tem como objectivo identificar, avaliar e interpretar todas as pesquisas relevantes para uma determinada questão de pesquisa [Kee07]. A questão de pesquisa que se pretende analisar é: "Qual o actual estado da arte sobre a **acessibilidade** de **ferramentas CASE**, especificamente de **modelagem de requisitos**, para **engenheiros com deficiências**?". Para responder a esta questão foram definidas três questões mais específicas, a sua estrutura PICO (**P***opulation*, **I***ntervention*, **C***omparation* e **O***utcomes*) e as respectivas *strings* de pesquisa para obter trabalhos que se encontram nas bibliotecas digitais.

5.1 Perguntas de Investigação

5.1.1 Primeira Questão: Acessibilidade para ferramentas CASE

A primeira questão é estabelecida da seguinte forma: "Quais as recentes **estratégias**/abordagens de **acessibilidade** para **ferramentas CASE**?". A questão foi formulada de acordo com a estrutura PICO:

• **População**: Ferramentas especificadas para apoiar o desenvolvimento de um sistema de *software*.

• **Intervenção**: Estratégias/abordagens de acessibilidade para ferramentas CASE.

• **Aplicações**: Abordagens de acessibilidade para ferramentas CASE.

Através da questão foi definida a seguinte *string* de pesquisa: "*user interfaces* **AND** *blind users* **AND** *(usability* **OR** *accessibility)* **AND** *(diagram* **OR** *case tools)*".

5.1.2 Segunda Questão: Principais desafios enfrentados por um engenheiro de requisitos com deficiências (visuais ou motoras)

A segunda pergunta é definida através da seguinte maneira: "Quais os principais **desafios** enfrentados por **engenheiros de requisitos com deficiência (visuais ou motoras)** em **projectos de software**?". A estrutura PICO foi estabelecida da seguinte forma:

• **População**: Projectos de *software*.
• **Intervenção**: Engenheiros de requisitos com deficiência (visuais ou motoras).
• **Aplicações**: Dificuldades dos engenheiros de requisitos invisuais ou com deficiência motora como membros de uma equipa de desenvolvimento de *software*.

A *string* de pesquisa para a segunda pergunta foi formulada do seguinte modo: "*(accessibility* **AND** *disability* **AND** *inclusion* **AND** *requirements engineering* **AND** *(software projects* **OR** *computing courses))* **OR** *(blind people* **AND** *diagrams)*".

5.1.3 Terceira Questão: Modelos de requisitos com estratégias de acessibilidade

Na última questão específica é questionado: "Quais os **modelos de requisitos** que utilizam **estratégias de acessibilidade**?". A questão foi formulada de acordo com a estrutura PICO:

• **População**: Actividades do processo de Engenharia de Requisitos.
• **Intervenção**: Elicitação de requisitos.
• **Aplicações**: Estratégias de acessibilidade em modelos de requisitos.

A *string* de pesquisa foi formulada da seguinte forma: "*(requirements models* **OR** *UML modeling)* **AND** *(diagram* **OR** *case tools)* **AND** *(accessibility* **OR** *speech recognition* **OR** *(blind* **OR** *disabled) users)*".

5.2 Estratégias de Pesquisa

Depois de definidas as questões de investigação e as respectivas *strings* de pesquisa, os trabalhos foram procurados através das seguintes bibliotecas digitais: ACM Digital Library, SpringerLink, IEEE Xplore e Science Direct. De modo a reduzir o número de artigos na biblioteca ACM, todas as questões de pesquisa usaram o filtro para retornar artigos em estado *proceeding*. Na biblioteca SpringerLink foi necessário filtrar informação para obter resultados mais enquadrados no contexto da dissertação, tendo seguido os seguintes critérios para as primeiras duas questões de pesquisa: 1) Tipo de conteúdo: artigo; 2) Área: *Computer Science*; 3) Sub-área: *Information Systems and Applications*; 4) Publicação: *Innovations in Systems and Software Engineering*; 5) Língua: Inglês. Na última

questão de pesquisa foram pesquisados artigos publicados no contexto *Computers Helping People with Special Needs*. No último sistema de pesquisa, Science Direct, os filtros foram diferentes para cada pergunta de investigação. Na primeira questão o tipo de publicação foi *journal*, o título do *journal* foi *Interacting with Computers* e o tópico seleccionado foi interfaces. Na segunda pergunta foi especificado directamente no sistema de pesquisa o tipo de publicação como *Journal Network and Computer Applications* e na última questão de pesquisa os artigos foram filtrados com o *Journal of Visual Languages & Computing*.

O processo de selecção teve três estados: **obtidos**, **seleccionados** e **descritos**. Os artigos obtidos foram o resultado directo que as bibliotecas atribuíram usando as *strings* de pesquisa e com os filtros para cada tipo de pesquisa. Para escolher os artigos seleccionados, foi tido em consideração se o título, o *abstract* e as *keywords* se enquadravam dentro do contexto de cada pergunta de pesquisa. Se estivessem enquadrados, eram armazenados numa base de dados com o título, autor(es), data de publicação, fonte da publicação e URL. Houve um último refinamento, com nova leitura do *abstract*, das conclusões e de algumas secções dos artigos seleccionados, para depois serem descritos de forma breve na dissertação.

5.3 Resultados

5.3.1 Estratégias/abordagens de acessibilidade para ferramentas CASE

Após a definição da primeira pergunta de pesquisa, a sua estrutura PICO e a *string* de pesquisa, presentes na subsecção 5.1.1, foram pesquisados os artigos referente a esta questão como mostra a Tabela 5.1.

Tabela 5.1: Resultados da primeira questão.

Bibliotecas Digitais	Obtidos	Seleccionados	Descritos
ACM Digital Library	57	7	**3**
SpringerLink	2	0	0
IEEE Xplore	7	2	**1**
Science Direct	14	1	0
Total	80	10	**4**

Abordagens proveniente da ACM Digital Library

As três abordagens que vão ser descritas, através da biblioteca ACM, são o **JavaSpeak**, **AudioGraf** e **PLUMB**. Joan M. e Ann C. [FS02] desenvolveram um ambiente integrado para desenvolvimento de *software*, de nome **JavaSpeak**, que possibilita ao utilizador saber a estrutura e semântica via voz de um programa em Java tais como: nome do pacote em que a classe está alocada, nomes das bibliotecas, classes, interfaces, variáveis, métodos, parâmetros dos métodos, número de níveis de *nesting* de um método, nome

dos blocos (condições *if then else*, *loop*), palavra por palavra ou por caracteres. O sistema usa como *front-end* o programa NetBeans[1] para tirar partido do serviço do *debugging* (entre outros), o Kopi Compiler[2] como *back-end* para a geração e edição de classes Java, o JASPI e o **J***ava* **S***peech* **M***arkup* **L***anguage* (JSML) para o reconhecimento e sintetizador de voz. Esta ferramenta apenas se foca na acessibilidade para a implementação e não para o desenvolvimento orientado a modelos, que é o contexto da dissertação.

AudioGraf [Ken96] é um sistema que apresenta diagramas através de um sistema de áudio. O utilizador invisual explora o diagrama usando um monitor táctil e com uma *pen*. É possível ouvir informação sobre um elemento do diagrama quando a *pen* está na mesma posição que o elemento do diagrama. Os elementos suportados neste programa são as **frames**, **texto** e **conexões**. As frames são compostas pela posição, tamanho, forma, área e cor das linhas e sombra. O texto é composto pelos atributos posição, *string*, tamanho, cor e nome da fonte. As conexões são constituídas por nó fonte, nó destino, tamanho, cor das linhas e setas. A Figura 5.1 sistematiza a interacção que a ferramenta disponibiliza para utilizadores cegos.

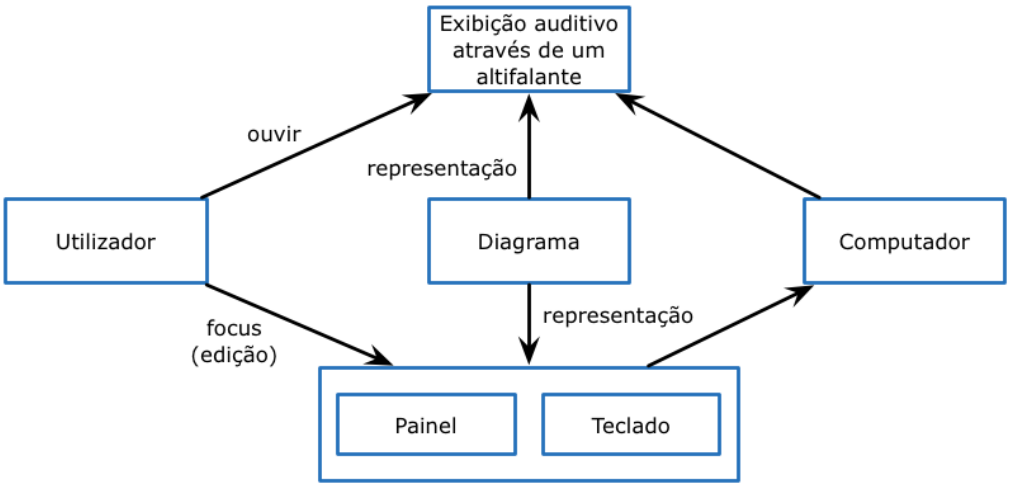


Figura 5.1: Interacção do AudioGraf para utilizadores cegos.

O contexto de acessibilidade nesta abordagem envolve o monitor táctil e a sintetização de voz. Apesar de ser possível estruturar os diagramas de forma acessível, o programa não o faz com mecanismos de reconhecimento de voz, o que pode aumentar o tempo de aprendizagem para pessoas cegas que não conhecem a aplicação.

A ferramenta *ex***PL***oring graphs at* **UMB** (**PLUMB**) [Coh+05] apresenta uma abordagem centrada nos utilizadores invisuais, com *feedback* via áudio para criar e editar grafos. O utilizador pode escolher entre usar o teclado e a caneta electrónica separadamente ou em conjunto como interfaces da ferramenta. A caneta electrónica permite desenhar o gráfico ao receber o *feedback* do áudio com a localização actual da posição do utilizador no gráfico. Este método permite localizar e especificar elementos e relações de um diagrama,

[1]NetBeans: https://netbeans.org (Acesso feito em Setembro de 2014)
[2]Kopi Compiler: http://cs.gmu.edu/~eclab/projects/robots/flockbots/uploads/Main/kopi.html (Acesso feito em Setembro de 2014)

embora a pesquisa dos elementos/relações possa ser lenta porque está dependente dos movimentos do utilizador. O teclado tenta explorar essa debilidade podendo navegar e obter informação dos elementos de forma mais eficiente, renunciando o *feedback* ao utilizador sobre a distância e as relações. A ferramenta não usa nenhum mecanismo de reconhecimento de voz, nem é usado no contexto de modelos de requisitos.

Abordagens proveniente do IEEE Xplore

Na biblioteca IEEE Xplore foram encontrados em dois artigos uma aplicação denominada de **VoCoTo**, discutida na terceira questão de pesquisa (subsecção 5.3.3), e a ferramenta **Scribble** [Nob96]. A aplicação **Scribble** tem como objectivo manter a flexibilidade e facilidade da interface da ferramenta CASE. O artigo evidencia que muitas das ferramentas tem interfaces muito complexas o que traduz numa barreira para usar a ferramenta com facilidade, propondo desenvolver um *design* fraco para manter o foco do utilizador no diagrama em vez de ser na interface. Também referem que a interface é *modeless* porque não existe noção de "*current input mode*" ou "*currently selected tool*" e *selectionless* porque qualquer operação pode ser aplicada em qualquer momento, desde que o objecto suporte essa operação. O contexto de acessibilidade nesta abordagem está centrado na simplicidade e não no sentido de fornecer uma alternativa para usar o programa sem usar o rato/teclado (e.g., reconhecimento de voz).

Abordagem proveniente de outras fontes

Há outra ferramenta que explora a acessibilidade mas que foi encontrada usando o sistema de pesquisa Google. **Kevin** [BE98] é uma ferramenta CASE para criar, editar e apagar **D***ata* **F***low* **D***iagrams*[3] (DFD) através de dispositivos genéricos tácteis. Neste dispositivo, com o DFD ao transitar de um processo para outro, o utilizador ouve mensagens do nome da transição e dos nomes dos estados. O dispositivo táctil é composto por botões de controlo para abrir um modelo, navegação entre as hierarquias de um DFD e criação de um DFD, representados por ícones tácteis, como se pode ver na Figura 5.2.

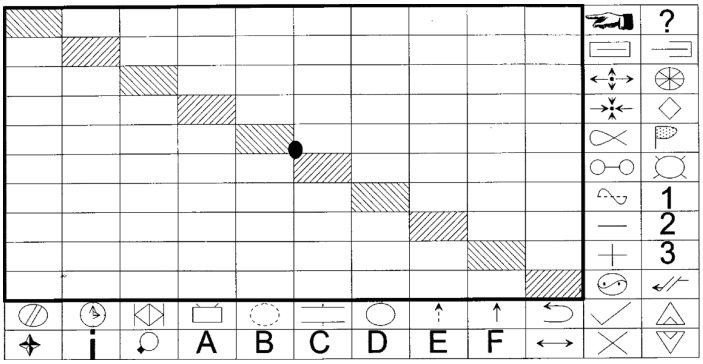


Figura 5.2: Dispositivo táctil do programa Kevin.

[3] **D***ata* **F***low* **D***iagrams*: Representação de um fluxo de dados.

No entanto a ferramenta não está preparada para disponibilizar a informação do *layout* quando é importada em outra ferramenta. A ferramenta foca-se em dispositivos genéricos tácteis e sintetização de voz para a manipulação do programa, quando poderia usar uma abordagem usando reconhecimento de voz na geração de grafos.

5.3.2 Principais desafios enfrentados por engenheiros de requisitos com deficiências (visuais ou motoras) em projectos de software

O número de artigos provenientes da segunda questão de pesquisa especificada na subsecção 5.1.2, está presente na Tabela 5.2.

Tabela 5.2: Resultados da segunda questão.

Bibliotecas Digitais	Obtidos	Seleccionados	Descritos
ACM Digital Library	4	3	0
SpringerLink	3	0	0
IEEE Xplore	5	2	**1**
Science Direct	8	1	**1**
Total	20	6	**2**

Artigo proveniente da Science Direct

No artigo obtido na biblioteca Science Direct, P. Blenkhorn e D. G. Evans [BE98] revelam que os monitores de leitura e os dispositivos em braile são os mais usados para linguagens de programação, mas muito ineficientes na área da modelação. Referem um caso de um engenheiro de *software*, que para interpretar um diagrama, tem que ler a sua representação textual (gerado pela ferramenta CASE), traduzindo em várias páginas de braile, mesmo para diagramas mais simples, requerendo um grande esforço por parte do engenheiro para interpretar o diagrama.

Artigo proveniente do IEEE Xplore

O artigo associado ao IEEE Xplore faz referência a um curso de engenharia de requisitos onde os estudantes teriam que interagir com *stakeholders* com deficiências usando requisitos com/sem acessibilidade [Lud07]. Os estudantes tiveram que se adaptar às tecnologias usadas pelos invisuais e de que forma os *stakeholders* usavam esses conhecimentos, concluindo que: os invisuais não usam o rato para interagir com o computador; a memorização é uma habilidade fundamental neste tipo de situações; o áudio é um componente principal, mas que precisa de ser adaptável ao ambiente do utilizador; o sentimento de independência e auto-suficiência é importante para utilizadores invisuais e leitores de monitor são úteis para assistência ao manipular um *software*. Muitos alunos associavam "utilizadores com deficiência" e "utilizadores com deficiências visuais"

como uma só classe, só que graças ao envolvimento dos *stakeholders* com deficiência visual e amblíopes, o grupo soube fazer a devida diferenciação. O resultado final permitiu aos estudantes por de lado os preconceitos existentes e valorizar as habilidades dos indivíduos com deficiências, podendo ajudar a diminuir o *gap* entre os *stakeholders* e os membros pertencentes à equipa de desenvolvimento.

O artigo do IEEE fez referência a um artigo interessante, proveniente da ACM, onde Joan M. e Ann C. [FS02] referem, num contexto de estudantes invisuais, que o uso de monitores de leitura e o braile (para ler textos longos) são as técnicas mais usadas para os invisuais durante a navegação num computador. Para diagramas e imagens mencionam duas abordagens: impressão com relevo ou manipulação da informação contida na imagem/diagrama.

Artigo proveniente de outras fontes

Por fim, foi encontrado um estudo usando o sistema de procura Google sobre a inclusão de um estudante invisual [PSR12] num curso superior de computação. O aluno tinha preferência em usar um leitor de monitor em vez de um sistema braile para aceder a textos longos. Na cadeira de modelação foi bastante difícil transmitir ao aluno os seus conceitos devido ao forte envolvimento dos diagramas. Numa primeira instância foi usada cartolina, madeira e fixação com velcro para os conceitos mais básicos e para conhecimentos mais avançados foi contratada uma pessoa com formação na área de computação para introduzir técnicas e ferramentas visuais de modelagem que ajudassem ao aluno prosseguir na aprendizagem. A impressão de diagramas com relevo não foi usada pelo estudante porque causava constrangimento devido à quantidade de equipamento a ser usado e a pouca eficiência quando realizava trabalhos de grupo com outros colegas. O uso de tabelas foi a solução encontrada para reduzir esse tipo de dificuldades, melhorando a comunicação entre o professor e estudante e obtendo uma melhor compreensão dos conceitos principais.

5.3.3 Modelos de requisitos que utilizam estratégias de acessibilidade

Os resultados da última pergunta de pesquisa (subsecção 5.1.3), estão apresentados na Tabela 5.3.

Tabela 5.3: Resultados da terceira questão

Bibliotecas Digitais	Obtidos	Seleccionados	Descritos
ACM Digital Library	10	0	0
SpringerLink	110	1	**1**
IEEE Xplore	1	1	**1**
Science Direct	16	1	0
Total	137	3	**2**

Os escassos trabalhos encontrados nesta pesquisa apenas fazem referência a abordagens com recurso a UML (e.g., diagrama de classes, diagrama de estados, diagrama de sequências). Houve muitos resultados provenientes da biblioteca SpringerLink porque a pesquisa estava centrada na interacção pessoa-máquina, mas na prática só houve um resultado enquadrado na área de modelação de requisitos/UML. O artigo seleccionado na biblioteca Science Direct é o mesmo que vai ser descrito proveniente do IEEE Xplore.

Abordagem proveniente do IEEE Xplore e Science Direct

No IEEE Xplore foi obtido um trabalho muito semelhante à abordagem usada nesta dissertação. A proposta consistiu em desenvolver uma ferramenta chamada VoCoTo [LP03], implementando uma abordagem de reconhecimento de voz através de uma gramática para os seguintes diagramas UML: diagrama de classes, diagrama de estados, diagrama de sequências. A aplicação tem três interfaces disponíveis: teclado, rato e comandos via voz. A **G***raphical* **U***ser* **I***nterface* (GUI) da ferramenta foi concebida para ser o mais discreta possível, mas sempre visível, de modo a disponibilizar a informação importante para usar o mecanismo de voz de forma eficaz. A ferramenta usa a aplicação Microsoft SAPI como ferramenta de mecanismo de voz e as gramáticas em formato XML. Apesar de a ferramenta oferecer um mecanismo de acessibilidade para três diagramas UML, que podem ser usados numa fase de desenho no processo de Engenharia de *Software*, a fase inicial do processo de Engenharia de Requisitos continua esquecida no que diz respeito a aspectos de acessibilidade.

Abordagem proveniente da Springer Link

Outro artigo relevante encontrado, proveniente da Springer Link, é o **T***echnical* **D***iagram* **U***nderstanding for the* **B***lind* (TeDUB) [Kin+04]. Permite extrair o XMI (**X***ML* **M***etadata* **I***nterchange*) de um modelo e convertê-lo para uma representação que possibilite ao utilizador com deficiências visuais a interpretação do diagrama. O projecto disponibiliza uma interface acessível com componentes de interface semelhantes a outras aplicações do Windows, permitindo aos utilizadores manipular a interface através de monitores de leitura, *joystick* ou teclado. A aplicação mostra o nome e nó actual oferecendo ao utilizador a possibilidade de aceder aos outros nós que estão conectados ao nó actual através de um *joystick*. No entanto os criadores do TeDUB mencionam uma grande falha caso existam vários nós conectados na mesma direcção no diagrama, pois é apresentado apenas o nó mais próximo. Para além desta limitação, o projecto apenas disponibiliza acessibilidade para os diagramas de casos de uso, classe, sequência e estados para a leitura de diagramas que já tenham sido previamente criados.

A aplicação TeDUB não usa mecanismos de reconhecimento de voz, apenas periféricos tais como monitores de leitura, *joystick* ou teclado. Os diagramas UML usados são o diagrama de classes e estados, sendo que não abordam modelos que se comportam como um grafo (e.g., modelo KAOS e *features*).

Abordagem proveniente de outras fontes

Por fim, foi encontrado o projecto D4ALL [Pan+12] com recurso a pesquisas no sistema Google. É um projecto que se centra em abordagens e técnicas alternativas para aceder e manipular diagramas por pessoas invisuais. Este trabalho recorre à mesma técnica para para capturar a informação do diagrama usado pelo TeDUB, ou seja, utiliza o padrão XMI para ter acesso aos conteúdos dos elementos do diagrama. A ferramenta tem duas interfaces: representação dos elementos UML numa tabela e representação do diagrama de forma visual, mas com a possibilidade de percorrer todos os elementos através do teclado. Esta abordagem não oferece opções para que sejam criados modelos.

A forma de acessibilidade é apenas por teclado, para percorrer as tabelas ou os modelos, usando diagramas em UML. Não existe qualquer uso de mecanismos de reconhecimento de voz, nem da geração de modelos de análise de requisitos como o KAOS ou de *features*.

5.4 Análise comparativa

Após de ter sido feito uma descrição de cada abordagem que cada trabalho usa para melhorar a acessibilidade em ferramentas CASE, a Tabela 5.4 mostra que tipo de abordagens é que as aplicações usam, enquadradas da primeira e terceira questão de pesquisa.

Tabela 5.4: Tipo de abordagens usadas pelas aplicações.

Tipo	JavaSpeak	AudioGraf	PLUMB	Scribble	Kevin	VoCoTo	TeDUB	D4ALL
Reconhecimento de voz	✗	✗	✗	✗	✗	✓	✗	✗
Sintetização de voz	✓	✓	✓	✗	✓	✓	✓	✓
Monitor táctil	✗	✓	✗	✗	✗	✗	✓	✗
Caneta electrónica	✗	✓	✓	✗	✗	✗	✗	✗
Teclado	✓	✗	✓	✓	✗	✗	✓	✗
Rato	✗	✗	✗	✓	✗	✗	✗	✗
Dispositivo táctil	✗	✗	✗	✗	✓	✗	✗	✗
Joystick	✗	✗	✗	✗	✗	✗	✓	✗
Tabelas	✗	✗	✗	✗	✗	✗	✗	✓

As abordagens Reconhecimento de voz e Sintetização de voz são os mecanismos mais importantes no contexto da dissertação. No que diz respeito às aplicações que usam este tipo de abordagens, apenas o VoCoTo usa mecanismos de reconhecimento de voz e o Scribble é a única aplicação que não usa sintetização de voz para melhorar a acessibilidade da ferramenta.

Estas duas abordagens vão ser integradas na ferramenta da dissertação, para que seja possível gerar modelos de requisitos, nomeadamente modelos KAOS, conceptuais e de *features*. A Tabela 5.5 mostra, com base nas aplicações que usam mecanismos de reconhecimento e sintetização de voz, especificadas na Tabela 5.4, se usam modelos KAOS, conceptual ou *features* nas suas ferramentas.

Tabela 5.5: Tipo de modelos usados pelas aplicações.

Tipo	JavaSpeak	AudioGraf	PLUMB	Scribble	Kevin	VoCoTo	TeDUB	D4ALL
Modelo KAOS	✗	✗	✗	✗	✗	✗	✗	✗
Modelo Conceptual	✗	✗	✗	✗	✗	✓	✓	✓
Modelo de *Features*	✗	✗	✗	✗	✗	✗	✗	✗

Nenhuma aplicação usa abordagens para derivar modelos KAOS e de *features*. Apenas as aplicações VocoTo, TeDUB e D4ALL usam os seus mecanismos para derivar diagramas de classes usando a linguagem de modelação UML. No entanto, as ferramentas TeDUB e D4ALL não usam mecanismos de reconhecimentos de voz para gerar os diagramas de classes, tal como foi apresentado na Tabela 5.4.

Após a análise comparativa, através das Tabelas 5.4 e 5.5, pode-se concluir que a aplicação VoCoTo é a abordagem mais próxima do contexto da dissertação, visto que usa mecanismos de reconhecimento e sintetização de voz, embora não se foque em modelos de análise de requisitos.

5.5 Sumário

O estudo dos trabalhos relacionados permitiu analisar alguns trabalhos que oferecerem acessibilidade em ferramentas CASE para utilizadores com deficiências. Os dispositivos principais neste tipo de abordagens englobam leitores de monitor, dispositivos tácteis/braile e mecanismos de reconhecimento e sintetização de voz. Foram encontrados apenas trabalhos sobre acessibilidade para modelos UML. No próximo capítulo é descrita a abordagem VoiceToModel.

6

Abordagem VoiceToModel

6.1 Descrição

De forma a proceder à implementação da ferramenta, foram analisadas as técnicas e *frameworks* apresentadas nos capítulos 2, 3 e 4. A *framework* Eclipse, juntamente com os *plugins* EMF/GMF (subsecção 4.5.1 e 4.5.3), definem a sintaxe abstracta e concreta dos modelos de requisitos KAOS, modelos conceptuais e modelos de *features*, respectivamente apresentados nas subsecções 2.3.1, 2.3.2 e 2.3.3. A linguagem EVL especifica as restrições de cada modelo de requisitos (subsecção 4.5.2). As ferramentas Sphinx-4 (subsecção 3.5.2) e Google Speech API (citado na secção 3.5) interpretam comandos através da voz e, por fim, o FreeTTS (subsecção 3.5.3) fornece *feedback* ao utilizador.

Nas próximas secções vão ser descritos todos os passos e decisões tecnológicas que foram tomadas durante todo o ciclo de implementação da ferramenta VoiceToModel.

6.2 Arquitectura da ferramenta

Para desenvolver uma linguagem para domínio específico de modo a gerar modelos KAOS, conceptuais e de *features*, de forma acessível, foi necessário o desenvolvimento de uma ferramenta capaz de gerar uma DSL compatível com tecnologias de reconhecimento e sintetizador de voz. A Figura 6.1 ilustra a arquitectura da ferramenta em camadas. Nas primeiras três camadas inferiores estão representadas a lógica de suporte para a linguagem de domínio específico, na camada seguinte estão especificados três módulos adjacentes que controlam o serviço de reconhecimento e sintetização de voz, e por fim, na camada superior encontra-se a interface que serve de ponte entre o utilizador e a ferramenta e que representa todas as camadas inferiores a esta, denominada de VoiceToModel.

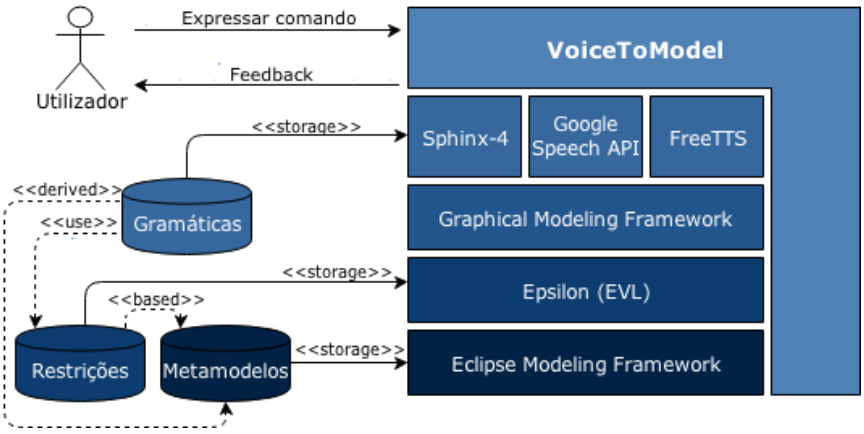


Figura 6.1: Aquitectura da ferramenta VoiceToModel.

Numa perspectiva *bottom-up* situa-se em primeiro lugar a camada Eclipse Modeling Framework, permitindo a especificação dos metamodelos usando a linguagem Ecore. Consoante a especificação de cada metamodelo, o EMF gera classes Java, ao nível da implementação, para manipular objectos de cada modelo. A camada seguinte está o EVL, possibilitando a especificação de restrições baseado no metamodelo, permitindo ao utilizador validar o modelo de requisito. O Graphical Modeling Framework fornece uma infra-estrutura para criação de editores gráficos baseados em EMF. Usando o GMF é possível visualizar os modelos KAOS, conceptual e de *features* graficamente com base na sintaxe abstracta especificada no metamodelo Ecore.

No nível seguinte existem três ferramentas que permitem a modelação dos modelos de requisitos de um dado domínio, de uma forma acessível. Em primeiro lugar está a ferramenta Sphinx-4, usando os componentes modelo acústico e o modelo de linguagem. No modelo acústico é usada a língua inglesa e no modelo de linguagem, com os seus tipos gramática (regras) e estatístico (ditado), foi considerada a gramática de regras. Esta produz melhores resultados do que uma gramática de ditado, devido a várias experiências realizadas durante a especificação da ferramenta. As regras são derivadas de cada elemento do metamodelo, permitindo desencadear acções para manipular o estado do modelo e saber se está correctamente validado.

No entanto, era necessário uma API que tivesse uma boa taxa de sucesso em reconhecimento de palavras com um vocabulário genérico para definir o nome de cada elemento do respectivo requisito, por exemplo em *goals*, classes ou *features*. Com base nessa necessidade é utilizada a Google Speech API desenvolvida pela Google. A ferramenta VoiceToModel envia um ficheiro de áudio com as palavras a serem traduzidas em texto e a API retorna os resultados em formato JSON[1].

Para obter *feedback* de cada operação é usado o FreeTTS, um sintetizador de voz implementado em Java e de fácil integração com as restantes tecnologias.

[1] ***J**ava**S**cript **O**bject **N**otation*: possibilita o armazenamento de informação de forma organizada, legível e fácil de aceder.

6.3 Diagrama de componentes da aplicação

A secção anterior teve como intuito fornecer uma percepção de forma geral como está composta a aplicação nas tecnologias utilizadas, baseada numa arquitectura em camadas. A Figura 6.2 apresenta uma perspectiva mais detalhada da ferramenta, apresentando a organização das classes e quais delas é que fornecem interfaces para as restantes classes, baseado num diagrama de componentes.

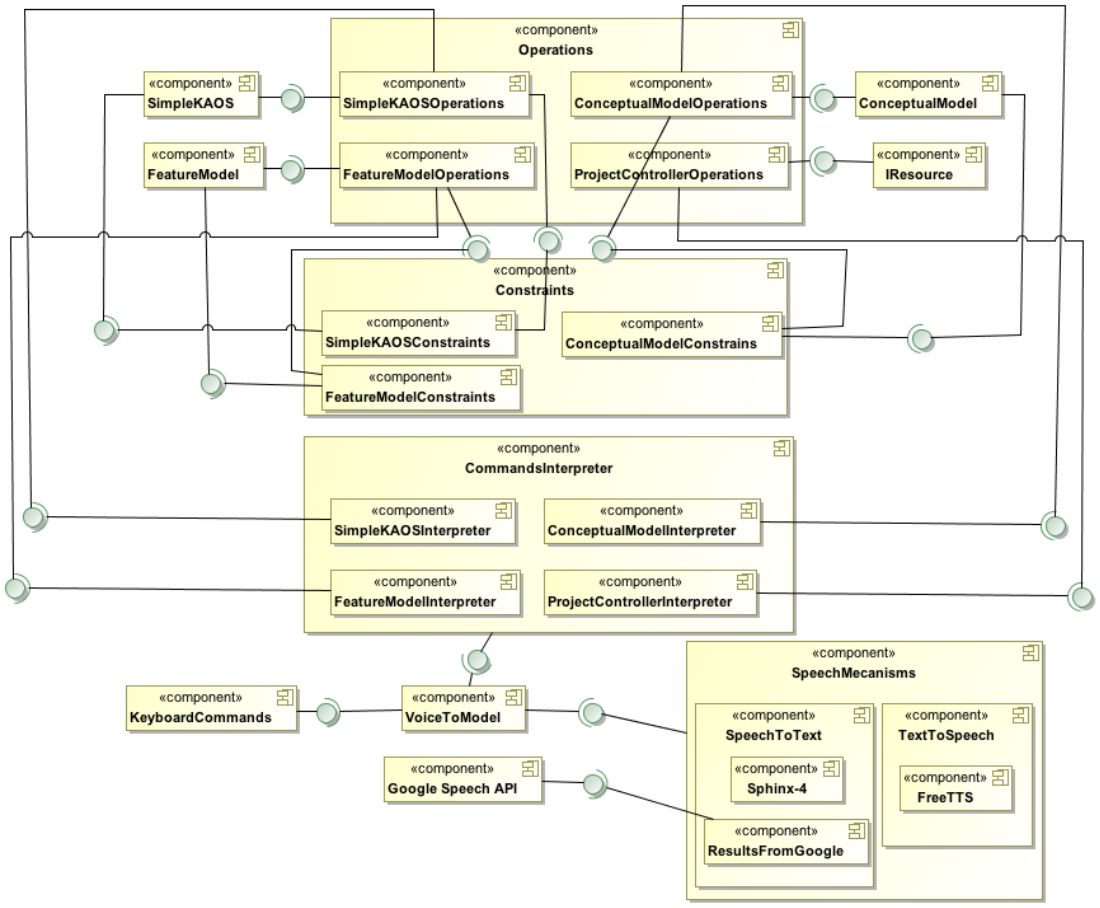


Figura 6.2: Diagrama de componentes da ferramenta VoiceToModel.

O componente **VoiceToModel** requer as funcionalidades dos componentes **Operations**, **Constraints**, **CommandsInterpreter**, **KeyboardCommands** e **SpeechMecanisms**, o que significa que usa todos os serviços de todas as classes, destacando-se como o componente principal do sistema. O mesmo acontece na camada VoiceToModel da arquitectura de camadas da Figura 6.1, dado que é a camada superior e usa todas as tecnologias que suportam a aplicação.

O componente **Operations** disponibiliza todos os métodos necessários para criar instâncias nos modelos de requisitos. Os seus sub-componentes estão directamente relacionados ao seu metamodelo, que são representados pelos componentes **SimpleKAOS**, **ConceptualModel** e **FeatureModel**, assemelhando-se ao repositório de metamodelos na Figura 6.1. O componente **ProjectControllerOperations** é um serviço extra da ferramenta

VoiceToModel que permite criar/editar/apagar projectos e ficheiros de modelos de requisitos. Para controlar o *workspace* do sistema de projectos do VoiceToModel é usado o componente **IResource**, disponibilizado pelo Eclipse, que permite manipular o sistema de ficheiros e directorias.

O componente **Constraints** contém todas as restrições EVL, sejam erros ou *warnings*, de cada modelo de requisito. As restrições são especificadas de acordo com o metamodelo em questão, e invocadas pelo componente Operation. Estas restrições estão situadas num repositório em cada modelo, associado ao repositório de restrições da Figura 6.1. O componente **CommandsInterpreter** tem uma lista de interpretação de comandos para cada modelo de requisitos e do controlador de projectos, representando o repositório de gramáticas definido na Figura 6.1.

O componente **SpeechMecanisms** contém dois componentes essenciais para o reconhecimento e sintetizador de voz. Em primeiro lugar está o componente **SpeechToText**, que tem os objectos necessários para reconhecer voz, com base nos componentes **Sphinx-4** e **ResultsFromGoogle**. O componente Sphinx-4 está localizado na *source* do projecto e o componente ResultsFromGoogle realiza pedidos HTTP que permitem obter expressões provenientes de ficheiros de áudio usando o componente **Google Speech API**, externo à aplicação VoiceToModel. No componente **TextToSpeech** é disponibilizado *feedback* ao utilizador através do sintetizador de voz FreeTTS.

Por fim, o componente **KeyboardCommands** funciona como complemento da ferramenta, facultando atalhos no teclado para suspender a aplicação ou alterar o nome de um elemento em foco, tais como uma classe, atributo, *feature* ou *goal*.

Nas próximas secções vai ser explicado como foi definido o modelo de interacção e é apresentado em mais detalhe como foram implementados os componentes SimpleKAOSOperations, ConceptualModelOperations, FeatureModelOperations e ProjectControllerOperations e as restrições do componente Constraint.

6.4 Modelo de interacção

Um modelo de interacção permite definir **como** e **de que modo** é que um utilizador pode interagir com um sistema informático. Em primeiro lugar é necessário estabelecer qual o estilo de interacção que irá ocorrer entre o utilizador e o sistema. Na seguinte lista são apresentados alguns dos principais estilos de interacção [Tuf03].

- Linguagem de comandos via voz/texto;
- Linguagem natural via voz/texto;
- Manipulação de menus;
- Formulários;
- *Window*, *icon*, *menu* e *pointing device* (WIMP);
- 3D *interface*.

Dado o contexto da dissertação, a escolha inicial recaia para linguagens de comandos e linguagem natural via voz. Como apenas é necessário um estilo de interacção, foi necessário analisar cada aspecto dos dois estilos de interacção. A linguagem natural [Tuf03] é um estilo relaxado dado que não necessita de grandes restrições para comunicar, apenas necessita de ser expressado num determinado idioma (e.g., Inglês). Apesar de ser um estilo que não obrigue a um treino muito específico, as frases que o utilizador expressa podem ser muitas vezes ambíguas e, consequentemente, gramaticalmente incorrectas. As linguagens de comandos [Tuf03] são mais restritas que a linguagem natural, pois permitem estruturar mais facilmente os comandos e parâmetros, concebendo ao utilizador expressar mais frequentemente gramáticas correctas e simples.

Como a aplicação tem como foco a modelação de requisitos, foi necessário a especificação de comandos simples e intuitivos de modo a reduzir a margem de erros na modelação, sendo que a escolha reflectiu para a linguagem de comandos.

6.4.1 Visão geral dos comandos

A composição dos comandos baseia-se nos princípios das operações básicas de **C***reate*, **R***ead*, **U***pdate* e **D***elete* (CRUD), permitindo ao utilizador inserir, ouvir, actualizar e apagar informação em cada modelo de requisitos e na manipulação de projectos. Na Tabela 6.1 são apresentados os comandos implementados de forma genérica que respeitam estes princípios.

Tabela 6.1: Comandos principais para manipulação do VoiceToModel.

Comando	Descrição
add <parâmetros>	Usado para adicionar elementos no respectivo modelo de requisitos (exemplo: classe, atributo, *goal*, *feature*).
create <parâmetros>	Permite criar projectos e modelos de requisitos.
change <parâmetros>	Altera elementos no respectivo modelo de requisitos (exemplo: classe, atributo, *goal*, *feature*).
delete <parâmetros>	Serve para apagar elementos no respectivo modelo de requisitos (exemplo: classe, atributo, *goal*, *feature*) ou projectos/-modelos.
say <parâmetros>	Tem como objectivo informar ao utilizador um estado específico de um modelo ou de um projecto.

Através destes comandos apresentados na Tabela 6.1 o utilizador poderá realizar operações básicas para a manipulação de um modelo de requisitos ou até mesmo na gestão dos projectos num ambiente *workspace*. O sistema VoiceToModel oferece outros comandos, demonstrados na Tabela 6.2, que permitem melhorar a experiência ao usar a aplicação VoiceToModel.

Tabela 6.2: Restantes comandos.

Comando	Descrição
find <parâmetros>	Permite que o sistema se foque num determinado elemento do respectivo modelo de requisitos (exemplo: classe, atributo, *goal*, *feature*).
open <parâmetros>	Abre um projecto/modelo de requisito.
sleep mode <parâmetros>	Possibilita ao utilizador meter/tirar o sistema VoiceToModel em *stand-by*.
repeat last feedback	Repete o *feedback* da última operação.
validate model	Informa o utilizador dos erros/*warnings* que o modelo poderá ter.
describe commands	Descreve ao utilizador os comandos via voz de um determinado modelo ou do controlador de projectos.
back to project controller	Com este comando o utilizador poderá manipular projectos/apagar ou abrir outros modelos.
help mode <parâmetros>	Através deste comando o sistema informa ao utilizador o comando correcto caso este seja mal expressado (exemplo: a falta de argumentos).
undo command	Desfaz a última operação realizada.
start program	Inicia o sistema VoiceToModel.
end program	Encerra o sistema VoiceToModel.

O comando **find** ao focar-se num determinado elemento vai possibilitar, nos comandos seguintes (e.g., adicionar/criar/apagar), não seja necessário mencionar o nome do elemento em questão. Esta funcionalidade permite que o utilizador omita um parâmetro, como por exemplo, adicionar 5 features associadas à *root* sem mencionar o nome da *root* nas 5 operações.

A operação **validate model** analisa se um modelo está correctamente bem especificado através de restrições implementadas na sintaxe abstracta, mais concretamente no metamodelo do modelo de requisitos, e na semântica estática em EVL. O sistema irá comunicar ao utilizador eventuais erros ou *warnings* que existam no modelo.

Os comandos **describe commands** e **help mode** ajudam a melhorar a acessibilidade quando o utilizador está a interagir com o sistema. O comando **describe commands** informa ao utilizador que tipo de comandos tem o modelo actual/controlador de projectos, como por exemplo: "add", "change", entre outros. O comando **help mode**, no modo activo, é executado quando o utilizador diz algo incompleto nos comandos implementados, isto é, caso diga "add" ou mesmo quando falha algum dos parâmetros obrigatórios o sistema informa que o comando está incompleto e que deve expressar de uma maneira específica. A Figura 6.3 mostra o processo que a ferramenta usa na detecção de ajuda.

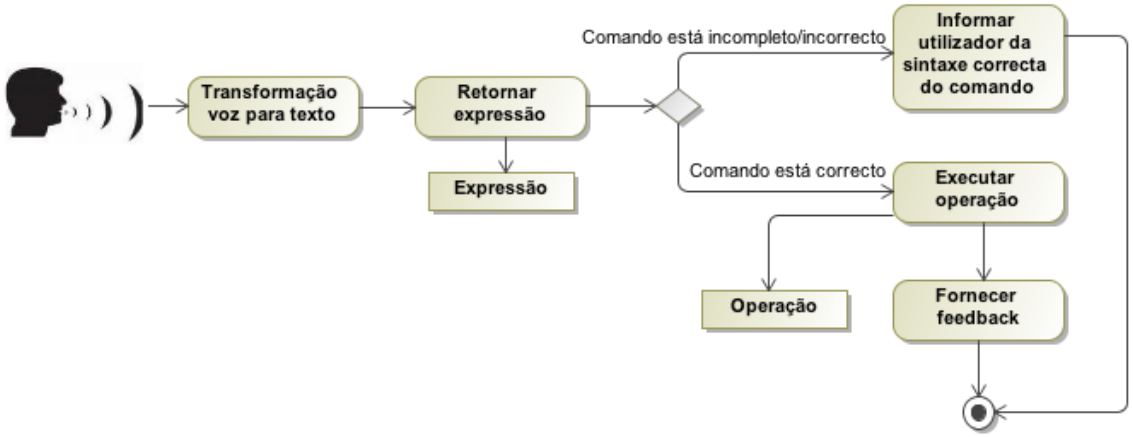


Figura 6.3: Processo do Help Mode.

O **undo command** é uma funcionalidade usada em todos os modelos de requisitos que permite recuperar o estado anterior do modelo. O controlador de projectos não suporta esta funcionalidade devido à quantidade de metadata que era necessário armazenar. A estrutura de dados implementada é baseada numa *stack* com ordem **L***ast* **I***n* **F***irst* **O***ut* (LIFO), como se pode ver nas Figuras 6.4 e 6.5.

estado ∪ {add feature Image}
estado ∪ {add feature Text}
estado ∪ {add group OR}
estado ∪ {find Send}
estado ∪ {add optional Notify}
estado ∪ {add mandatory Receive}
estado ∪ {add mandatory Send}
estado ∪ {add root Email}

Figura 6.4: Primeira *stack*.

estado ∪ {add feature Text}
estado ∪ {add group OR}
estado ∪ {find Send}
estado ∪ {add optional Notify}
estado ∪ {add mandatory Receive}
estado ∪ {add mandatory Send}
estado ∪ {add root Email}

Figura 6.5: Segunda *stack*.

Na Figura 6.4 é apresentado um modelo de *features* através do formato de comandos, inserido numa estrutura *stack*. No fundo da pilha está especificado o começo do modelo e no topo está o último elemento a ser inserido no modelo de *features*. O tamanho da *stack* pode ser dinâmico ou fixo. A parametrização está alocada num ficheiro de configurações situado no *workspace* do projecto. A Figura 6.5 contém o mesmo modelo da Figura 6.4, excepto o último elemento. Esta diferença deveu-se ao facto do utilizador ter usado o comando **undo command**, permitindo retroceder na última operação realizada, e desta forma, removendo o comando no estado do objecto (estado \ {add feature Image}).

Nas subsecções 6.5.2, 6.6.2, 6.7.2 e 6.8.2 estão descritos os comandos do SimpleKAOS, Modelo Conceptual, Modelo de *Features* e do Controlador de Projectos.

6.4.2 Comandos através do teclado

O estilo principal de interacção no sistema VoiceToModel é por linguagem de comando. Contudo decidiu-se implementar dois comandos via teclado que permitissem aos utilizadores usarem, caso o sistema de reconhecimento de voz não estiver a funcionar correctamente, apresentado na Tabela 6.3. Tais funcionalidades não substituem o sistema VoiceToModel, mas sim como um complemento que permitirá melhorar a experiência ao utilizador.

Tabela 6.3: Comandos via teclado.

Comando	Semelhança	Descrição
ALT+C	**change** <parâmetros>	Permite alterar o nome de um elemento do modelo de requisitos.
ALT+SPACE	**sleep mode** <parâmetros>	Hiberna/acorda o sistema VoiceToModel.

No primeiro comando assume-se que o elemento a ser alterado, como por exemplo classe, atributo, *goal* ou *feature*, é o último elemento que afectou o estado do modelo num comando "add"/"change" como exemplo. Este comando também fica guardado na pilha de comandos caso no futuro o utilizador pretenda retroceder usando o "undo command".

O segundo comando está implementado usando lógica *boolean*, ou seja, assume-se que no início do programa o sistema está acordado. Caso o utilizador use o comando ALT+SPACE o sistema muda o estado a *false* e o sistema passa a não reconhecer comandos que o utilizador diga (excepto o comando "sleep mode off"). Para retomar o programa ao activo basta executar de novo o comando ALT+SPACE ou o comando "sleep mode off" que o sistema passa a reconhecer novamente os comandos.

6.4.3 Processo do VoiceToModel

Nas subsecções anteriores foi apresentado o estilo de interacção entre o utilizador e o programa e a estrutura dos comandos que permite ao utilizador expressar de modo a desencadear uma acção. Todavia foi necessário desenvolver um processo que permitisse ao utilizador comunicar com o VoiceToModel, e este responder. A Figura 6.6 apresenta um processo genérico que foi implementado.

Figura 6.6: Processo genérico da aplicação VoiceToModel.

Processo do controlador de projectos

Quando o utilizador inicia a aplicação VoiceToModel, o sistema disponibiliza a gramática de controlador de projectos. A Figura 6.7 sumariza o processo do controlador de projectos.

O processo começa com o utilizador a iniciar o programa. Quando o programa tem as suas bibliotecas todas carregadas em memória, notifica o utilizador que o programa está preparado através da seguinte mensagem "*VoiceToModel is ready. Please say start program to begin.*". O utilizador terá que dizer o comando **start program** para a comunicação continuar. Caso o programa tenha um modelo aberto, isto é, que tenha sido criado anteriormente e que a janela esteja activa, o sistema pergunta ao utilizador se deseja continuar a manipular o modelo em questão com a mensagem "*The KAOS/Conceptual/Feature Model <number>, in project <number> is open. Do you want to start modeling? Yes or No?*". Se a resposta for afirmativa, então a gramática do controlador de projectos deixa de estar activa, e consequentemente, irá ser trocada pela gramática do modelo KAOS/Conceptual/*Features* para editar o estado do modelo. Se a resposta for negativa, o utilizador poderá manipular o *workspace* dos projectos e modelos.

Caso não haja nenhum modelo aberto, o sistema notifica o utilizador de quantos projectos existem no *workspace* e sugere o uso do comando "create project" ou "open project <number>". Quando se abre um projecto o programa informa de imediato o tipo e identificadores dos modelos que já foram criados. O utilizador poderá desta forma criar/abrir/apagar projectos e modelos (ver comandos na secção 6.8) ou sair do programa VoiceToModel expressando o comando **end program**.

Processo de modelação de requisitos

Depois de o utilizador usar o comando **open <model> <number>** o processo de reconhecimento e sintetização de voz deixa de reconhecer comandos do controlador de projectos e passa a reconhecer comandos do respectivo modelo de requisitos, como está especificado na Figura 6.8.

Se o modelo contiver elementos já criados, o sistema recupera esses dados, caso contrário cria um modelo em branco. O sistema tem dois modos para interpretar os comandos. Se um comando necessitar do sistema Google Speech API, então grava um ficheiro áudio e envia para a Google com o intuito de receber os resultados. Se apenas for necessário usar o Sphinx-4, o sistema processa o que o utilizador disse, internamente, sem ser necessário de usar o serviço da Google. Depois dos serviços de reconhecimento de voz terem transformado o que o utilizador expressou em texto, o sistema executa o comando e altera o estado do modelo (caso seja necessário).

Para voltar ao controlador de projectos o utilizador deverá expressar o comando **back to project controller**. Através desse comando o processo muda novamente para o que está representado na Figura 6.7, ou seja, troca de gramática permitindo ao utilizador usar os comandos para manipular o *workspace* ou sair do sistema.

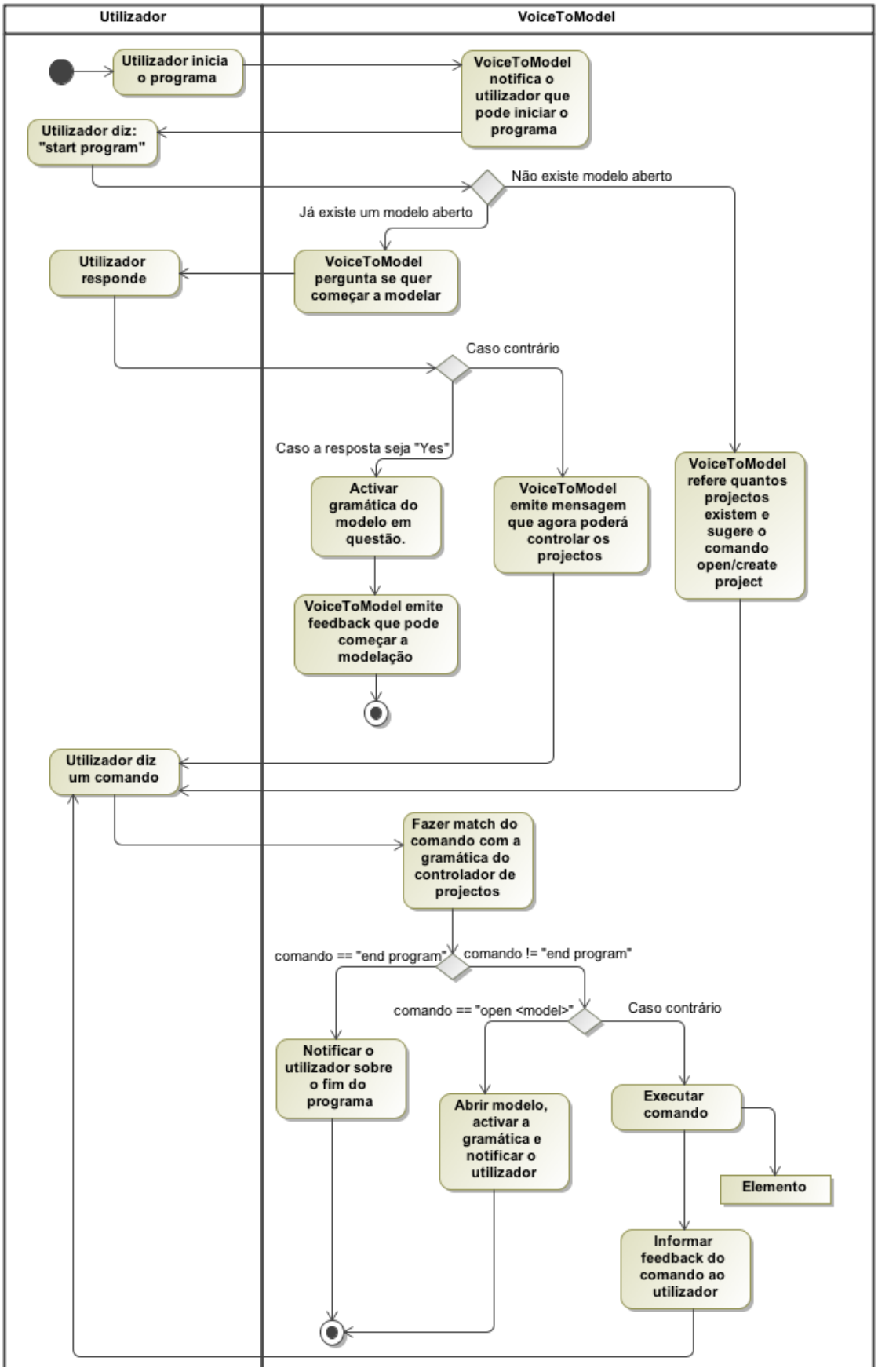


Figura 6.7: Processo do controlador de projectos.

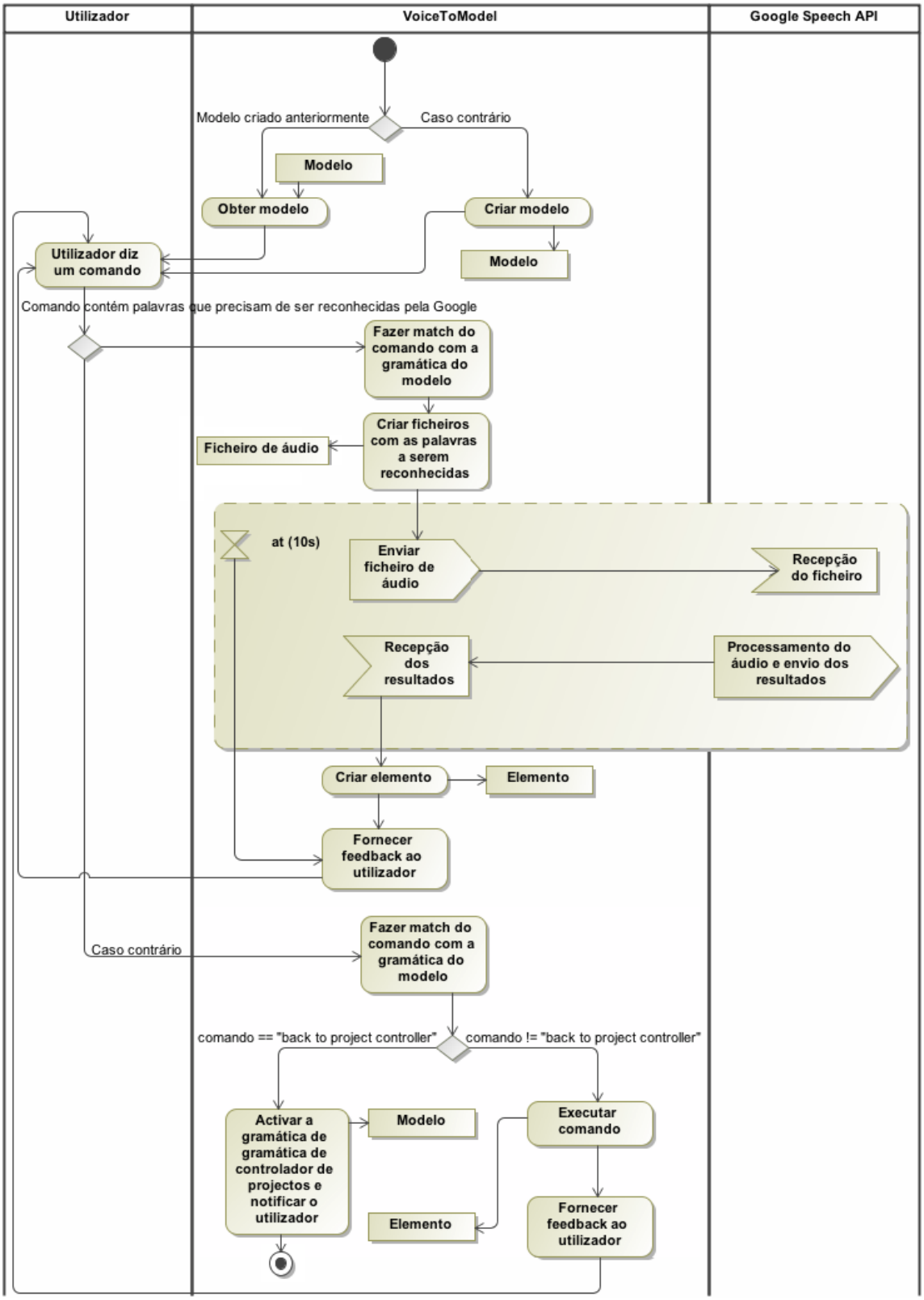


Figura 6.8: Processo de modelação de requisitos.

6.5 SimpleKAOS

SimpleKAOS é uma versão simplificada do modelo KAOS integrada no sistema VoiceToModel. Este modelo mapeia os requisitos de um sistema para uma perspectiva *goal-oriented* usando os seguintes conceitos: *goal*, refinamento *AND*/*OR* e agentes. Não foram implementados os restantes conceitos (e.g., operacionalização, obstáculos) porque iria aumentar a complexidade da solução do problema e o intuito é analisar o comportamento da estruturação dos requisitos usando o paradigma *goal-oriented* de forma simplista.

Nas próximas subsecções serão descritas com detalhe a sintaxe abstracta, comandos e restrições do SimpleKAOS.

6.5.1 Metamodelo

O metamodelo do SimpleKAOS está especificado na Figura 6.9. O modelo KAOS contém **nodes** e **links**, representados abstractamente. Como um modelo KAOS é reflectido através de uma árvore, tem que haver a noção de nó inicial, representando o conceito do sistema a ser modelado. Essa especificação está presente na referência *root* entre as classes **KAOSModel** e **Goal**. A classe *Goal* é um caso de especificação da classe abstracta *node* que permite fazer ligações com vários *links*, referenciado em **links**.

As classes **OR** e **AND** são duas especializações da classe abstracta *Link* e *GoalRefinement*. Como um *GoalRefinement* contém *goals*, foi criado no metamodelo a associação *goals* de forma a armazenar uma lista de *goals* que um refinamento pode ter.

Um **Agent** é um caso de especialização da classe *Object*, que por sua vez estende da classe *Node*. Para ligar um agente a um *goal* foi adicionada a classe **AgentLink** (especialização da classe *Link*), com o objectivo de criar uma ligação entre estes dois nós.

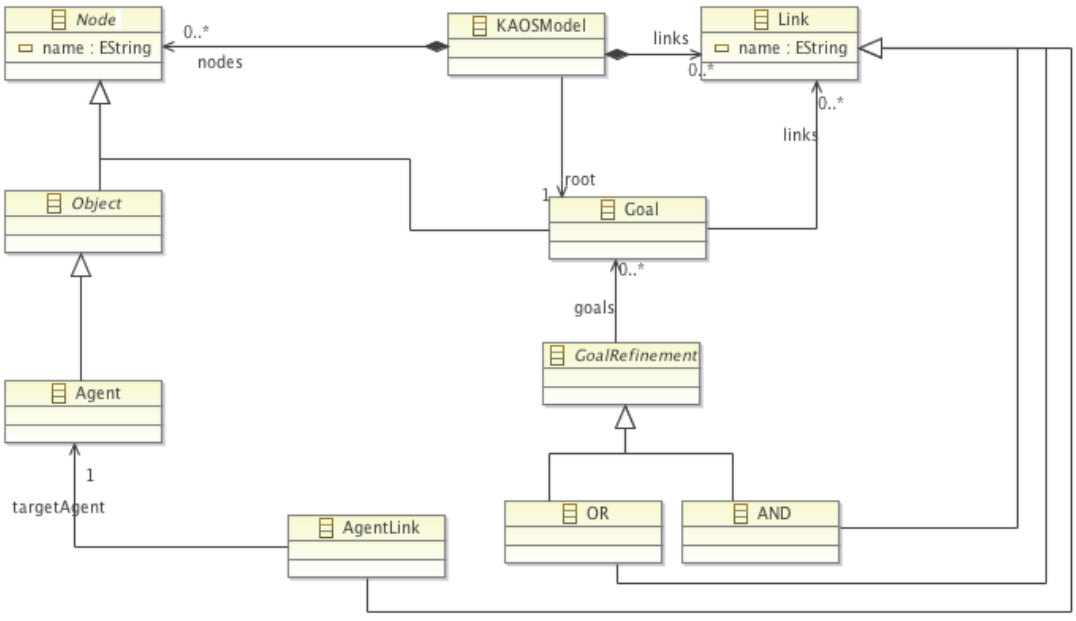


Figura 6.9: Metamodelo SimpleKAOS, versão simplificada do KAOS [MH05].

6.5.2 Gramática e Comandos

Com base no metamodelo do SimpleKAOS é possível extrair conceitos que vão permitir definir comandos para gerar um modelo KAOS, tais como: *goal*, *agent* e *refinement*. Estes conceitos são provenientes do nome de cada classe, presentes no metamodelo, permitindo derivar os seus elementos para comandos que possibilitem a manipulação do estado desses objectos. A secção B.1 do Anexo B associa cada *goal*, agente e refinamento para um ou mais comandos (e.g., "@v2m.add", "@v2m.change", "@v2m.delete", entre outros).

De modo a estruturar os comandos a serem utilizados para a construção do modelo KAOS, a Figura 6.10 mostra as regras gramaticais que foram especificadas e implementadas.

```
grammar SimpleKAOS;

<SimpleKAOS>      =  add <add_element>              |
                     change <change_element>       |
                     delete <delete_element>       |
                     find                          |
                     find by position              |
                     say <describe>                |
                     sleep mode (on | off)         |
                     repeat last feedback          |
                     validate model                |
                     undo command                  |
                     describe commands             |
                     help mode (on | off)          |
                     <choice>                      |
                     back to project controller    |
                     <NULL>                        ;

<add_element>     =  root | goal refinement <refinement> | agent | <NULL>;

<change_element>  =  goal name | agent name | refinement (or | and) to <refinement> | <NULL>;

<delete_element>  =  goal | agent | refinement <refinement> | <NULL>;

<refinement>      =  or | and | <NULL>;

<describe>        =  number of sub goals | number of agents | sub goals name | agents name |
                     total of goals | total of agents | actual goal | <NULL>;

<choice>          =  yes | no | <NULL>;

public <command>  =  <SimpleKAOS>;
```

Figura 6.10: Gramática SimpleKAOS.

Estes comandos permitem ao utilizador alterar o estado do modelo, tais como "add", "change" e "delete" e ler o estado do modelo como "find" e "say". O SimpleKAOS é representado como um grafo K = (G, R, A), em que:

- G ∈ {*goal*};
- R ∈ {*refinement AND*, *refinement OR*};
- A ∈ {*agent*}.

Baseada na especificação e na gramática da Figura 6.10, a Tabela 6.4 ilustra 10 comandos que mostram diferentes comportamentos. A secção A.1 do Anexo A disponibiliza todos os comandos implementados.

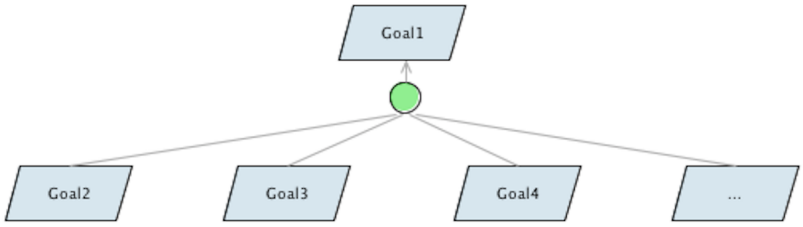
Na primeira coluna é apresentado o comando a ser executado e na segunda coluna a sua descrição. O *feedback* vai possibilitar ao utilizador de saber o resultado da operação via voz. A pós-condição especifica qual o resultado expectável da respectiva operação. Por fim, na última coluna é apresentada a complexidade no pior caso. Por exemplo, quando se adiciona a *root* do modelo KAOS, a complexidade é sempre constante seja no melhor caso, pior ou caso esperado. Para adicionar um agente, a complexidade pode não ser constante visto que se no comando se especificar o nome do *goal* que se quer associar a um agente, terá que ser necessário procurar no grafo do modelo KAOS o objecto que representa o *goal*.

Tabela 6.4: Exemplos dos comandos do SimpleKAOS.

Comando	Descrição	Feedback	Pós-condição	Complexidade
add root <name>	Adiciona um *goal* no modelo.	*Root <name> has been added successfully*.	*Goal* adicionado.	O(1)
add goal refinement OR <name>	Adiciona um *sub-goal* com refinamento OR.	*Goal <name> with refinement OR has been added successfully*.	*Sub-goal* com refinamento OR adicionado.	O(1)
add agent <name>	Associa um agente a um *goal*.	*Agent <name> has been added successfully*.	Agente adicionado.	O(1)
change goal name <old> to <new>	Muda o nome de um *goal*.	*Goal <old> was changed successfully to <new>*.	Nome alterado.	O(|G|+|R|)
delete goal <name>	Apaga um *goal* no modelo.	*Goal <name> was deleted successfully*.	*Goal* removido.	O(|G|+|R|)
delete agent <name>	Apaga um agente no modelo.	*Agent <name> was deleted successfully*.	Agente removido.	O(|A|)
find <name>	Procura um *goal* e instancia como *sub-root*.	*Goal <name> found successfully*.	*Goal* encontrado.	O(|G|+|R|)
find by position <number >	Procura um *goal* e instancia como *sub-root* através da sua posição.	*Goal <name> with the position <number> found successfully*.	*Goal* encontrado.	O(|G|+|R|)
say total of goals	Indica o número total de *goals*.	*KAOS Model has <number> goals*.	Número de *sub goals* retornado.	O(|G|+|R|)
say sub goals name <name>	Indica os nomes de cada *sub goals* que um *goal* tem.	Exemplo: *Goal <name> has a refinement AND with the following goals: <subgoals>*.	Nomes de cada *sub goals* retornado.	O(|G|+|R|)

No que diz respeito aos comandos implementados, foi necessário uma reflexão sobre a forma como os comandos devem ser utilizados de forma intuitiva. Por exemplo, ao adicionar uma *root* e *sub goals*, é possível fazê-lo através dos comandos "add root <name>" e "add goal refinement OR/AND <name>", o que faz com que os todos os *goals* adicionados sejam associados à *root*. No entanto é possível ao utilizador usar o comando "add goal refinement OR/AND <goal2> to <goal1>", caso queira especificar o nome do *goal* que quer refinar.

A Figura 6.11 mostra dois cenários de implementação de um *goal* com refinamentos. O primeiro cenário adiciona a *root* do modelo KAOS e depois atribui refinamentos OR para a *root* usando a flexibilidade dos parâmetros, ou seja, sem haver a necessidade de especificar o *goal* pai. O segundo cenário é idêntico ao primeiro com excepção nos comandos que associa os refinamentos OR, isto é, o utilizador expressa o nome da *root* nos parâmetros opcionais.



Cenário 1
add root Goal1
add goal refinement OR Goal2
add goal refinement OR Goal3
add goal refinement OR Goal4
add goal refinement OR ...

Cenário 2
add root Goal1
add goal refinement OR Goal2 to Goal1
add goal refinement OR Goal3 to Goal1
add goal refinement OR Goal4 to Goal1
add goal refinement OR ... to Goal1

Figura 6.11: Exemplo do uso da omissão de parâmetros no SimpleKAOS.

O comando "find <goal>" foca um *goal* de modo a que futuros comandos como "change"/"delete"/"say" não seja necessário especificar o último argumento. Neste exemplo que está apresentado na Tabela 6.4, se o "find" tivesse sido usado anteriormente, então o comando "change goal name" poderia omitir o parâmetro <old>, os comandos "delete goal/agent" omitiriam o <name> e o comando "say sub goals name" poderia omitir o parâmetro <name>. Caso seja usado um "add"/"change"/"delete" então os próximos comandos também poderão omitir o último argumento. A diferença é que o comando "find" não altera directamente o estado do modelo e os restantes comandos "change"/"delete" alteram, mas todos proporcionam a possibilidade de omitir o último parâmetro de cada argumento.

A Figura 6.22 demonstra dois cenários de refinamentos de *goals* com e sem o comando "find". No primeiro cenário são usados os comandos "find", duas vezes o "add goal refinement", que perfaz um total de três comandos, e o segundo cenário usa apenas dois

"add goal refinement" num total de dois comandos. Apesar do segundo cenário ser teoricamente mais rápido de realizar as operações, se forem adicionados muitos refinamentos no mesmo *goal* o primeiro cenário acaba por ser mais prático.

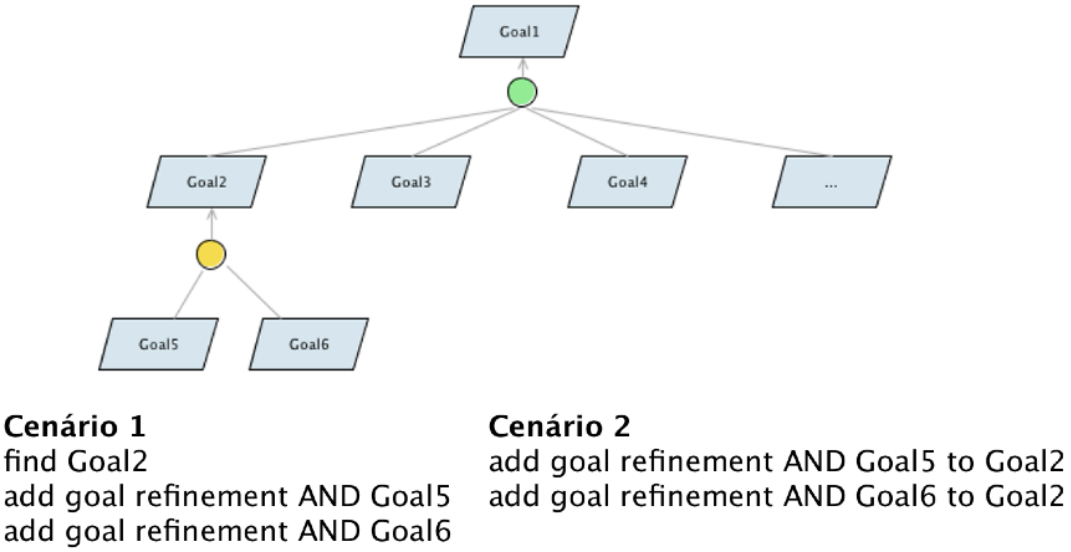


Figura 6.12: Exemplo do uso do comando find no SimpleKAOS.

O objectivo do comando "find by position <number>" é idêntico ao comando "find <goal>", contudo a parametrização não requer um nome, mas sim o número que está associado através da sua posição no grafo. A Figura 6.13 ilustra um exemplo de como é atribuído uma posição a cada *goal*.

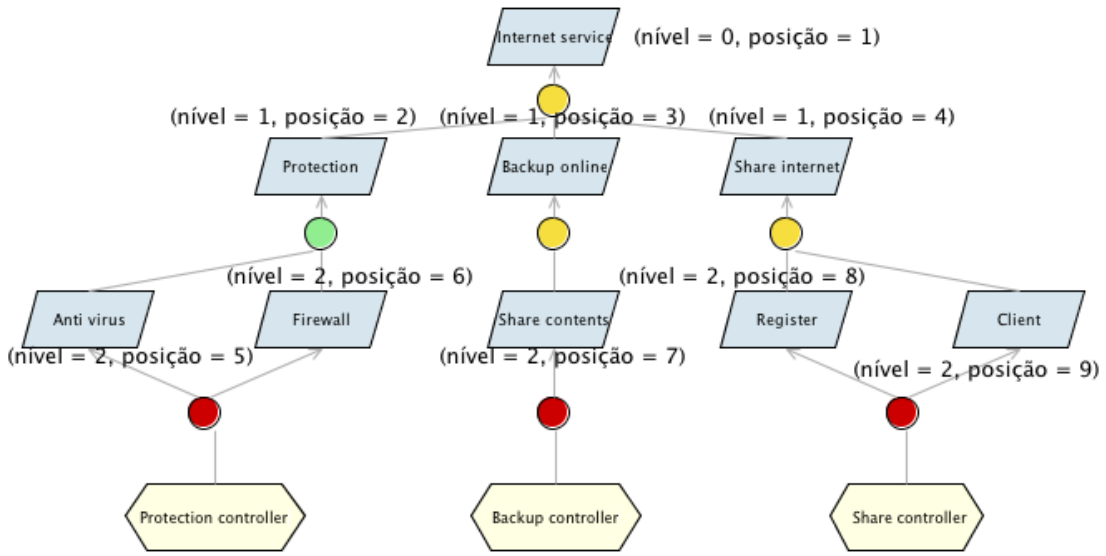


Figura 6.13: Posições dos goals num modelo KAOS.

Cada posição é calculada com base no nível da árvore e do seu índice de quando foi adicionado no modelo. Por exemplo, o *goal Protection* foi o segundo elemento a ser

adicionado e encontra-se no nível 1, sendo que a sua posição é 2. O cálculo genérico de cada posição é dada pela expressão *position = level + index*.

6.5.3 Restrições

Como já foi relatado na subsecção 6.4.1, o comando "validate model" permite obter erros/*warnings* através da semântica estática. A Tabela 6.5 mostra as restrições que foram implementadas, o seu *feedback* e o seu tipo.

Tabela 6.5: Restrições do SimpleKAOS.

Tipo	Descrição	Feedback
Erro	Um *goal* tem que ter nome único no sistema.	*Goal <name> with the position <position> already exists*.
Erro	Um *goal* não pode ter o nome vazio.	*Goal <name> have <number> sub goals without name at the positions <positions>*.
Erro	Um agente tem que ter nome único no sistema.	*Agent <name> with the parents <goals> already exists*.
Erro	Um agente não pode ter o nome vazio.	*Goals <goals> have a agent without name*.

Como se pode verificar na tabela, o *feedback* atribuído a cada restrição oferece mais detalhes do que somente o nome do elemento de um dado contexto para resolver o erro. Esta decisão deve-se ao facto que se o utilizador for um indivíduo com limitações visuais, o nome do elemento poderia não bastar para resolver o erro visto que o comando "find" irá procurar o primeiro elemento e termina a procura (mesmo que exista dois elementos com o mesmo nome). Por exemplo, se o primeiro erro for verificado, significa que existem dois *goals* no modelo. Usando a ferramenta GMF, é bastante fácil identificar os dois elementos com nome repetido dado que o utilizador está a manipular o modelo através de uma vista diagramática. Isso não acontece no contexto da dissertação, logo é necessário oferecer mais conteúdo no *feedback* de modo a que o utilizador tenha dado suficientes para proceder à resolução do erro.

Os comandos implementados nunca vão permitir que se insira um nome em branco ou em duplicado. No entanto, a fase de elicitação de requisitos é um processo colaborativo, e dado que a ferramenta também disponibiliza componente gráfica, um membro da equipa de desenvolvimento pode criar elementos em duplicado/vazio no editor diagramático. Caso aconteça, pode ser usado posteriormente o comando "validate model", e é fornecida a informação de como se pode resolver o problema.

Nos primeiros dois erros listados na tabela o problema pode ser resolvido usando o comando "find by position", dado que fornece as posições, e os últimos dois erros podem ser resolvidos recorrendo ao comando "find goal", graças ao retorno do nome dos *goals*.

6.5.4 Aplicação

O caso de estudo escolhido para a aplicação foi de um serviço de internet que pode ser incluído num serviço de *Internet Service Provider* de uma empresa de telecomunicações. O serviço de internet contém funcionalidades como protecção (anti vírus e *firewall*), *backup online* dos dados do computador e partilha do ponto de acesso da internet para outras pessoas. Com base na especificação do metamodelo e dos comandos do SimpleKAOS, o utilizador diz os comandos para modelar este modelo KAOS e a ferramenta executa as operações, fornecendo o *feedback* apropriado. Um cenário que ilustra a construção do modelo KAOS para o serviço de internet é mostrado na Tabela 6.6. Este cenário mostra o início da aplicação, a criação da *root*, os seus *sub-goals* e os agentes.

Tabela 6.6: Cenário de construção de um serviço de Internet em SimpleKAOS.

1. Utilizador: add root Internet service **Feedback**: Root Internet service has been added successfully. **2. Utilizador**: add goal refinement AND Protection **Feedback**: Goal Protection with refinement AND has been added successfully. **3. Utilizador**: add goal refinement AND Backup online **Feedback**: Goal Backup online with refinement AND has been added successfully. **4. Utilizador**: add goal refinement AND Share internet **Feedback**: Goal Share internet with refinement AND has been added successfully. **5. Utilizador**: find Protection **Feedback**: Goal Protection found successfully. **6. Utilizador**: add goal refinement OR Anti virus **Feedback**: Goal Anti virus with refinement OR has been added successfully.	**11. Utilizador**: add goal refinement AND Share contents **Feedback**: Goal Share contents with refinement AND has been added successfully. **12. Utilizador**: add agent Backup controller to Backup online **Feedback**: Agent Backup controller has been added successfully. **13. Utilizador**: find Share internet **Feedback**: Goal Share internet found successfully. **14. Utilizador**: add goal refinement AND Register **Feedback**: Goal Register with refinement AND has been added successfully. **15. Utilizador**: add goal refinement AND Client **Feedback**: Goal Client with refinement AND has been added successfully. **16. Utilizador**: add agent Share controller to Register **Feedback**: Agent Share controller has been added successfully.

7. Utilizador: add agent Protection controller **Feedback**: Agent Protection controller has been added successfully. **8. Utilizador**: add goal refinement OR Firewall **Feedback**: Goal Firewall with refinement OR has been added successfully. **9. Utilizador**: add agent Protection controller **Feedback**: Agent Protection controller has been added successfully. **10. Utilizador**: find Backup online **Feedback**: Goal Backup online found successfully.	**17. Utilizador**: add agent Share controller to Cliente **Feedback**: Agent Share controller has been added successfully. **18. Utilizador**: say total of goals **Feedback**: The KAOS Model has 9 goals. **19.Utilizador**: say sub goals name Internet service **Feedback**: Goal Internet service has a refinement AND with the following goals: Protection, Backup online and Share internet.

A Figura 6.14, apresenta a componente gráfica do cenário da Tabela 6.6. Na *framework* VoiceToModel os *goals* são representados em losango com cor azul e o nome do *goal* está contido no respectivo losango. Os refinamentos OR são constituídos em círculo com cor verde, os refinamentos AND em círculo com cor amarela e as ligações dos agentes para com os *goals* são representados em círculos com cor vermelha. Por fim, os agentes são apresentados com a forma de um hexágono com cor creme.

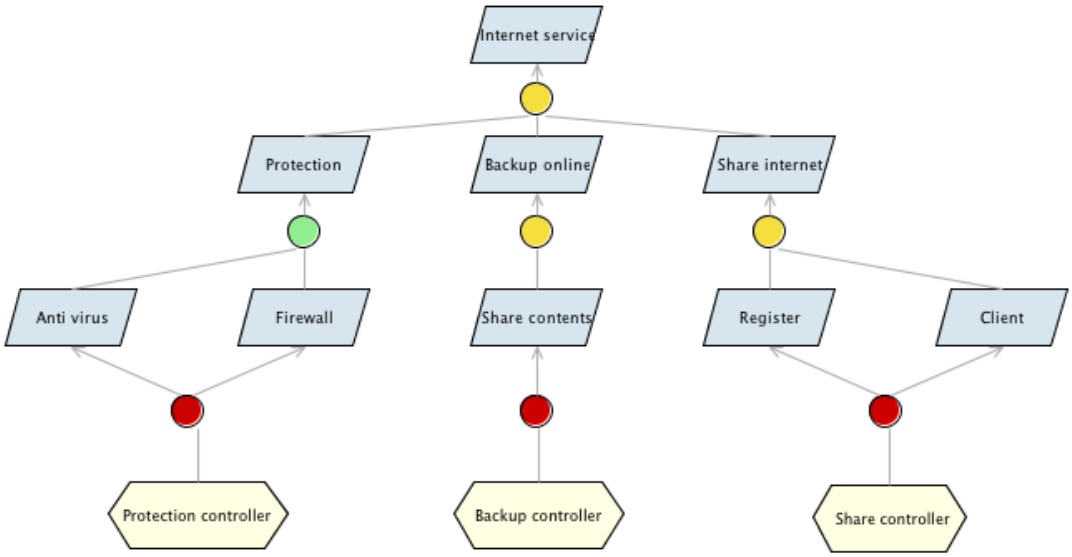


Figura 6.14: SimpleKAOS de serviço Internet gerado.

6.6 Modelo Conceptual

O modelo conceptual da aplicação VoiceToModel transforma os requisitos de um sistema para uma perspectiva *object-oriented* através dos conceitos de classe, atributo, associação e multiplicidades. Este modelo é uma versão simplificada do diagrama de classes, dado que não contém herança, composição, agregação e métodos. O objectivo foi implementar comandos e analisar se é possível construir um sistema robusto que permita mapear os requisitos num paradigma orientado a objectos.

Nas próximas subsecções serão descritas com mais detalhe a sintaxe abstracta, comandos e restrições do Modelo Conceptual.

6.6.1 Metamodelo

O metamodelo do Modelo Conceptual [OMG09] está especificado na Figura 6.15. O modelo conceptual contém **classifiers**. Um *classifier* é representado através de **classes** e **associations**. As *classes* são especializações de *classifiers* que pode conter **attributes** e **properties**. Uma *property* se for *share* significa que a relação entre classes é do tipo **associação**. A classe *MultiplicityElement* permite definir a multiplicidade de uma relação entre duas classes.

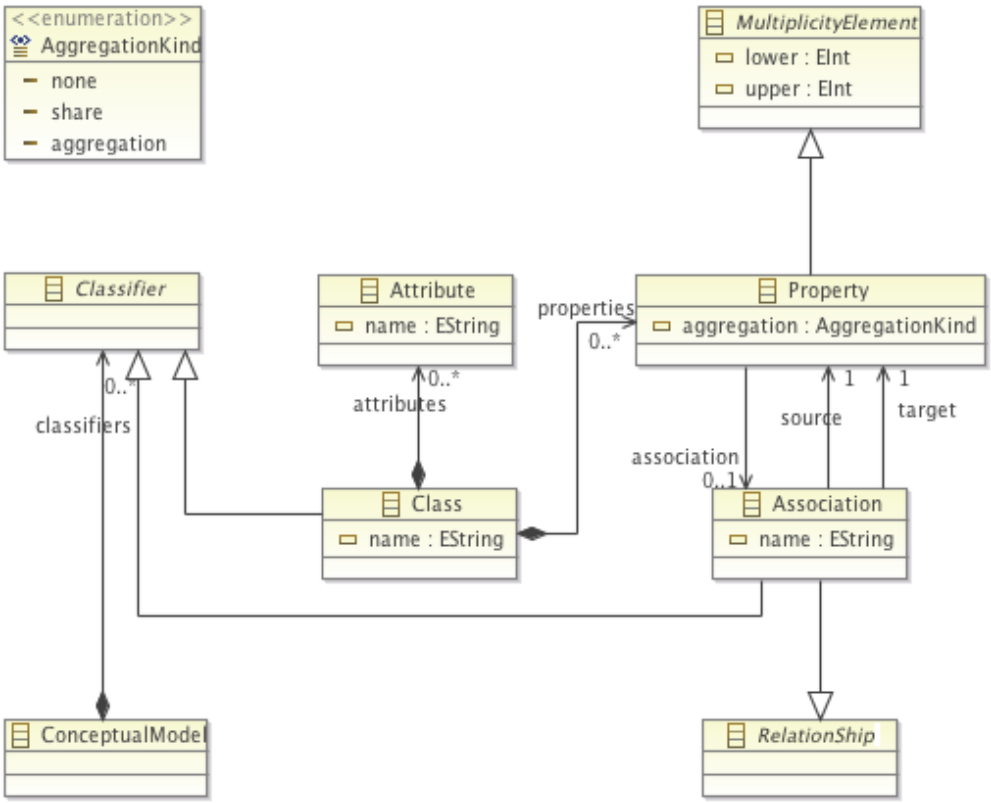


Figura 6.15: Metamodelo Modelo Conceptual.

6.6.2 Gramática e Comandos

Tal como o SimpleKAOS, a estratégia de geração dos comandos passou pela análise do seu metamodelo. Através dos conceitos *class*, *attribute*, *association* e *multiplicity element* apresentados no metamodelo do modelo conceptual, foram gerados comandos que permitem modificar o estado do modelo conceptual. A secção B.2 do Anexo B apresenta os comandos que estão anotados ao respectivo elemento do metamodelo.

A Figura 6.16 mostra a estrutura dos comandos que foram especificados e implementados.

```
grammar conceptualModel;

<conceptualModel>   =   add <add_element>              |
                        change <change_element>        |
                        delete <delete_element>        |
                        find                           |
                        say <describe>                 |
                        sleep mode (on | off)          |
                        repeat last feedback           |
                        validate model                 |
                        undo command                   |
                        describe commands              |
                        help mode (on | off)           |
                        <choice>                       |
                        back to project controller     |
                        <NULL>                         ;

<add_element>       =   class | attribute | association <multiplicity> | <NULL>;

<multiplicity>      =   many to many | many to one | many to optional | one to one |
                        one to many | one to optional | optional to optional | optional to many |
                        optional to one | <NULL>;

<change_element>    =   class name | attribute name | association multiplicity <multiplicity> |
                        <NULL>;

<delete_element>    =   class | attribute | association | <NULL>;

<describe>          =   number of classes | number of attributes | classes name | attributes name |
                        associations | actual class | <NULL>;

<choice>            =   yes | no | <NULL>;

public <command>    =   <conceptualModel>;
```

Figura 6.16: Gramática Modelo Conceptual.

Com base nos comandos implementados para construir um modelo conceptual, o utilizador pode inserir classes, atributos, relacionar classes através de associações e obter informações sobre o estado do modelo. O Modelo Conceptual é representado como uma lista de classes C = (CL, AT, AS), em que:

- CL $\in$ {*class*};
- AT $\in$ {*attribute*};
- AS $\in$ {*association*}.

A Tabela 6.7 exemplifica 10 comandos para auxiliar a modelação através da voz. A secção A.2 do Anexo A disponibiliza todos os comandos implementados. De forma análoga ao SimpleKAOS, na primeira coluna é apresentado o comando a ser executado e na segunda coluna a sua descrição. O *feedback* (via voz) é retornado quando é executada uma operação. A pós-condição especifica o resultado da operação. Na última coluna é apresentada a complexidade no pior caso. Por exemplo, quando se adiciona um atributo a

complexidade pode ser constante, caso tenha sido usado anteriormente o comando "find class" ou "add class". Se for especificado o nome da classe no comando que adiciona um atributo, a complexidade irá ser linear.

Tabela 6.7: Exemplos dos comandos do Modelo Conceptual.

Comando	Descrição	Feedback	Pós-condição	Complexidade
add class <name>	Adiciona uma classe no modelo.	*Class <name> has been added successfully*.	Classe adicionada.	O(1)
add attribute <name>	Adiciona um atribute numa classe.	*Attribute <name> in class <name> has been added successfully*.	Atributo adicionado.	O(1)
add association (many | one | optional) to (many | one | optional) <class1> to <class2>	Adiciona uma associação entre duas classes.	*Association between <class1> and <class2> has been added successfully*.	Associação adicionada.	O(|CL|)
change class name <old> to <new>	Muda o nome de uma classe.	*Class <old> was changed successfully to <new>*.	Nome de classe alterada.	O(|CL|)
change attribute name <old> to <new>	Muda o nome de um atributo.	*Attribute <old> in class <class> was changed successfully to <new>*.	Nome do atributo alterado.	O(|AT|)
delete class <name>	Apaga uma classe no modelo.	*Class <name> has deleted successfully*.	Classe removida.	O(|CL|)
delete attribute <attribute> to <class>	Apaga um atributo de uma classe.	*Attribute <attribute> has deleted successfully*.	Atributo removido.	O(|CL| + |AT|)
find <class>	Procura uma classe e foca-a.	*Class <class> found successfully*.	Classe encontrada.	O(|CL|)
say number of classes	Indica o número total de classes que há no modelo.	*Conceptual Model has <number> classes*.	Número de classes retornado.	O(|CL|)
say classes name	Indica os nomes das classes do modelo.	Exemplo: *Conceptual Model has the following classes name: <classes>*.	Nomes das classes retornados.	O(|CL|)

O conceito de dinâmica dos parâmetros é o mesmo apresentado no SimpleKAOS da subsecção 6.5.2. Ao criar um atributo através do comando "add attribute" o utilizador pode especificar ou omitir o nome da classe, como se pode verificar na Figura 6.17. O primeiro cenário adiciona uma classe e de seguida um atributo sem mencionar o contexto da classe (está implícito), e o segundo cenário especifica o nome da classe quando se adiciona o atributo, não tirando proveito da flexibilidade dos parâmetros.

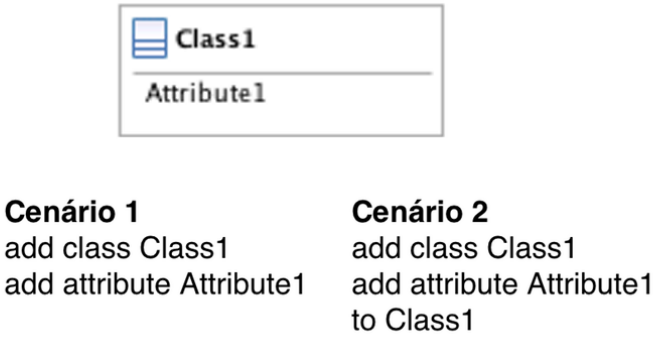


Figura 6.17: Exemplo do uso da omissão de parâmetros no Modelo Conceptual.

O comando "find <class>" também tem a mesma semântica do SimpleKAOS. A Figura 6.18 mostra a criação de duas classes com associação entre elas usando as duas vertentes do uso dos comandos. No primeiro cenário são adicionadas duas classes e uma associação usando todos os parâmetros (total de 3 comandos) e no segundo cenário é utilizado o comando "find <class>" para omitir o nome da primeira classe quando se faz a associação (total de 4 operações). Usando o comando "find <class>" torna-se mais prático caso seja necessário adicionar várias associações à primeira classe, evitando a repetição do seu nome no comando "add association".

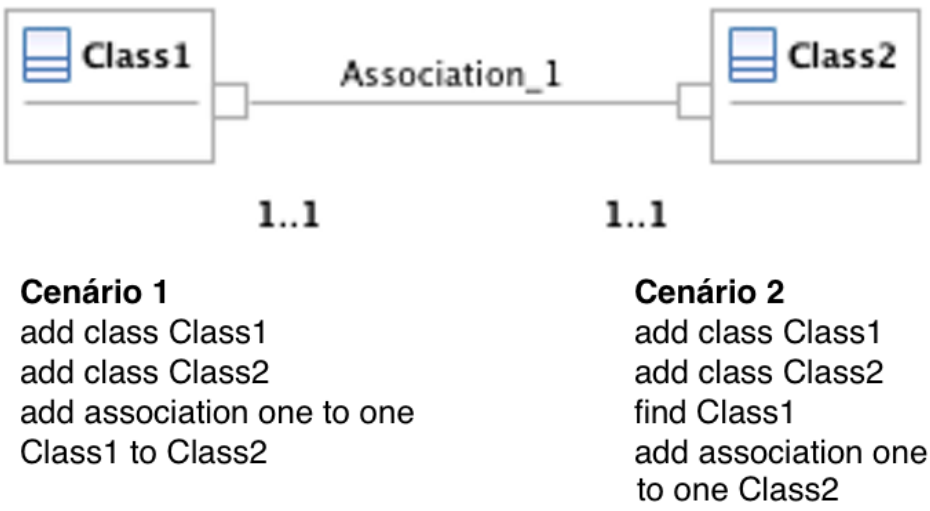


Figura 6.18: Exemplo do uso do comando find no Modelo Conceptual.

Quando se adiciona uma associação entre duas classes, o nome da associação é gerado automaticamente e não é editável através dos comandos via voz. Esta limitação ocorre devido à flexibilidade dos parâmetros, porque caso se incluísse o nome da associação no comando "add association" e o utilizador omitisse um parâmetro, o sistema não conseguiria distinguir se o utilizador estava a omitir o nome da primeira classe ou o nome da associação. Apesar disso, o sistema foi implementado já com a possibilidade de especificar o nome da associação para que futuras versões possam dar uso dessa funcionalidade.

6.6.3 Restrições

O comando "validate model" emite mensagens via voz ao utilizador devido a erros e/ou *warnings* que exista no modelo conceptual criado. A Tabela 6.8 contém as restrições que foram especificadas em EVL, o seu tipo e o seu *feedback*.

Tabela 6.8: Restrições do Modelo Conceptual.

Tipo	Descrição	Feedback
Erro	Uma classe tem que ter nome único no sistema.	*Class <name> already exists*.
Erro	Um atributo de uma classe tem que ser único.	*Class <class name> has the following attributes name repeated: <attributes name>*.
Erro	Não pode haver classes sem nome.	*There are classes without name*.
Erro	Não pode haver atributos sem nome.	*Class <name> has attributes without name*.

Em termos de comparação com a tabela de restrições do SimpleKAOS, é possível verificar que não existe o conceito de posição na lista de comandos. Isso deveu-se ao facto de o modelo conceptual não ser representado como uma árvore e, deste modo, a posição entre as classes pode ser alterável muito facilmente. Devido a este motivo, o *feedback* da tabela de restrições não usa a noção de posição entre classes.

Os dois primeiros erros podem ser resolvidos através da especificação do nome da classe num comando "change"/"delete"/"find" e os últimos dois erros podem ser corrigidos por um membro da equipa de desenvolvimento que use a vista diagramática (não é possível inserir nomes vazios através dos comandos implementados).

6.6.4 Aplicação

O caso de estudo usado para exemplificar o uso dos comandos do modelo conceptual é o mesmo do SimpleKAOS, isto é, um serviço de internet que contém funcionalidades como protecção, partilha de internet e *backup* dos dados do computador via *online*. Através do metamodelo e dos comandos implementados para o modelo conceptual, a Tabela 6.9 mostra o cenário de construção do serviço de internet.

Tabela 6.9: Cenário de construção de um serviço de Internet para o Modelo Conceptual.

1. Utilizador: add class Internet service **Feedback**: Class Internet service has been added successfully. **2. Utilizador**: add class Protection **Feedback**: Class Protection has been added successfully. **3. Utilizador**: add class Backup online **Feedback**: Class Backup online has been added successfully. **4. Utilizador**: add class Share internet **Feedback**: Class Share internet has been added successfully. **5. Utilizador**: add class Client **Feedback**: Class Client has been added successfully. **6. Utilizador**: find Internet service **Feedback**: Class Internet service found successfully. **7. Utilizador**: add association one to one Protection **Feedback**: Association between Internet service and Protection has been added successfully. **8. Utilizador**: add association one to one Share internet **Feedback**: Association between Internet service and Share internet has been added successfully.	**9. Utilizador**: add association one to one Backup online **Feedback**: Association between Internet service and Backup online has been added successfully. **10. Utilizador**: add association one to many Share internet to Client **Feedback**: Association between Share internet and Client has been added successfully. **11. Utilizador**: say number of classes **Feedback**: Conceptual Model has 5 classes. **12. Utilizador**: say classes name **Feedback**: Conceptual Model has the following classes name: Internet service, Protection, Backup online, Share internet and Client. **13. Utilizador**: say associations Internet service **Feedback**: Class Internet service has the following associations: one to one for class Protection, one to one for class Share internet and one to one for class Backup online.

A Figura 6.19 mostra o cenário da Tabela 6.9 através de notação gráfica. Na ferramenta VoiceToModel as classes são compostas em figuras rectangulares, com o título no topo, e os atributos apresentados em forma de lista (debaixo do nome da classe). Uma associação é representada por uma linha, directamente relacionada com dois pequenos quadrados, que permitem conectar a associação entre duas classes. O nome de uma associação é apresentado pelo formato "Association_N+1", sendo N o número total de associações já criadas no modelo. Por fim, as multiplicidades são apresentadas da seguinte forma: 1..1 - um para um; 1..10.000 - um para muitos; e 0..1 - opcional.

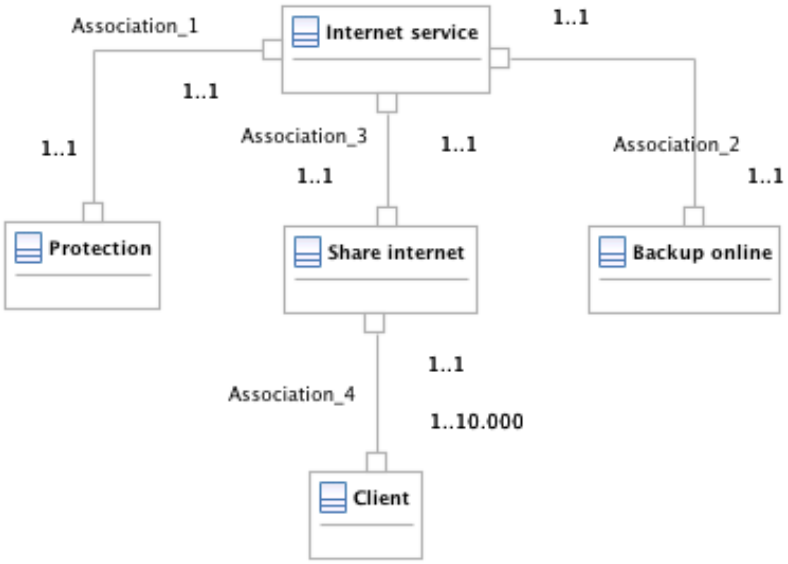


Figura 6.19: Modelo Conceptual de serviço Internet gerado.

6.7 Modelo de Features

Modelo de *Features* é um modelo de variabilidade que permite gerar por exemplo uma linha de produtos dado um domínio por intermédio dos conceitos *feature mandatory/optional*, *group OR/Aternative* e *constraint requires/excludes*. É o único modelo dos três existentes no VoiceToModel que não foi simplificado porque a complexidade do modelo não o justificava, pois foi usada a versão mais simples do modelo de *features*.

As próximas subsecções vão mostrar como foi construído o metamodelo, os comandos e as restrições.

6.7.1 Metamodelo

Figura 6.20 mostra o metamodelo do modelo de *features* [CHE05]. Este contém uma **root**, especificado abstractamente como *FeatureNode*, representando o conceito principal do modelo. É possível decompor a *root* em **Solitary** ou em **Group**, que são representadas como *Member*. A classe *Solitary* pode ser instanciada como *Optional* (*lower* = 0, *upper* = 1) ou como *Mandatory* (*lower* = 1, *upper* = 1). Um *Group* pode ser do tipo *OR* (*lower* = 1, *upper* = ∞[2]), ou seja, podem ser seleccionadas uma ou mais *features* (dependendo do número de *features* dentro do grupo) ou podem ser do tipo *Alternative* (*lower* = 1, *upper* = 1) em que apenas uma *feature* é seleccionada no grupo. Um **Grouped** apenas é associado a grupos.

A classe **Constraint** pode ser do tipo *Requires* ou *Excludes* e tem como *source* e *target* uma *feature Solitary* ou *Grouped*.

[2]A definição de infinito representa o número máximo de *features* que um grupo tem.

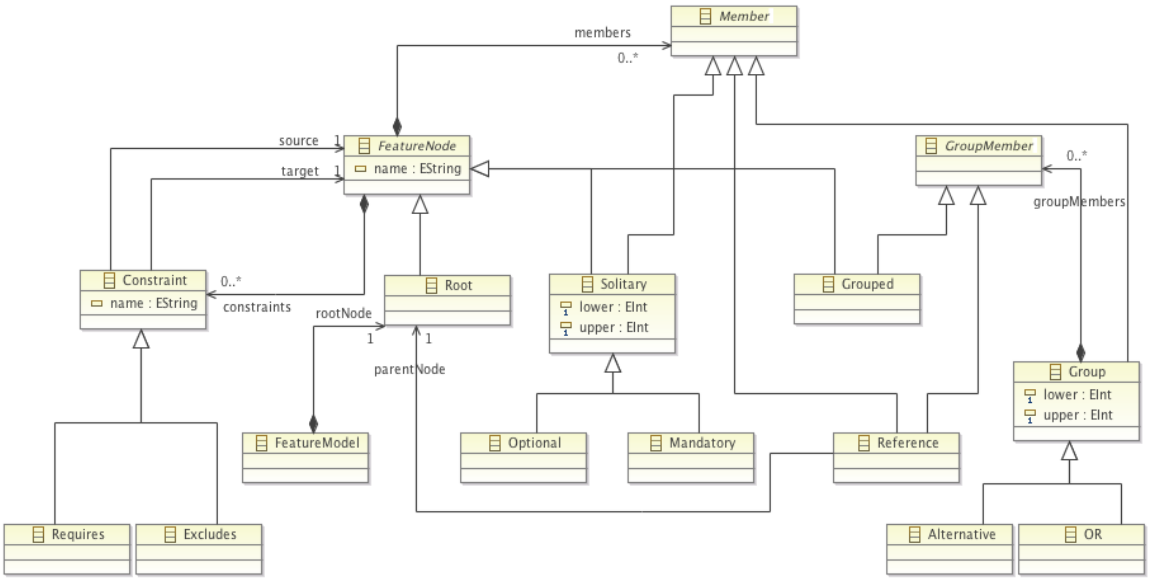


Figura 6.20: Metamodelo Modelo de *Features*.

6.7.2 Gramática e Comandos

O método como foi gerada a gramática para o modelo de *features* foi idêntico ao SimpleKAOS e modelo conceptual. Foi realizada uma análise dos conceitos definidos do metamodelo, extraindo a informação que vai permitir definir comandos de modo a gerar um modelo de *features* através da Figura 6.21, tais como: *feature* root, obrigatória, opcional, *grouped*, grupo OR, grupo *alternative*, *constraint requires* e *excludes*. A secção B.3 do Anexo B apresenta os comandos que estão anotados aos respectivos conceitos do metamodelo de *features*.

```
grammar featureModel;

<featureModel>      =   add (<feature> | group <group> | constraint <constraint>)   |
                        change <change_element>                                     |
                        delete <element>                                            |
                        find                                                        |
                        find by position                                            |
                        say <describe>                                              |
                        sleep mode (on | off)                                       |
                        repeat last feedback                                        |
                        validate model                                              |
                        undo command                                                |
                        describe commands                                           |
                        help mode (on | off)                                        |
                        <choice>                                                    |
                        back to project controller                                  |
                        <NULL>                                                      ;

<feature>           =   root | mandatory | optional | feature | <NULL>;

<group>             =   or | alternative | <NULL>;

<constraint>        =   requires | excludes | <NULL>;

<change_element>    =   name | type of <element> | <NULL>;

<element>           =   feature | group | constraint | <NULL>;

<describe>          =   number of sub features | sub features name | constraints |
                        total of mandatory feature | total of optional feature |
                        total of features | total of constraints | actual feature |
                        <NULL>;

<choice>            =   yes | no | <NULL>;

public <command>    =   <featureModel>;
```

Figura 6.21: Gramática Modelo de *Features*.

Os comandos permitem alterar o estado do modelo ("add", "change", "delete") ou obter informação sobre o conteúdo das *features* e como estão relacionadas, com base nas ligações e *constraints*, através dos comandos "find" e "say". O modelo de *features* é representado como um grafo MF = (F, G, C), em que:

- F ∈ {*root*, *mandatory*, *optional*, *grouped*};
- G ∈ {*group OR*, *group Alternative*};
- C ∈ {*constraint requires*, *constraint excludes*}.

Segundo a especificação da gramática apresentada Figura 6.21, a Tabela 6.10 ilustra 10 comandos. A secção A.3 do Anexo A disponibiliza todos os comandos implementados.

Na primeira coluna está o comando a ser executado e na segunda coluna a respectiva descrição. O *feedback* permite ao utilizador ser notificado sobre o resultado da operação via voz. A pós-condição especifica qual o resultado expectável da respectiva operação. Na última coluna é apresentada a complexidade no pior caso. Quando é adicionada a *root*, a complexidade do algoritmo é constante. Para adicionar um grupo, a complexidade pode não ser constante porque a *feature* pode ter *sub features* (e.g., obrigatório ou opcional). Deste modo, é necessário transformar as *sub features* em simples *features* relacionadas com o tipo do grupo (OR ou *Alternative*), tornando a complexidade linear.

Tabela 6.10: Exemplos dos comandos do Modelo de *Features*.

Comando	Descrição	Feedback	Pós-condição	Complexidade
add root <name>	Adiciona uma *feature* no modelo.	*Root <name> has been added successfully.*	*Feature* adicionada.	O(1)
add mandatory <name>	Adiciona uma *feature* obrigatória.	*Feature <name> has been added successfully.*	*Feature* obrigatória adicionada.	O(1)
add optional <name>	Adiciona uma *feature* opcional.	*Feature <name> has been added successfully.*	*Feature* opcional adicionada.	O(1)
add group OR <name>	Adiciona um grupo OR.	*Group OR of feature <name> has been added successfully.*	Grupo OR adicionado.	O(|F| + |G|)
add constraint requires <feature1> to <feature2>	Adiciona uma restrição *requires*.	*Constraint <feature1> requires <feature2> has been added successfully.*	Restrição *requires* adicionada com sucesso.	O(|F| + |G|)

change name <old> to <new>	Muda o nome de uma *feature*.	*Feature <old> was changed successfully to <new>.*	Nome alterado.	O(|F| + |G|)
delete feature <name>	Apaga uma *feature* no modelo.	*Feature <name> was deleted successfully.*	*Feature* removida.	O(|F| + |G| + |C|)
find <name>	Procura uma *feature* e instancia como *sub-root*.	*Feature <name> found successfully.*	*Feature* encontrada.	O(|F| + |G|)
find by position <number >	Procura uma *feature* e instancia como *sub-root* através da sua posição.	*Feature <name> with the position <number> found successfully.*	*Feature* encontrada.	O(|F| + |G|)
say total of features	Indica o número total de *features* que há no modelo.	*Feature Model has <number> features.*	Número de *features* retornado.	O(|F| + |G|)

É seguida a mesma lógica de implementação dos comandos do SimpleKAOS e do modelo conceptual. Isto é, é possível adicionar a *root* e as suas *sub features* através dos comandos "add root" e de seguida "add mandatory/optional/group", em vez da sequência: "add root", "find <name of root>" e "add mandatory/optional/group", como é possível verificar nos dois cenários da Figura 6.22. O primeiro cenário usa a flexibilidade dos parâmetros (nunca especificando a *feature* pai) e o segundo cenário explicita o nome da *root* em todos os comandos "add mandatory".

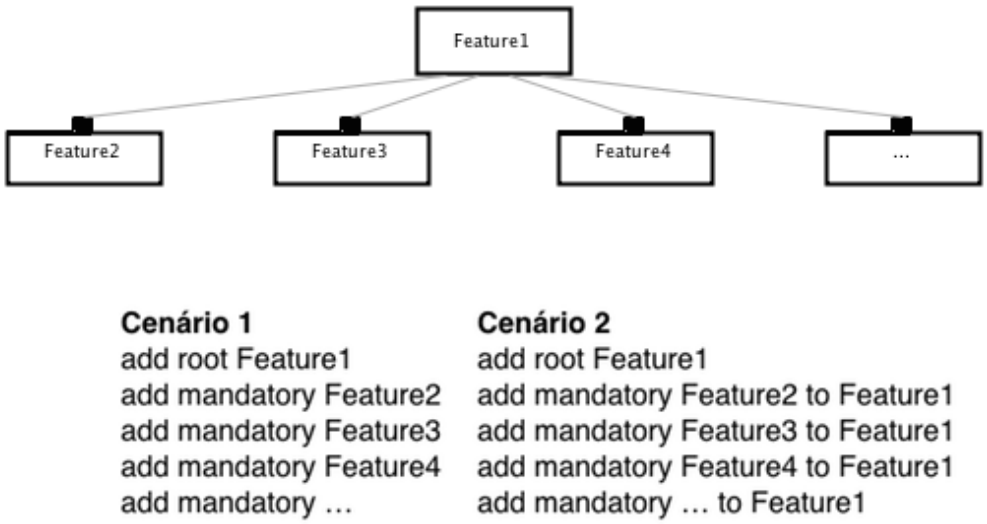


Figura 6.22: Exemplo do uso da omissão de parâmetros no Modelo de *Features*.

O comando "find <feature>" foca uma *feature* de modo a que comandos imediantamente seguir ao "find", como "change"/"delete"/"say", não sejam necessário especificar o último argumento. Na Tabela 6.10, se o "find" tivesse sido usado então o comando "change feature name" poderia omitir o parâmetro <old>, passando a ser "change feature <new>".

A Figura 6.23 mostra dois cenários com o "find" e sem o "find". No primeiro cenário são usados três comandos: "find" e dois "add mandatory" e no segundo cenário são usadas duas operações que devolvem o mesmo resultado do primeiro cenário. A flexibilidade dos parâmetros usados no primeiro cenário é apropriada quando se adiciona mais que uma *feature* de forma consecutiva.

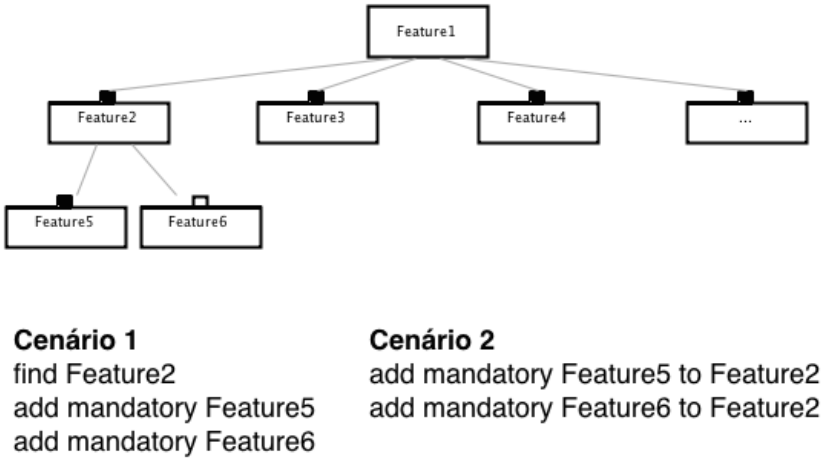


Figura 6.23: Exemplo do uso do comando find no Modelo de *Features*.

O comando "find by position <number>" tem o mesmo significado do comando "find by position" do SimpleKAOS, ou seja, dada a posição do grafo da *feature*, foca a *feature* como *sub root*, tal como faz o comando "find <name>". A Figura 6.24 demonstra um exemplo de como é atribuído uma posição a cada *feature*, com base no cálculo *position* = *level* + *index*.

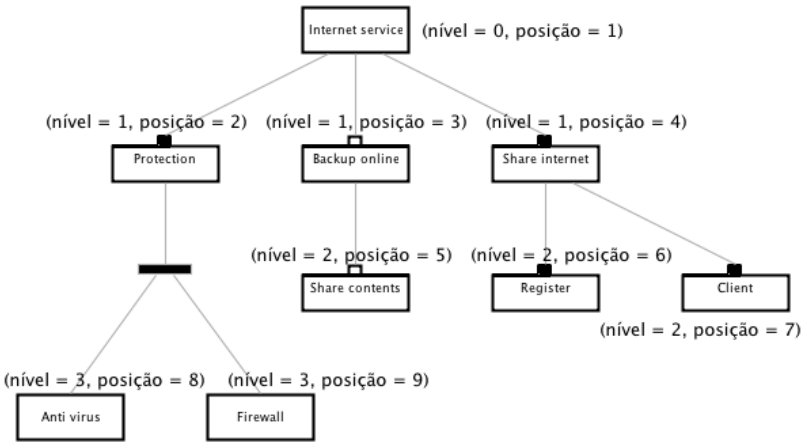


Figura 6.24: Posições das *features* num Modelo de *Features*.

6.7.3 Restrições

Tal como o SimpleKAOS e o modelo conceptual, com base no comando "validate model" é possível ouvir mensagens para o utilizador, de modo a que saiba os erros e/ou *warnings* que o modelo contenha. A Tabela 6.11 contém as restrições que foram especificadas em EVL, o seu tipo e o seu *feedback*.

Tabela 6.11: Restrições do Modelo de *Features*.

Tipo	Descrição	Feedback
Erro	Uma *feature* tem que ter nome único no sistema.	*Feature <name> with the position <position> already exists*.
Erro	Não pode haver *features* sem nome.	*Feature <name> has <number> sub features without name at the positions <positions>*.
Erro	Não pode haver *features* com que tenha em simultâneo *sub features* e grupos.	*Feature <name> can not have as a children features and group simultaneously*.
Erro	Uma *feature* não pode ter mais que um grupo.	*Feature <name> can not have more than one group*.
Warning	Um grupo deve ter *sub features*.	*Group of feature <name> should have sub features*.
Erro	Não pode haver duas ou mais restrições para as duas *features*.	*The constraint <source> to <target> has more than one instance*.
Erro	Uma restrição *requires* não pode apontar para a mesma *feature*.	*The constraint requires in the feature <name> cannot point to the same feature*.
Warning	Uma restrição *requires* não deve apontar para o seu antecessor.	*Constraint requires of feature <name> should not point to antecessor <antecessor name>*.
Warning	A *root* não deve ter restrições *requires*.	*Root <name> should not have constraint requires*.
Erro	Uma restrição *excludes* não pode apontar para a mesma *feature*.	*The constraint excludes in the feature <name> cannot point to the same feature*.
Warning	Uma restrição *excludes* não deve apontar para o seu antecessor.	*Constraint excludes of feature <name> should not point to antecessor <antecessor name>*.
Warning	A *root* não deve ter restrições *excludes*.	*Root <name> should not have constraint excludes*.

Foi usada a mesma analogia apresentada no SimpleKAOS, no que diz respeito à forma como é fornecido o *feedback* ao utilizador dos erros e *warnings* (graças à sua estrutura como árvore). São fornecidos dados da posição caso uma *feature* esteja duplicada ou sem nome. Os comandos do modelo de *features* também não permitem a inserção de *features* sem nome, mas num processo colaborativo entre membros da equipa de desenvolvimento, essa situação pode ocorrer caso seja usado a vista diagramática da aplicação VoiceToModel ou a importação do XMI do modelo em questão.

Os dois primeiros erros podem ser resolvidos através do comando "find by position", e os restantes erros e *warnings* através do comando "find".

6.7.4 Aplicação

Seguindo o mesmo exemplo do caso de estudo do serviços de internet do SimpleKAOS e modelo conceptual, a Tabela 6.12 mostra o cenário do serviço de internet com as funcionalidades de protecção, *backup online* da informação que estiver alocada no computador e a partilha do ponto de acesso da internet usando o modelo de *features*.

Tabela 6.12: Cenário de construção de um serviço de Internet para o Modelo de *Features*.

1. Utilizador: add root Internet service **Feedback**: Root Internet service has been added successfully. **2. Utilizador**: add mandatory Protection **Feedback**: Feature Protection has been added successfully. **3. Utilizador**: add optional Backup online **Feedback**: Feature Backup online has been added successfully. **4. Utilizador**: add mandatory Share internet **Feedback**: Feature Share internet has been added successfully. **5. Utilizador**: add group OR Protection **Feedback**: Group OR of feature Protection has been added successfully. **6. Utilizador**: add feature Anti virus **Feedback**: Feature Anti virus has been added successfully.	**8. Utilizador**: find Backup online **Feedback**: Feature Backup online found successfully. **9. Utilizador**: add optional Share contents **Feedback**: Feature Share contents has been added successfully. **10. Utilizador**: add mandatory Register to Share internet **Feedback**: Feature Register has been added successfully. **11. Utilizador**: add mandatory Client to Share internet **Feedback**: Feature Client has been added successfully. **12. Utilizador**: say total of features **Feedback**: Feature Model has 9 features. **13. Utilizador**: say total of mandatory features **Feedback**: Feature Model has 4 mandatory features.

7. Utilizador: add feature Firewall **Feedback**: Feature Firewall has been added successfully.	**14. Utilizador**: say total of optional features features **Feedback**: Feature Model has 2 optional features.

A transformação do cenário da Tabela 6.12 pode ser visualizado na Figura 6.25. Na aplicação VoiceToModel as *features* obrigatórias são representadas por pequenos quadrados com fundo preto no topo da *feature* e as *features* opcionais com quadrados de fundo branco. Um grupo OR é constituído através de um rectângulo com fundo preto e um grupo *Alternative* com fundo branco. Por fim, as restrições *requires* e *excludes* apresentam-se em linhas em tracejado, entre duas *features*.

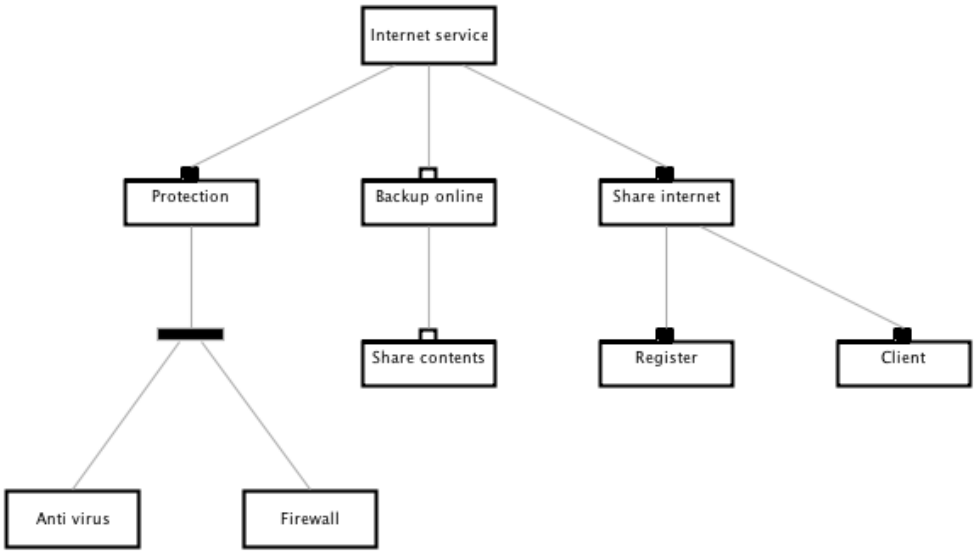


Figura 6.25: Modelo de *Features* de serviço Internet gerado.

6.8 Controlador de projectos

O controlador de projectos é uma funcionalidade extra que permite criar/abrir/apagar projectos e ficheiros de modelos de requisitos num ambiente *workspace*. Esta funcionalidade permite reforçar o conceito de **acessibilidade** devido ao facto de oferecer a possibilidade de gerir o directório de projectos através da voz.

6.8.1 Workspace do Eclipse

Para que fosse possível criar/abrir/apagar projectos e ficheiros de modelos foi necessário analisar como funciona o *workspace* em ambiente Eclipse. A base do *workspace* situa-se no objecto *IResource*. Com o **IResource** é possível criar **projectos** e **ficheiros**. Os projectos são criados usando o objecto *IProject* e os ficheiros com o objecto *IFile*. A Figura 6.26 apresenta a hierarquia do *workspace* através do Eclipse.

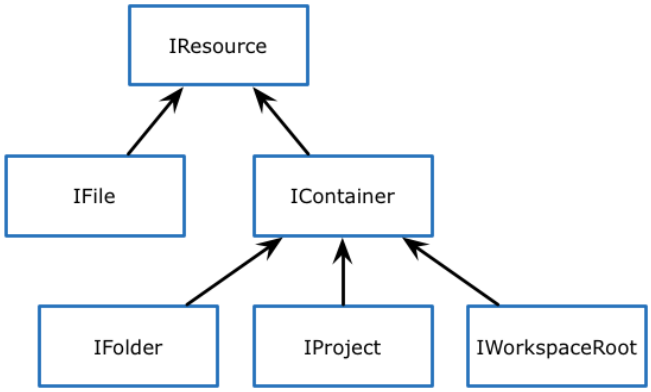


Figura 6.26: Hierarquia do *workspace* do Eclipse [Cre13].

6.8.2 Gramática e Comandos

A lista de comandos criada para o controlador de projectos teve como base a experiência de criação de projectos em ambiente Eclipse e de outros programas. É neste contexto que se inicia e termina a aplicação VoiceToModel, baseado nos comandos "start program" e "end program". Ao criar um projecto, através do comando "create project" é atribuído um identificador ao projecto. Esse identificador é usado para depois abrir/apagar um projecto. A mesma implementação foi feita para criar, abrir e apagar modelos.

Para além da manipulação do *workspace*, também é possível obter informação sobre um projecto, isto é, quantos modelos tem e o seu identificador usando o comando "say project details" e através do comando "say number of projects" informar do número de projectos criados. A Figura 6.27 sistematiza todos os comandos criados para o controlador de projectos.

```
grammar projectController;

<projectController>   =   create <type>              |
                          delete <type>              |
                          open <type>                |
                          say <describe>             |
                          sleep mode (on | off)      |
                          repeat last feedback       |
                          describe commands          |
                          help mode (on | off)       |
                          <choice>                   |
                          <NULL>                     ;

<type>                =   project | feature model | conceptual model | KAOS model | <NULL>;

<describe>            =   current model | project details | number of projects | <NULL>;

<choice>              =   yes | no | <NULL>;

public <command>      =   start [program] | <projectController> | end [program] | <NULL>;
```

Figura 6.27: Gramática Controlador de Projectos.

A Tabela 6.13 apresenta 10 comandos do controlador de projectos. A secção A.4 do Anexo A.4 disponibiliza todos os comandos implementados.

Na primeira coluna é apresentado o comando seguido da sua descrição. Na terceira coluna é fornecido o *feedback* que o utilizador ouve quando executa a operação e, por fim, na última coluna é apresentada a pós-condição expectável da operação.

Tabela 6.13: Exemplos dos comandos do Controlador de Projectos.

Comando	Descrição	Feedback	Pós-condição
create project	Cria um projecto.	*Project number <number> has been created successfully.*	Projecto criado.
create kaos model	Cria um Modelo KAOS associado a um projecto.	*KAOS Model number <number> has been created successfully.*	Modelo KAOS criado.
create conceptual model	Cria um Modelo Conceptual associado a um projecto.	*Conceptual Model number <number> has been created successfully.*	Modelo Conceptual criado.
create feature model	Cria um Modelo de *Features* associado a um projecto.	*Feature Model number <number> has been created successfully.*	Modelo de *Features* criado.
open project <number>	Abre um projecto dado o seu número.	*Project number <number> is open.*	Projecto aberto.
open kaos model <number>	Abre um Modelo KAOS dado o seu número.	*KAOS Model number <number> is open. The commands of KAOS Model are available to start modeling.*	Modelo KAOS aberto.
open conceptual model <number>	Abre um Modelo Conceptual dado o seu número.	*Conceptual Model number <number> is open. The commands of Conceptual Model are available to start modeling.*	Modelo Conceptual aberto.
open feature model <number>	Abre um Modelo de *Features* dado o seu número.	*Feature Model number <number> is open. The commands of Feature Model are available to start modeling.*	Modelo de *Features* aberto.
say number of projects	Informa o número total de projectos que estão criados.	*Workspace has <number> projects.*	Número total de projectos devolvidos.
say project details	Informa de quantos modelos de requisitos estão associados ao projecto que está aberto e o seu respectivo identificador.	*Project number <number> has <number> KAOS Models with the numbers <numbers>, <number> Conceptual Models with the numbers <numbers> and <number> Feature Models with the numbers <numbers>.*	Informação sobre o projecto retornado.

Para o utilizador criar um projecto e iniciar a modelação de um Modelo KAOS/Conceptual/*Features* deve-se em primeiro lugar criar um projecto, usando o "create project", criar um modelo de requisitos através do comando "create kaos/conceptual/feature model" e por fim usar o comando open para abrir o modelo de requisitos.

6.8.3 Aplicação

A aplicação de exemplo escolhida para o controlador de projectos passa por criar um modelo de *features*, apagá-lo, criar três modelos de requisitos e abrir o segundo modelo de *features*. O cenário de utilização encontra-se presente na Tabela 6.14.

Tabela 6.14: Cenário de utilização do controlador de projectos.

1. Utilizador: start program **Feedback**: Start VoiceToModel. Workspace has zero projects. Please use the command create project to create a new project. **2. Utilizador**: create project **Feedback**: Project number 1 has been created successfully. **3. Utilizador**: create feature model **Feedback**: Feature Model number 1 has been created successfully. **4. Utilizador**: delete feature model two **Feedback**: Feature Model number 2 not found. **5. Utilizador**: delete feature model one **Feedback**: Feature Model number 1 has been deleted successfully. **6. Utilizador**: create feature model **Feedback**: Feature Model number 2 has been created successfully. **7. Utilizador**: create kaos model **Feedback**: KAOS Model number 1 has been created successfully. **8. Utilizador**: create conceptual model **Feedback**: Conceptual Model number 1 has been created successfully. **9. Utilizador**: say project details **Feedback**: Project number 1 has the following numbers: 1 Feature Model with the number 2, 1 Conceptual Model with the numbers 1 and 1 KAOS Model with the number 1. **10. Utilizador**: open feature model two **Feedback**: Feature Model number 2 is open. The commands of Feature Model are available to start modeling.

Na Figura 6.28 é possível visualizar a geração dos comandos da Tabela 6.14. No lado esquerdo encontram-se os projectos e ficheiros de modelos de requisitos criados e no lado esquerdo encontra-se o modelo de requisito actualmente aberto.

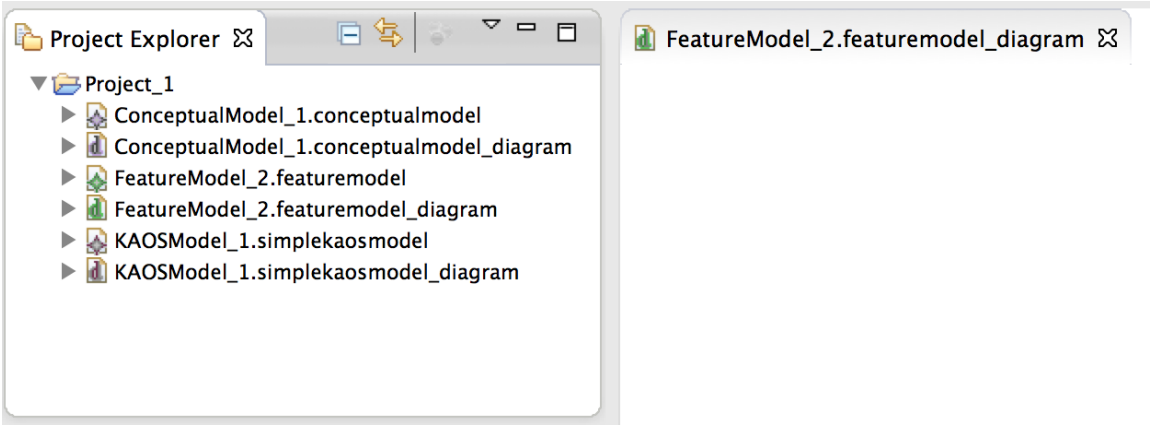


Figura 6.28: Exemplo de aplicação do controlador de projectos.

6.9 Técnicas adicionais

Durante o ciclo de implementação da ferramenta foi necessário recorrer a várias técnicas com o objectivo de solucionar diversas questões que vieram surgindo ao longo do trabalho. A ordem de implementação das técnicas que vão ser explicadas nas próximas subsecções é por ordem cronológica.

6.9.1 Algoritmo de procura

Quando o utilizador diz o nome de um elemento com o objectivo de manipular a informação já contida no modelo de requisito, é necessário realizar uma procura para obter o objecto do elemento em específico. Nos modelos abordados nesta dissertação, dois deles têm um comportamento semelhante (Modelo KAOS e de *Features*), dado que podem ser representados como uma árvore[3]. Na teoria de grafos existem vários tipos de procura em grafo, mas no contexto da dissertação apenas foram considerados a procura por largura e profundidade, como se pode verificar nas Figuras 6.29 e 6.30.

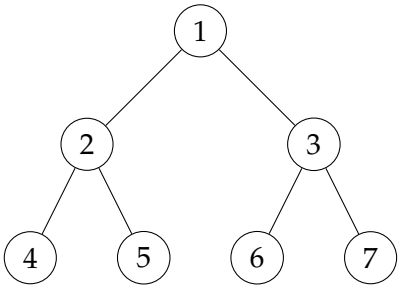


Figura 6.29: Procura por largura.

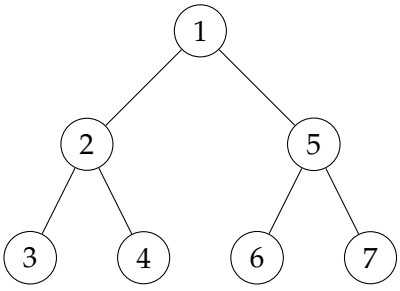


Figura 6.30: Procura por profundidade.

A procura por largura é pela ordem que está especificada na Figura 6.29, isto é, primeiro são analisados os nós do primeiro nível, e caso não se encontre o nó pretendido, então procede-se à procura do nó no nível seguinte. O mesmo já não acontece na Figura 6.30 porque, como a ordem da figura sugere, é explorado inicialmente a *root* e se

[3]Em teoria de grafos uma árvore é um grafo que não tem ciclos e oferece um caminho entre dois nós.

não for o nó desejado, então são explorados um dos seu filhos até chegar à situação de não haver filhos para analisar, retornando ao nó pai e repetir o procedimento até chegar a uma solução.

Para os modelos KAOS e de *features* o algoritmo escolhido foi a procura por largura. Existe diferenças a nível teórico entre estes dois algoritmos, como por exemplo, se o grafo tiver tamanho infinito então a procura por profundidade nunca iria encontrar uma solução caso não tivesse localizado num dos primeiros caminhos. Contudo, dado o contexto da dissertação, tais definições teóricas de cada algoritmo não influencia muito na performance do programa porque os modelos têm tamanho finito e a escolha do algoritmo de procura por largura apenas foi uma preferência no estilo de implementação para a aplicação.

No que diz respeito ao modelo conceptual, não foi usado um algoritmo específico porque, tal como o metamodelo sugere (subsecção 6.6.1), um modelo conceptual contém uma lista de classificadores (classes e associações) e, quando é realizada uma procura por uma classe, associação ou atributo basta executar uma simples iteração sobre a lista.

6.9.2 Detecção de silêncio

Como foi explicado no processo de modelação de requisitos 6.8, se um comando necessitar de usar o serviço Speech API da Google, o sistema VoiceToModel cria e grava um ficheiro de áudio com a expressão a ser traduzida para texto.

Numa primeira instância da implementação, era gravado um ficheiro de aúdio com uma duração de 4 segundos. Ao longo de várias experiências começou-se a perceber que ocorreriam situações extremistas. Caso o utilizador expressasse nomes muito extensos, os 4 segundos não chegavam, ou se o nome fosse muito pequeno, desperdiçavam-se segundos que poderiam ser usados para enviar o pedido à Google e reduzir o tempo total do pedido.

Para que duração de áudio fosse dinâmica, mas nunca superior a 10 segundos devido a limitações do serviço da Google, decidiu-se implementar um algoritmo de detecção de silêncio para quando o utilizador deixasse de pronunciar algo, a gravação do áudio terminasse através da análise de amostras de dados em **P***ulse*-**C***ode* **M***odulation*[4] (PCM). Basicamente o algoritmo analisa um intervalo do *buffer* (i.e., intensidade do som) que contém a amostra e caso seja superior a um *threshold* previamente estabelecido então existe ocorrência de voz. Se o valor for inferior ao *threshold* é detectado silêncio. Esse *threshold* depende do ambiente em que o utilizador está, pois caso esteja num ambiente sem muito ruído, naturalmente o valor do *threshold* será menor do que num ambiente com mais propenso a ruído. Este valor está sujeito a parametrização num ficheiro de configuração, para que o utilizador possa ajustar o valor consoante o ambiente que o rodeia.

[4]PCM: http://en.wikipedia.org/wiki/Pulse-code_modulation (Acesso feito em Setembro de 2014)

6.9.3 Filtragem de palavras

Apesar de actualmente os sistemas de reconhecimento de voz terem melhorado ao longo dos anos, existem algumas limitações que podem originar a traduções semelhantes ao que foi realmente dito, como foi discutido na secção 3.1 do capítulo de reconhecimento e sintetização de voz. Essa afirmação verificou-se nas experiências realizadas na aplicação VoiceToModel o que implicou na implementação de um sistema de filtragem como se pode ver na Figura 6.31.

Figura 6.31: Processo de filtragem.

Este sistema de filtragem não é gerado automaticamente, o que implica que o utilizador com a sua experiência adicione expressões manualmente na lista para que posteriormente o sistema realize o sistema de filtragem dessas palavras automaticamente.

6.9.4 Diferença entre palavras

Na subsecção anterior foi apresentado um algoritmo de filtragem com o intuito do utilizador pudesse adicionar palavras, para que no futuro o sistema realizasse a substituição das expressões para as palavras desejadas. No entanto pode acontecer a situação do utilizador dizer uma palavra que não esteja no sistema de filtragem e que seja muito semelhante ao que pretende pronunciar. Caso o comando que o utilizador pronunciou necessite de realizar uma procura pelo elemento do modelo de requisito, a palavra mal pronunciada nunca irá ser identificada pelo elemento que se está a procura no modelo.

Para tentar evitar que o utilizador necessite de repetir continuamente o comando, foi implementado o algoritmo Levenshtein Distance [Lev66] permitindo determinar a semelhança entre duas palavras. No próximo exemplo é possível verificar que entre as duas palavras, apenas existe a diferença de uma letra, ou seja, uma diferença de 20%.

$$\begin{array}{r} HELLO \\ HALLO \\ \hline 1 \end{array}$$

A percentagem de aceitação pode ser estabelecida pelo utilizador da aplicação no ficheiro de configuração. Uma percentagem com valor 0% significa que palavras semelhantes não serão aceites, logo é necessário ao utilizador expressar a palavra exacta.

6.9.5 Divisão de palavras

No que diz respeito ao sintetizador de voz, foi verificado que caso uma operação retorne um *feedback* muito extenso, o utilizador pode não assimilar a informação toda. Uma forma de resolver a situação passou por estabelecer um limite de caracteres por linha, mas caso esse limite dividisse uma palavra, então o limite é estendido até a palavra terminar. A ferramenta estabelece um intervalo de 0.5 segundos por cada linha, tempo suficiente para realizar uma transição entre as frases que seja percepível ao utilizador.

6.10 Ferramenta

As Figuras 6.32, 6.33 e 6.34 mostram os editores da aplicação VoiceToModel. No lado esquerdo são apresentados os projectos e os ficheiros dos modelos que podem ser manipulados através do controlador de projectos (ver secção 6.8). No meio surge os elementos de cada modelo de requisitos através dos comandos presentes nas secções 6.5, 6.6 e 6.7, respectivamente associados às Figuras 6.32, 6.33 e 6.34. No lado direito de cada figura são mostrados os menus de selecção que permite adicionar um elemento.

6.10.1 Configurações das tecnologias de reconhecimento de voz

Para a ferramenta saber interpretar inglês foi necessário configurar as tecnologias de reconhecimento de voz nesse idioma. Para o Sphinx-4 foi usado um modelo acústico em idioma EN-US com um dicionário com 129247 palavras. O Google Speech API também tem o idioma EN-US. Com estas configurações destas duas ferramentas foi possível derivar gramáticas para os modelos de requisitos e ter à disposição uma vasta quantidade de palavras no dicionário. A voz interpretada no sintetizador FreeTTS pertence a Kevin Lenzo com uma taxa de amostragem de 16k. Também foi definido um valor para o *pitch* para que a voz não fosse muito acelerada.

6.10.2 Limitações tecnológicas

Depois de a implementação estar concluída, averiguou-se que a API de reconhecimento de voz desenvolvida pela Google tinha limitado os pedidos para o uso do seu serviço. Foi fixado um valor de 50 pedidos por dia, o que não aconteceu durante todo o ciclo de implementação da ferramenta. Esta limitação ocorre pelo facto que nesta dissertação de mestrado apenas estar disponível ferramentas desenvolvidas e mantidas por terceiros, sujeitando-se a este tipo de cenários.

O objectivo desta dissertação de mestrado é desenvolver uma prova de conceito que permitisse demonstrar que é possível gerar modelos de requisitos através da voz numa Linguagem para Domínio Específico, tentando abstrair ao máximo das limitações que existem actualmente nas ferramentas *open source*.

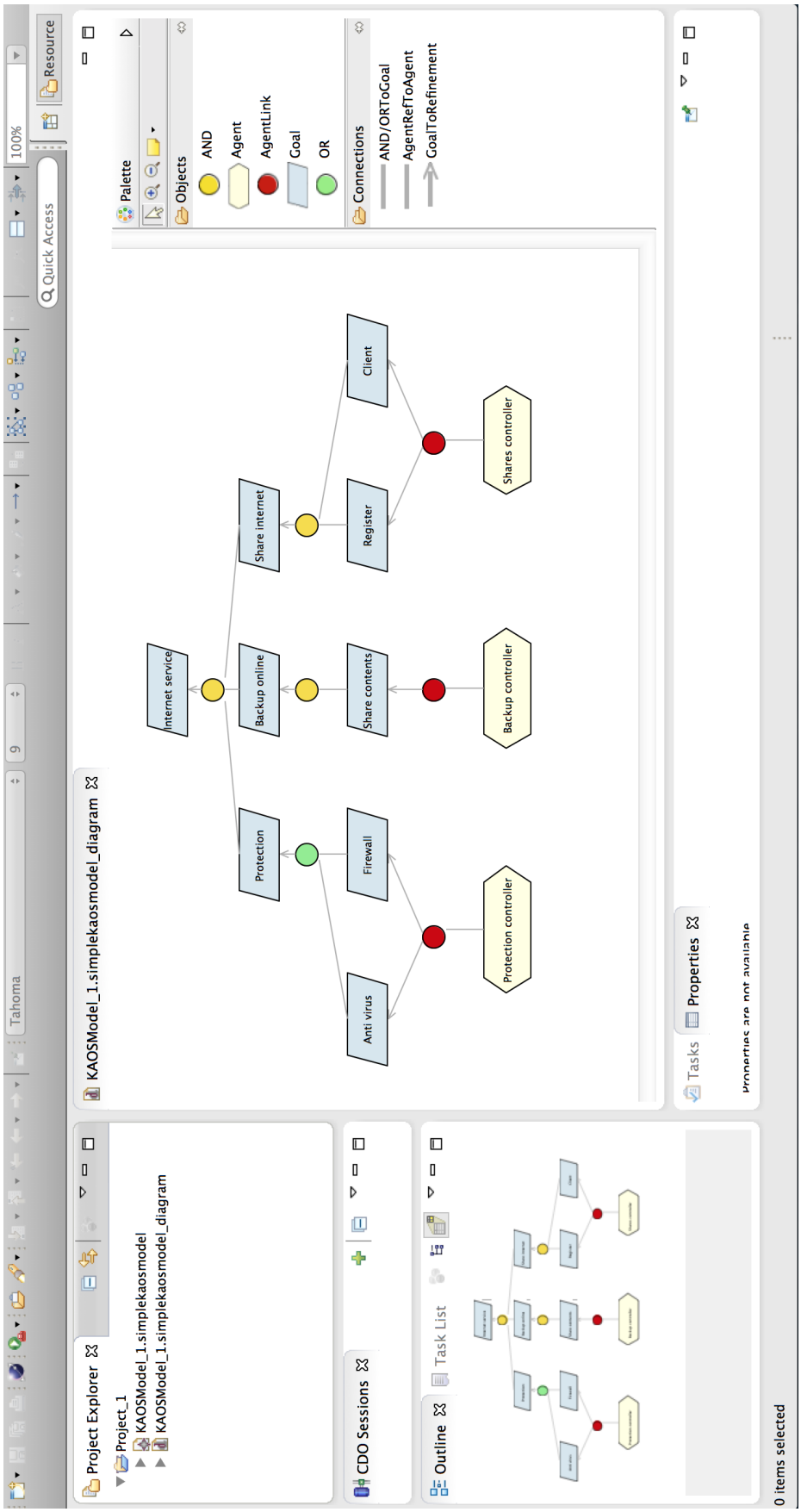


Figura 6.32: Aplicação para o Modelo SimpleKAOS.

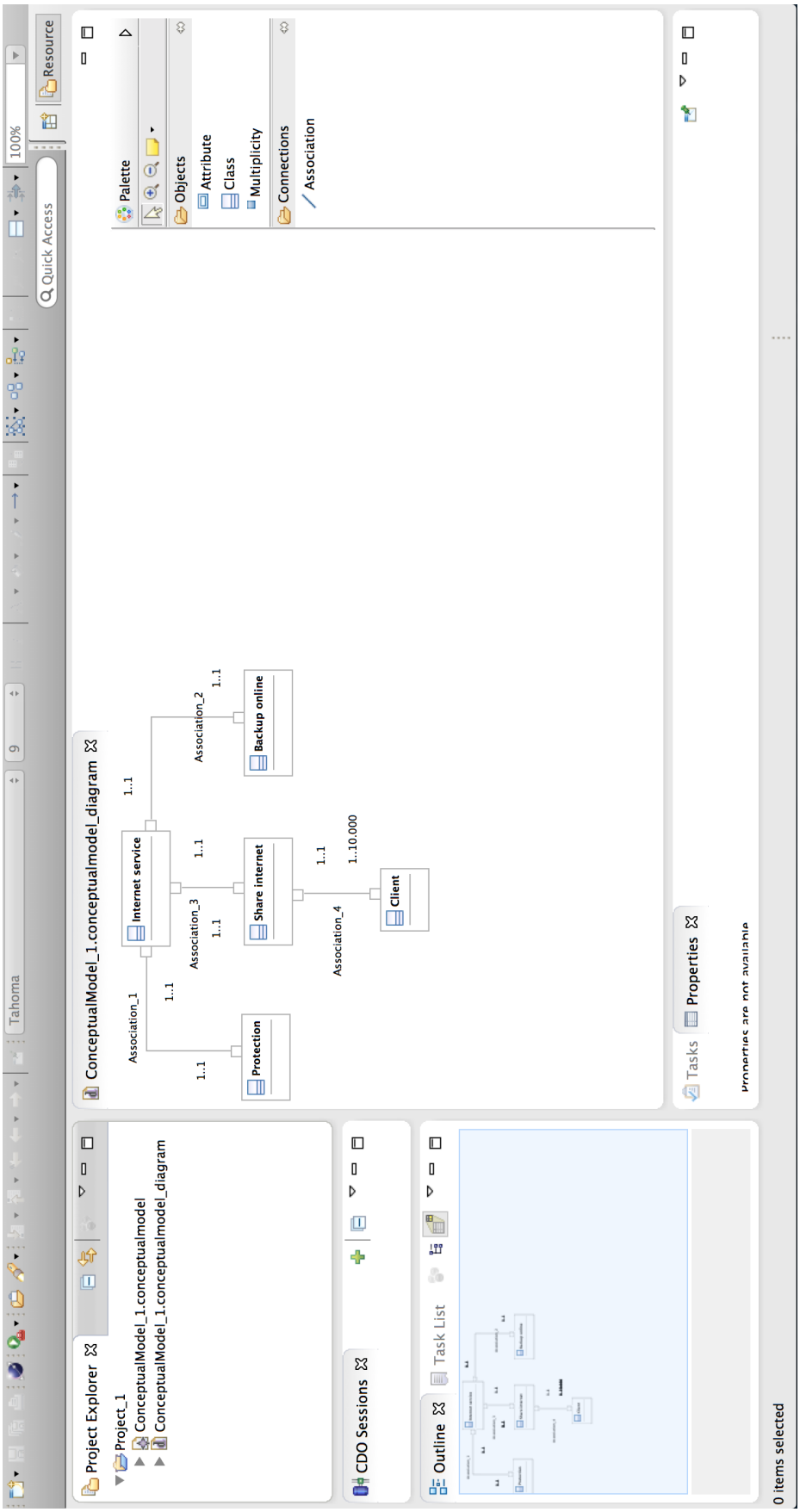


Figura 6.33: Aplicação para o Modelo Conceptual.

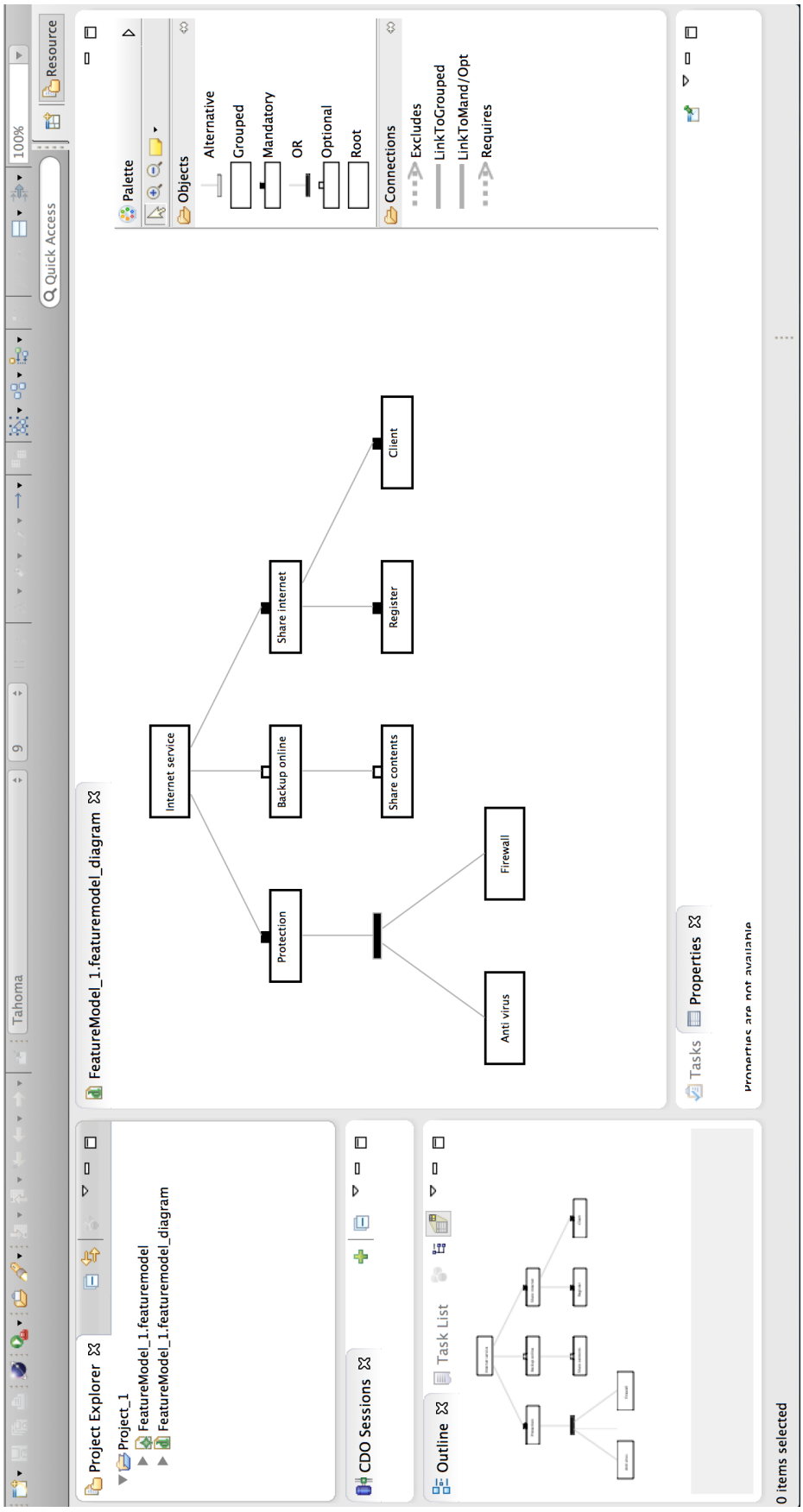


Figura 6.34: Aplicação para o Modelo de *Features*.

6.11 Sumário

Neste capítulo foi detalhado o processo de implementação da ferramenta VoiceToModel. Começou-se por escolher as tecnologias que permitiriam reconhecer voz (Sphinx-4 + Google Speech API) e permitir um *feedback* audível com base nas operações realizadas, através da ferramenta FreeTTS. Para definir as Linguagens para Domínio Específico foram usados os *plugins* EMF, Epsilon, GMF. O primeiro permitiu gerar classes Java para manipular o modelo de requisito, o segundo especifica as restrições em EVL e o GMF disponibiliza uma vista diagramática do respectivo modelo.

Depois foi apresentado o modelo de componentes da aplicação, mostrando de que forma é que as classes se interagem entre elas. Foi escolhida a linguagem por comandos como o modelo de interacção usado para servir de ponte entre o utilizador e a ferramenta. Nas subsecções seguintes foram explicados como foi feita a derivação de comandos a cada modelo de requisitos. Por fim foram apresentadas algumas técnicas que foram usadas durante o ciclo de desenvolvimento da ferramenta VoiceToModel. O próximo capítulo é descrito a avaliação e discussão de resultados da ferramenta VoiceToModel.

7

Avaliação

7.1 Introdução

Segundo Travassos et al. [TGA02], a experimentação é o centro do processo científico. De acordo com os autores, os experimentos têm a capacidade de verificar teorias e podem explorar factores críticos e originar novos fenómenos para que as teorias possam ser formuladas e rectificadas. É por meio dos experimentos que é possível obter um método sistemático, disciplinado, computável, e controlado para avaliação da actividade humana. O experimento geralmente é realizado em laboratórios e oferece, portanto, um nível de controlo maior. O objectivo do experimento é manipular uma ou algumas variáveis e manter outras fixas medindo o efeito do resultado, para que se possam extrair algumas evidências.

Para a Engenharia de *Software*, a existência de metodologias específicas é cada vez mais necessária para ajudar a estabelecer uma base de engenharia e ciência propriamente dita. Para isto, existem quatro métodos relevantes para a condução de experimentos na área de Engenharia de *Software*: científico, de engenharia, experimental e analítico [Woh+00].

Nesta dissertação foi usado o método experimental, aplicando um experimento na ferramenta VoiceToModel para um conjunto de utilizadores. Os utilizadores foram convidados a realizar tarefas usando a aplicação e, posteriormente, a realização do questionário, que pode ser consultado no Anexo C. Também foram aplicadas métricas para determinar a duração total da execução das tarefas, número total de operações, entre outros. A partir dos resultados obtidos será possível concluir se o utilizador consegue modelar os requisitos de acordo com as suas expectativas usando a aplicação VoiceToModel.

De modo a realizar a avaliação experimental, foi reunido um conjunto de 14 utilizadores para que pudessem usar a ferramenta. O grupo de participantes varia de vários estabelecimentos quanto à sua formação. É constituído por um grupo de 14 indivíduos, 12 pertencem à Faculdade de Ciências e Tecnologias da Universidade Nova de Lisboa e Instituto Politécnico de Setúbal, frequentando a Licenciatura e Mestrado de Engenharia Informática. Os restantes 2 indivíduos são pessoas cegas, um com conhecimentos técnicos em Informática e outro com Mestrado em Engenharia Informática proveniente da Faculdade de Ciências da Universidade de Lisboa. Entre os participantes, 12 tem contato visual com a ferramenta, i.e., não usam venda. As próximas secções vão detalhar o experimento, como é que os dados vão ser analisados e a discussão dos resultados.

7.2 Planeamento para a avaliação experimental

O processo de planeamento consistiu em primeiro lugar estabelecer critérios para eleger o grupo de utilizadores, nomeadamente pessoas que tenham conhecimentos a nível superior em informática e participantes cegos - motivação da dissertação. Também foi feita uma análise de quais as ferramentas que os utilizadores poderiam utilizar para estabelecer um termo de comparação em relação à ferramenta VoiceToModel.

Foram definidas as tarefas que os participantes iriam executar, aplicando um caso de estudo para os modelos KAOS, conceptual e de *features* usando a ferramenta VoiceToModel. Por fim, foi detectada uma possível limitação que poderia prejudicar a avaliação. As próximas subsecções abordam estes aspectos de forma mais aprofundada.

7.2.1 Processo de eleição dos utilizadores

De forma a eleger o grupo de utilizadores para experimentar a ferramenta foram estabelecidos os seguintes pré-requisitos: não tivessem conhecimento de como funciona a aplicação ou nunca o tivessem experimentado; conhecimentos no mínimo técnicos a nível informático e falar Inglês de forma adequada. A razão pela qual solicitava-se no mínimo conhecimentos técnicos em Informática e não de Licenciatura/Mestrado em Engenharia Informática deve-se ao facto do desejo de incluir no experimento pessoas cegas e como foi bastante difícil encontrar pessoas com essas características e com disponibilidade, os requisitos mínimos fixaram-se ao nível técnico visto que o *framework* se destina ao público de desenvolvimento de *software*. A inclusão de utilizadores cegos enquadra-se na motivação da dissertação e oferece a possibilidade de analisar o seu comportamento perante a aplicação nas mais variadas situações.

A inclusão dos utilizadores cegos deveu-se graças à colaboração com a **A**ssociação dos **C**egos e **A**mblíopes de **P**ortugal (ACAPO) que, com base nos requisitos mínimos dos conhecimentos que teriam de ter para usar a aplicação, foram fornecidos os contactos necessários para comunicar com essas pessoas para saber se estariam interessadas em participar no processo de avaliação.

7.2.2 Relação com outras ferramentas

Durante o processo de avaliação seria interessante incluir ferramentas que usassem uma abordagem semelhante à aplicação VoiceToModel para os utilizadores experimentarem e estabelecer termos de comparação a nível de usabilidade e modelação dos requisitos entre as ferramentas. A ferramenta VoCoTo [LP03], discutida na subsecção 5.3.3 do capítulo de Trabalhos Relacionados, é a mais próxima no contexto da dissertação porque possibilita a modelação em UML usando sistemas de reconhecimento de voz.

A aplicação VoCoTo não se encontra disponível *online* para fazer o *download* do programa e a alternativa passou por contactar os autores. Após o contacto, foi dito que a aplicação está obsoleta, isto é, seria necessário como requisitos mínimos o Windows XP, a API **M***icrosoft* **S***peech* (SAPI) 5.1, **R***ational* **R***ose* **E***nterprise*, algumas ferramentas adicionais para o Rose e uma versão antiga do **V***isual* **S***tudio*. Perante estas adversidades, e para além do facto do preço de uma licença do Rational Rose de 2004 pela IBM ser muito alto, a decisão recaiu em não incluir a aplicação VoCoTo no experimento.

As ferramentas populares de modelação Objectiver [IT14], MagicDraw [Mag14] e FeatureIDE [Kas+05] que permitem modelar modelos KAOS, modelos conceptuais e modelos de *features*, não foram incluídas no processo de avaliação porque não oferecerem nenhuma estratégia de acessibilidade, acabando por não se incluir ferramentas adicionais no experimento.

7.2.3 Especificação das tarefas

Depois de definido o grupo de utilizadores e a análise das ferramentas disponíveis para comparação, o próximo passo foi delinear as tarefas para a avaliação. Foram criadas 6 tarefas para cada utilizador executar no experimento. Em todas as tarefas o ponto comum é sobre o caso de estudo: considera-se um sistema de *Email* que permite enviar, receber e notificar sobre a recepção de um *email*. O envio do *email* só pode ser feito com o uso de texto ou imagem e a recepção só pode ser feita em casa ou no trabalho. O objectivo da avaliação não é a apresentação de um caso de estudo complicado para modelar usando a aplicação VoiceToModel, mas sim apresentar um sistema bastante comum para todos os utilizadores e realizar a sua modelação, já que o objectivo é analisar como o utilizador se comporta ao usar a ferramenta.

Na primeira tarefa pede-se a modelação do sistema do *Email* através do modelo KAOS, na segunda tarefa usando o modelo conceptual e na terceira tarefa através do modelo de *features*. Estas são as tarefas principais da avaliação porque permitem aos utilizadores terem um primeiro contacto com a ferramenta e usarem todos os comandos que o sistema oferece. As tarefas 4, 5 e 6 solicitam ligeiras alterações nos modelos criados nas tarefas anteriores e a obtenção de informação sobre o modelo, tais como perguntar ao modelo quantos *goals* e agentes há no modelo e quais os nomes dos *sub goals* do *goal Email* no modelo KAOS, o número total e o nome das classes que havia no modelo no que diz respeito ao modelo conceptual e obter informação sobre as *features* no modelo de *features*.

Antes de apresentar as tarefas aos participantes, foi realizada uma formação a cada utilizador, de modo a contextualizar onde é que esta ferramenta poderia ser utilizada. Depois foram apresentados os conceitos do modelo KAOS usados na ferramenta, um exemplo de como poderia ser usado o modelo usando o exemplo do Ginásio apresentado na subsecção 2.3.1 do KAOS e, de seguida, um vídeo com esse mesmo exemplo usando a ferramenta. Depois de o vídeo terminar foram apresentados aos utilizadores videntes[1] os comandos em forma de uma tabela, com os comandos, descrição, nota e uma imagem de como é o resultado após a operação (semelhante ao que está no Anexo A) em folhas A4; para os utilizadores cegos foram apresentadas as tarefas e os comandos em formato braile, disponível no Anexo D. A mesma sucessão de passos repetiu-se nas tarefas 2 e 3, respectivamente o modelo conceptual e o modelo de *features*, sendo que nas tarefas 4, 5 e 6 os utilizadores já tinham conhecimentos dos comandos e de como funcionava a ferramenta e, por isso, não foi necessária uma formação adicional.

No decorrer de cada tarefa, o sistema guarda automaticamente num ficheiro *log* todos os comandos que foram comunicados e executados na aplicação de forma automática, com o fim de aplicar métricas nos comandos que foram implementados. No fim de todas as tarefas, foi solicitado a cada utilizador que preenchesse um questionário de modo a fornecer *feedback* sobre a ferramenta.

7.2.4 Capacidade da ferramenta

Na subsecção 6.10.2 foi mencionado o facto da API da Google apenas disponibilizar 50 pedidos diários. Dado o volume das tarefas, os 50 pedidos poderiam não chegar para o utilizador completar todas as tarefas porque depende de como se adapta ao sistema. De forma a contornar tal obstáculo, foram feitos contactos com os responsáveis do departamento de *Speech Recognition* da Google para que pudessem aumentar o número de pedidos para realizar a avaliação. Depois de explicado os motivos, um dos responsáveis aumentou o número de pedidos diários para 300, de modo a realizar a avaliação com todos os utilizadores sem problemas.

As próximas secções vão fornecer os resultados de cada experimento, mais especificamente, na secção 7.3 vão ser expostas as métricas que foram estabelecidas e os resultados de cada tarefa. Na secção 7.4 vai ser feita uma análise do *feedback* dado por cada utilizador com base no questionário.

7.3 Métricas

Para analisar dados provenientes do utilizador, foram estabelecidas 6 métricas, que permitem extrair dados para a discussão de resultados. As métricas são baseadas na abordagem **G***oal*-**Q***uestion*-**M***etric* (GQM) [Bas92]. A metodologia GQM, desenvolvida

[1] Vidente: Que ou pessoa que vê, em contraposição a pessoa cega. - http://www.priberam.pt/DLPO/vidente (Acesso feito em Setembro de 2014)

por Basili, permite a medição de actividades num sistema de *software* a partir de perguntas e objectivos. A metodologia tem três níveis [CR94]:

- **Nível Conceptual - Objectivo**: identificação de objectivos, a partir de vários pontos de vista, sobre um determinado ambiente;
- **Nível Operacional - Questão**: elaboração de questões para determinar a forma como a avaliação de um objectivo específico vai ser realizado;
- **Nível Quantitativo - Métrica** : definição de métricas, associada a cada questão.

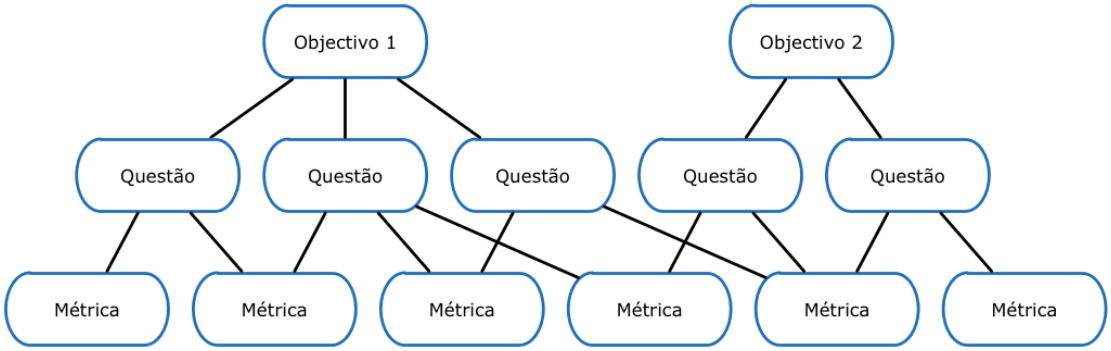


Figura 7.1: Estrutura da abordagem GQM [CR94].

A Figura 7.1 ilustra, de forma hierárquica, a estrutura do nível conceptual, operacional e quantitativo. Primeiro são definidos objectivos no nível conceptual e, posteriormente, refinados em questões para determinar como esses objectivos poderão ser obtidos (nível operacional). Por fim, cada questão é refinada em métricas, podendo ser objectivas ou subjectivas (nível quantitativo).

Para este trabalho, foi estabelecido o seguinte objectivo: **avaliar a performance da ferramenta VoiceToModel**. Foram formuladas duas perguntas: (P1) Qual o tempo total em cada tarefa?; e (P2) Qual o esforço que foi necessário para executar cada tarefa? A Tabela 7.1 apresenta as métricas que respondem às perguntas P1 e P2, sendo que a P1 está associada à métrica 1, e a P2 associada às restantes métricas. Cada métrica é composta pelo seu identificador, descrição e a unidade de medida.

Tabela 7.1: Métricas para a análise de dados.

Métrica 1	
M1	Duração total da realização de uma tarefa.
Unidade	Minutos e segundos com o formato mm:ss.
Métrica 2	
M2	Número total dos comandos executados durante uma tarefa.
Unidade	Operação.

Métrica 3	
M3	Número total dos comandos que tenham apenas um parâmetro (que tenham a flexibilidade dos parâmetros).
Unidade	Operação.
Métrica 4	
M4	Número total dos comandos que tenham sido usados todos os parâmetros (que tenham a flexibilidade dos parâmetros).
Unidade	Operação.
Métrica 5	
M5	Número total de todos os comandos (delete, change, undo command) procedentes ao comando mal executado.
Unidade	Operação.
Métrica 6	
M6	Número total dos comandos que tenham sido mal interpretados e/ou diferentes que o utilizador expressou.
Unidade	Operação.

A métrica 1 permite obter a duração de cada tarefa, não sendo um dado crucial, porque o tempo pode ser afectado pela *accuracy* do sistema de reconhecimento de voz. A métrica 6 apresenta a quantidade de operações que foram mal expressadas pelo utilizador, podendo justificar o elevado/baixo número total de operações na métrica 2. As métricas 3, 4 e 5 não incluem comandos que não tenham a flexibilidade de parâmetros tais como "say", "sleep mode" e "help mode", logo é possível expressar a inequação (7.1), definindo que a métrica 2 pode não ser igual ao somatório das métricas 3, 4 e 5:

$$M_2 \leq M_3 + M_4 + M_5 \tag{7.1}$$

As próximas subsecções vão apresentar e discutir os resultados para cada métrica. Os dados vão permitir analisar o volume de tempo, os passos que foram necessários efectuar para finalizar o que cada tarefa pedia, se o utilizadores aderiram à flexibilidade dos parâmetros e o número total de operações que não foram reconhecidos.

7.3.1 Resultados das métricas

As Tabelas 7.2, 7.3, 7.4, 7.5, 7.6 e 7.7 disponibilizam os resultados das métricas estabelecidas na Tabela 7.1. Na primeira coluna de cada tabela está presente a quantidade de utilizadores com o formato UX e UCX, sendo U os utilizadores videntes, UC os utilizadores cegos e X o identificador de cada utilizador. Nas restantes colunas são apresentadas a duração de cada tarefa no formato mm:ss, no caso da Tabela 7.2, ou o número total de operações em cada tarefa, nas restantes tabelas. Na última linha da tabela é apresentada a média com os valores especificados na respectiva coluna.

Resultados da duração total de uma tarefa

Tabela 7.2: Resultados da duração total de uma tarefa (M1).

Utilizador	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
U1	7:08	7:55	8:28	1:30	2:54	2:00
U2	5:29	6:02	8:20	1:15	3:11	1:52
U3	6:10	6:30	4:19	1:58	2:42	1:45
U4	12:13	13:49	3:16	2:41	00:00	00:00
U5	8:10	10:45	5:13	2:19	2:38	2:04
U6	11:51	14:00	5:10	1:41	1:29	2:15
U7	6:42	7:37	5:22	2:09	3:31	2:25
U8	7:11	6:43	6:56	1:57	2:25	1:39
U9	7:28	6:59	5:44	3:05	3:39	2:12
U10	8:32	8:03	4:01	2:23	2:05	1:23
U11	8:12	9:28	5:15	2:39	2:30	2:48
U12	12:00	11:00	9:12	3:06	2:45	2:21
UC13	6:42	13:28	4:18	1:19	1:33	1:42
UC14	6:12	5:54	4:31	1:57	2:06	1:48
Média	8:29	9:09	5:43	2:08	2:23	1:52

Com base nos valores apresentados na Tabela 7.2 é possível verificar que nas primeiras três tarefas, a modelação usando um paradigma orientado a objectos na tarefa 2 teve uma média mais elevada comparativamente às tarefas 1 e 3. O tempo demorado na primeira tarefa deveu-se ao facto dos utilizadores nunca terem utilizado a ferramenta e os respectivos comandos, demorando algum tempo até se adaptarem. Nas restantes tarefas o tempo de duração foi muito próximo entre os três.

Os utilizadores cegos UC13 e UC14 tiveram um bom desempenho no geral. Apenas o utilizador UC13 na tarefa 2 demorou mais tempo em relação à média dos restantes utilizadores. O participante UC14 foi o mais consistente, em todos os utilizadores, na duração mais curta em todas as tarefas.

A Figura 7.2 apresenta um diagrama *boxplot*[2] com os valores da Tabela 7.2. Segundo a figura, é possível confirmar o que foi concluído na discussão dos resultados da Tabela 7.2, isto é, que a tarefa 2 tem uma duração superior em comparação às restantes tarefas. Também é possível verificar que as três primeiras tarefas têm uma duração semelhante entre elas e as três últimas igualmente. Esta conclusão vem confirmar o que foi referido na subsecção 7.2.3, onde é afirmado que as primeiras três tarefas são as mais complexas comparativamente às últimas três.

Além disso, a figura evidencia cinco *outliers*. Os primeiros três *outliers*, associados aos

[2] *Boxplot* gerado na linguagem R, que se foca em cálculo estatístico. - http://www.r-project.org (Acesso feito em Setembro de 2014)

utilizadores 4, 6 e 12, situam-se na primeira tarefa devido ao excessivo tempo que tiveram em relação aos restantes utilizadores e os dois últimos *outliers* da tarefa 5 e 6 são associados ao utilizador 4 (prestação nula). Como foi especificado na secção 6.4.1, que apresenta uma visão geral dos comandos, o comando "undo command" não está presente no controlador de projectos, implicando a não reversibilidade das operações. Devido a falhas do sistema de reconhecimento de voz, o sistema assumiu o comando "delete project" e apagou o projecto com os ficheiros dos modelos que o utilizador tinha criado nas tarefas 1, 2 e 3. Essa situação implicou a não realização das últimas duas tarefas, sendo que as mais importantes já tinham sido realizadas, e consequentemente, as métricas 1, 2, 3, 4, 5 e 6 apresentam resultados nulos nas tarefas 5 e 6 para o utilizador 4.

Este acontecimento levou à alteração da forma como são abordados os comandos "delete" no controlador de projectos e "undo command" nos modelos, isto é, para que essa operação seja executada o utilizador terá que confirmar a operação com a resposta *yes* ou *no*.

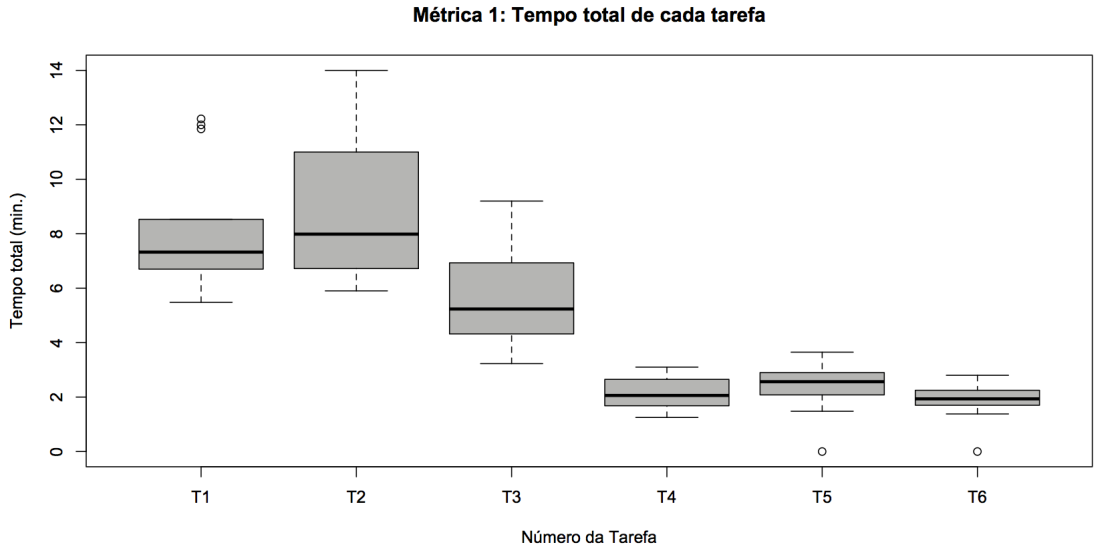


Figura 7.2: *Boxplot* da duração total de uma tarefa.

Resultados do número total de operações realizadas

A Tabela 7.3 mostra os resultados do número total de comandos realizado pelos participantes no experimento. Esta métrica poderá ter uma correlação com a métrica número um. No entanto, isto não significa que o número total de operações tenha que ser semelhante a todos os utilizadores, caso tenham tempos semelhantes. Cada utilizador teve o seu tempo de reflexão e análise do problema e dos comandos quando o sistema estava activo, não contabilizado no número total de operações.

Tabela 7.3: Resultados do número total de operações realizadas (M2).

Utilizador	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
U1	35	41	46	11	14	14
U2	32	35	42	8	17	10
U3	34	37	28	10	11	10
U4	68	79	23	16	0	0
U5	44	54	33	13	16	16
U6	62	84	30	11	9	11
U7	34	39	37	12	16	16
U8	38	33	39	11	12	12
U9	32	35	34	18	21	14
U10	41	46	25	14	14	10
U11	40	52	35	13	15	17
U12	65	61	54	16	18	12
UC13	58	68	25	10	12	11
UC14	30	30	28	9	10	9
Média	43.79	49.57	34.21	12.29	13.21	11.57

Como se pode verificar na Tabela 7.3, o número das operações está directamente relacionado com a duração de cada tarefa, especificado na Tabela 7.2. A média do número de operações da tarefa 2, responsável pela modelação do caso de estudo através do modelo conceptual, foi superior em relação às restantes. Foi possível analisar durante o experimento que os utilizadores apresentaram mais dificuldades neste modelo. A tarefa 5 também teve um valor superior em comparação às tarefas 4 e 6.

A prestação do utilizador cego UC13 no número total de operações foi superior em relação à média nas tarefas 1 e 2, e inferior à média nas tarefas 3, 4, 5 e 6. O utilizador não vidente UC14 teve um bom desempenho, sendo que o número total de operações foi inferior em todas as tarefas comparativamente à média dos restantes utilizadores.

A Figura 7.3 apresenta os resultados da Tabela 7.3 através do diagrama *boxplot*. Com base na figura é confirmado o facto das primeiras três tarefas apresentarem um maior fluxo de operações executadas, em comparação às tarefas 4, 5 e 6. Também se confirma o facto da tarefa que envolve o modelo conceptual apresentar um maior número de operações em relação ao SimpleKAOS da tarefa 1 e modelo de *features* da tarefa 3.

Existem diferenças dos *outliers* da Figura 7.2 com os *outliers* da Figura 7.3. Não existem excepções na primeira tarefa, concluindo que não há casos extremos no número total de operações. Os *outliers* nas tarefas 5 e 6, atribuídos ao utilizador 4, mantém-se devido ao facto do participante não ter realizado as respectivas tarefas.

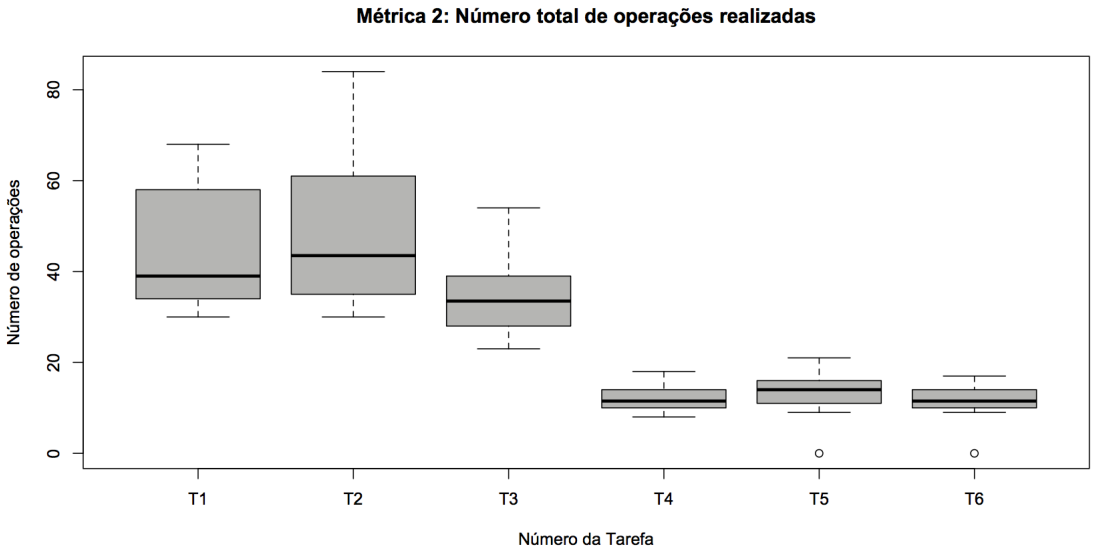


Figura 7.3: *Boxplot* do número total de operações realizadas.

Resultados do número total de comandos com um/dois parâmetros

As Tabelas 7.4 e 7.5 apresentam os resultados dos somatórios dos comandos que tenham sido utilizados com apenas um parâmetro ou todos os parâmetros, baseado na flexibilidade de utilização dos parâmetros implementados no projecto.

Tabela 7.4: Resultados do número total de comandos com apenas um parâmetro (M3).

Utilizador	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
U1	5	17	31	6	7	8
U2	12	16	16	7	13	8
U3	12	12	14	6	9	7
U4	19	22	12	8	0	0
U5	20	19	14	7	12	10
U6	21	30	14	9	6	6
U7	11	21	16	10	12	12
U8	12	19	18	6	10	8
U9	11	17	13	14	14	9
U10	14	23	12	11	12	8
U11	12	28	16	12	12	11
U12	19	20	28	10	13	7
UC13	24	24	14	8	10	6
UC14	13	16	16	6	7	7
Média	14.64	20.29	16.71	8.57	9.79	7.64

Tabela 7.5: Resultados do número total de comandos usando todos os parâmetros (M4).

Utilizador	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
U1	11	5	2	1	2	1
U2	2	1	0	1	0	1
U3	5	4	0	0	0	0
U4	5	3	0	0	0	0
U5	0	6	0	1	0	0
U6	1	0	0	0	1	0
U7	0	0	2	1	2	1
U8	0	0	0	2	0	1
U9	2	0	1	0	0	0
U10	0	0	0	0	0	0
U11	4	0	0	0	0	0
U12	0	0	0	0	0	0
UC13	0	0	0	0	0	0
UC14	0	0	0	0	0	0
Média	2.14	1.36	0.36	0.42	0.36	0.29

A partir das Tabelas 7.4 e 7.5 é possível concluir que, com base na média calculada em cada tarefa, os utilizadores aderiram à introdução de flexibilidade dos parâmetros que a maior dos comandos permitem.

Analisando a Tabela 7.4 é possível verificar que quando os utilizadores usaram comandos que eram versáteis nos seus parâmetros (e.g., "add mandatory", "add goal refinement OR/AND", entre outros), conclui-se que essa versatibilidade teve muita adesão registada por todos os participantes. A excepção é o utilizador 1, que teve uma menor média comparativamente aos restantes. Os utilizadores cegos UC13 e UC14 também contribuíram para a afluência do elevado uso da flexibilidade dos parâmetros que muitos comandos possuem.

Em contraste dos dados da Tabela 7.4 são os resultados da Tabela 7.5, comprovando que grande parte dos utilizadores não usou todos os parâmetros que cada comando possui, sendo que o utilizador 1 é a excepção mais acentuada. Não foram registadas ocorrências de operações usando todos os parâmetros da parte dos utilizadores UC13 e UC14.

Resultados do número total vezes que um comando foi corrigido e não reconhecidos

As Tabelas 7.6 e 7.7 mostram os resultados do número total de vezes numa tarefa que um comando foi corrigido e que o sistema não reconheceu devido à capacidade do sistema de reconhecimento de voz ou do utilizador não expressar correctamente o comando.

Tabela 7.6: Resultados do número total de vezes que um comando foi corrigido (M5).

Utilizador	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
U1	4	8	16	1	1	0
U2	6	4	11	0	1	1
U3	8	7	2	0	1	0
U4	14	14	0	0	0	0
U5	13	10	4	1	1	2
U6	12	14	3	0	1	0
U7	7	9	6	0	0	1
U8	9	4	6	1	0	1
U9	8	5	7	3	5	2
U10	7	10	2	1	1	0
U11	4	11	5	1	0	3
U12	15	15	10	2	2	2
UC13	10	16	4	0	1	1
UC14	4	2	8	0	0	0
Média	8.64	9.21	6	0.71	1	1.07

Tabela 7.7: Resultados da métrica do número total de comandos não reconhecidos (M6).

Utilizador	Tarefa 1	Tarefa 2	Tarefa 3	Tarefa 4	Tarefa 5	Tarefa 6
U1	11	8	4	2	0	0
U2	8	5	14	0	1	0
U3	7	7	8	3	0	2
U4	21	21	4	4	0	0
U5	15	15	6	2	2	3
U6	25	34	12	1	0	2
U7	9	7	12	0	1	1
U8	11	4	7	2	1	1
U9	9	11	8	0	2	0
U10	15	10	6	0	0	0
U11	9	12	8	0	3	2
U12	22	24	13	2	2	2
UC13	21	22	6	1	1	2
UC14	8	9	2	2	2	0
Média	13.64	13.5	7.86	1.36	1.07	1.07

Como se pode analisar nas duas tabelas, o número total de operações proveniente de correcções e não reconhecidos ainda são consideravelmente grandes. Estes números têm como base analisar a taxa de *accuracy*, o que pode conduzir ao utilizador o uso dos comandos como "delete", "change" ou "undo command" para reverter uma situação que não estava planeada. Contudo a correcção das operações nem sempre esteve relacionada com o mau reconhecimento dos comandos via voz. Alguns dos utilizadores usaram comandos como "change type of feature", "change refinement or/and to or/and" ou até mesmo "change name" porque aperceberam-se que especificaram mal o modelo. O uso de correcções do modelo devido a especificações incorrectas por parte dos participantes também levou ao aumento da duração da tarefa e do número total de operações.

O comportamento do participante não vidente UC13 na Tabela 7.6 foi superior na média das primeiras duas tarefas, enfrentando algumas dificuldades na execução correcta dos comandos. Por outro lado o utilizador UC14 teve uma prestação mais favorável, com excepção na estruturação do caso de estudo com o modelo de *features*, apresentado na tarefa 3. Na Tabela 7.7 o utilizador UC13 mostrou novamente dificuldades no reconhecimentos dos comandos que expressou, mais acentuado nas primeiras duas tarefas e o utilizador UC14 adaptou-se bem à *framework*, apresentando valores abaixo da média nas primeiras três tarefas.

Os resultados das Tabelas 7.6 e 7.7 na prática não são relacionáveis, dado que nem sempre um comando mal reconhecido implicava a sua correcção porque podia não alterar o estado do modelo, como por exemplo os comandos "say", "help mode" e "sleep mode". A próxima secção vai apresentar uma análise estatística em relação às questões impostas no questionário.

7.4 Análise do questionário

O questionário que o utilizador preencheu está dividido em quatros secções: dados pessoais do utilizador, classificação de aspectos de acessibilidade para gerar modelos de requisitos, opinar sobre o modelo de interacção implementado e obter *feedback* sobre os aspectos gerais da ferramenta.

7.4.1 Dados pessoais

Esta subsecção tem como objectivo analisar qual o *background* de cada utilizador e o nível do seu Inglês. O enquadramento serviu para analisar se o utilizador tem conhecimentos suficientes no contexto do desenvolvimento do *software* e o nível de inglês permite averiguar se o participante é capaz de usar e entender a aplicação VoiceToModel.

A Figura 7.4 mostra que 50% dos utilizadores estão enquadrados na categoria de mestrado em Engenharia Informática, 14% dos utilizadores estão a realizar o doutoramento no contexto da Engenharia de *Software*, 29% na licenciatura em Engenharia Informática e, por fim, 7% tem conhecimentos técnicos em Informática.

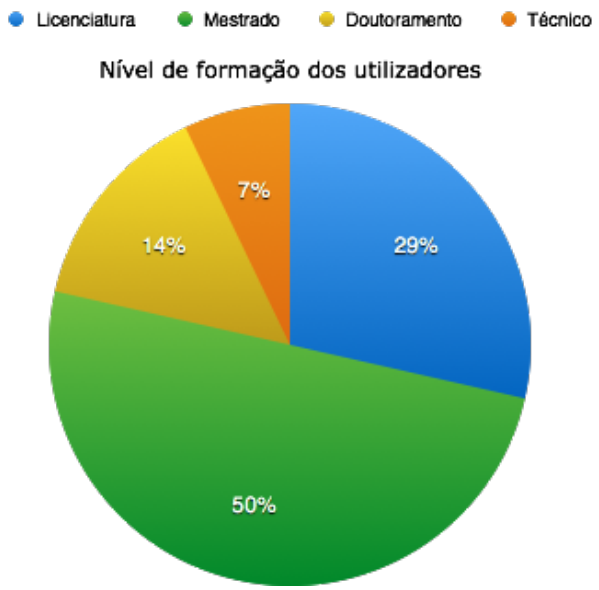


Figura 7.4: Resultados do nível de formação dos utilizadores.

A Figura 7.5 demonstra que 14% dos utilizadores têm um nível básico do Inglês, 57% dos utilizadores têm um nível intermediário no Inglês, e 29% apresentam um nível avançado. A questão sobre o nível do Inglês é importante no contexto da dissertação porque tanto o reconhecimento de voz como o sintetizador estão em Inglês e é fundamental para que o utilizador conseguir estabelecer uma boa comunicação com a ferramenta e diminuir a taxa de erros no reconhecimento de voz.

Figura 7.5: Resutados do nível do Inglês.

7.4.2 Acessibilidade

Na subsecção de acessibilidade foi pedido aos utilizadores para fornecerem *feedback* sobre aspectos de acessibilidade que permitem gerar modelos de requisitos com base na ferramenta VoiceToModel. Nas primeiras 6 questões, referentes à duração de cada tarefa, os resultados foram apresentados na Tabela 7.2 e posteriormente discutidos. No que diz respeito à pergunta "Qual a sua impressão ao usar a ferramenta", a Figura 7.6 mostra os resultados com base no que os utilizadores seleccionaram.

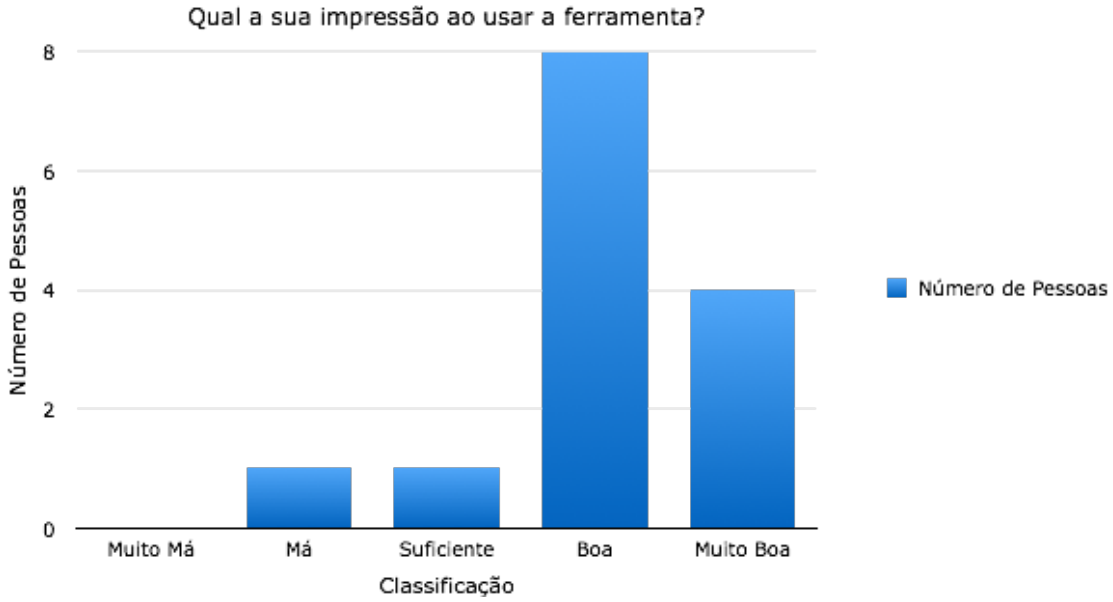


Figura 7.6: Resultados da impressão ao usar a ferramenta.

Com base nestes resultados, 7% achou má como a ferramenta funciona, 7% consideraram suficiente a experiência, 57% achou que a experiência foi boa e 29% seleccionou a classificação de muito boa. Os 7% que achou má a ferramenta é proveninente de um utilizador que criou expectativas de um produto final e não de um protótipo, dado as falhas que ocorrem dos mecanismos de reconhecimento de voz. Os utilizadores cegos ficaram satisfeitos com a acessibilidade da ferramenta, sendo que atribuíram a experiência como boa e muito boa.

A Figura 7.7 mostra os resultados da pergunta "Foi difícil usar o VoiceToModel para o exemplo apresentado na avaliação?", onde 21% sentiu dificuldades e 79% dos utilizadores sentiram-se capazes de ultrapassar as adversidades. A cotação de 21% pertence a 3 utilizadores e os restantes 11 utilizadores pertencem ao grupo dos 79% que conseguiram superar as dificuldades. O *feedback* dos três utilizadores que sentiram que foi muito difícil o uso da ferramenta foi devido ao facto da *accuracy* dos sistemas de reconhecimento de voz ter obrigado a repetir mais que uma vez o comando que pretendiam executar. Os utilizadores cegos enquadram-se no grupo de 11 utilizadores que consideraram que a ferramenta não foi difícil de ser manipulada.

Figura 7.7: Resultados das dificuldade ao usar a ferramenta.

Os resultados da pergunta "Os modelos gerados foram os esperados?" indicam que 14% seleccionaram a resposta "Não", atribuídos a 2 utilizadores, 86% seleccionaram a resposta "Sim", que traduz nos restantes 12 utilizadores. Os dois utilizadores cegos enquadram nos 12 participantes que conseguiram gerar os modelos de acordo com as suas expectativas.

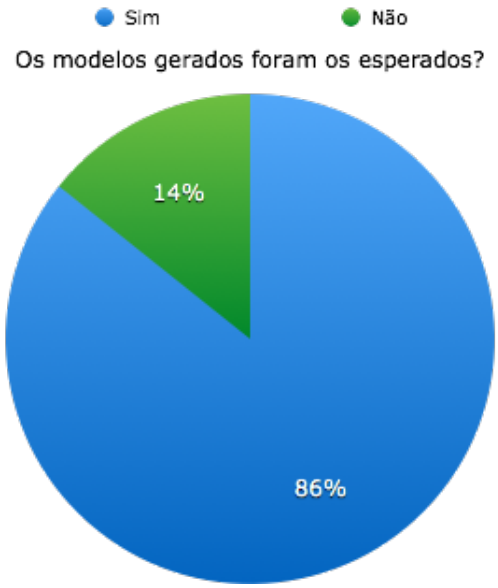


Figura 7.8: Resultados das expectativas dos modelos criados.

De modo a obter *feeback* das respostas negativas da pergunta anterior, foi submetida a seguinte pergunta "Se na pergunta anterior a resposta seleccionada foi 'Não', indique qual o modelo em causa e a(s) dificuldade(s) enfrentadas.".

O primeiro utilizador que seleccionou a resposta "Não" justificou a resposta por não ter conseguido completar as últimas duas tarefas, tal como foram explicadas as razões na subsecção 7.3.1. Apesar desse fundamento, as três últimas tarefas eram as menos relevantes (completou uma dessas três), finalizando com sucesso as primeiras três tarefas consideradas as mais fundamentais. O segundo utilizador justificou as falhas que teve em alguns modelos devido ao sistema de reconhecimento de voz, API essa não desenvolvida na dissertação.

7.4.3 Modelo de interacção

Esta subsecção fornece as respostas apresentadas pelos utilizadores sobre o modelo de interacção implementado na dissertação. A Figura 7.9 mostra que 14% dos utilizadores acharam suficiente a qualidade dos comandos do Controlador de Projectos, 36% acharam boa a qualidade implementada e 50% considerou que a qualidade dos comandos é muito boa. Um dos utilizadores cegos considerou que os comandos do controlador de projectos é boa no seu geral e o outro utilizador considerou muito bom o *design* dos comandos, porque permitem a criação de projectos e ficheiros de modelos de uma forma muito rápida e acessível.

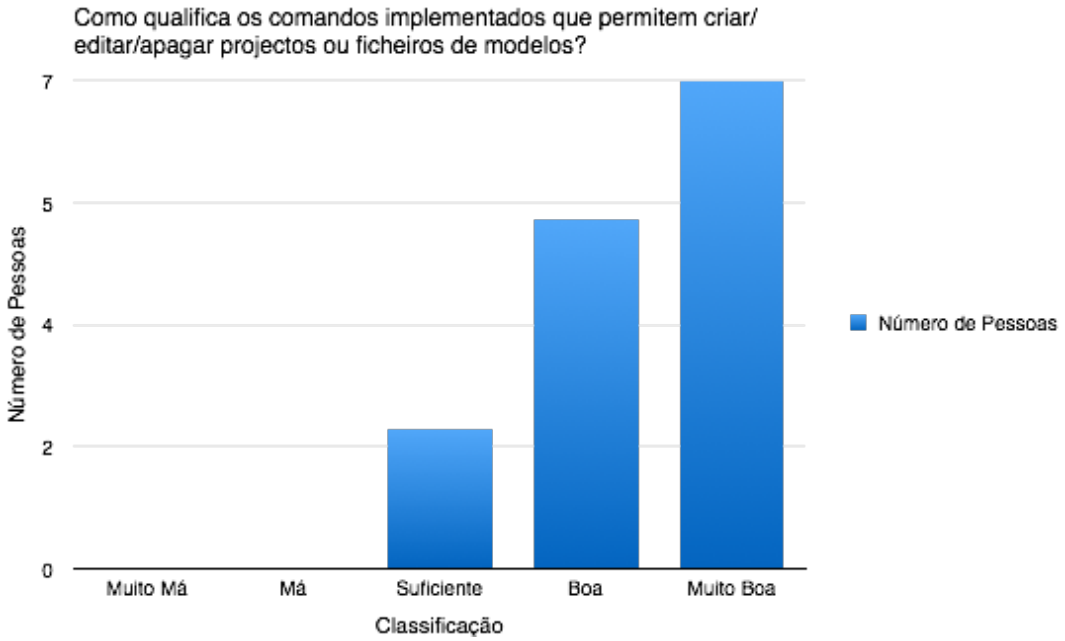


Figura 7.9: Resultados da qualidade dos comandos do Controlador de Projectos.

Não houve nenhuma sugestão de alteração dos comandos para o Controlador de Projectos, sendo que todos os participantes concordaram com a lógica implementada para iniciar o programa, terminar, criar/editar/apagar projectos e/ou ficheiros de modelos de requisitos, entre outros.

No que diz respeito à qualidade dos comandos para modelar um Modelo KAOS, a Figura 7.10 apresenta que 21% dos utilizadores acharam a qualidade suficiente, 50% consideraram a qualidade como boa e 29% entendeu que a qualidade é muito boa. Os utilizadores não videntes seleccionaram a qualidade como boa. Não houve sugestões no que diz respeito à pergunta que solicitava que tipo de alterações faria nos comandos que foram designados para este modelo.

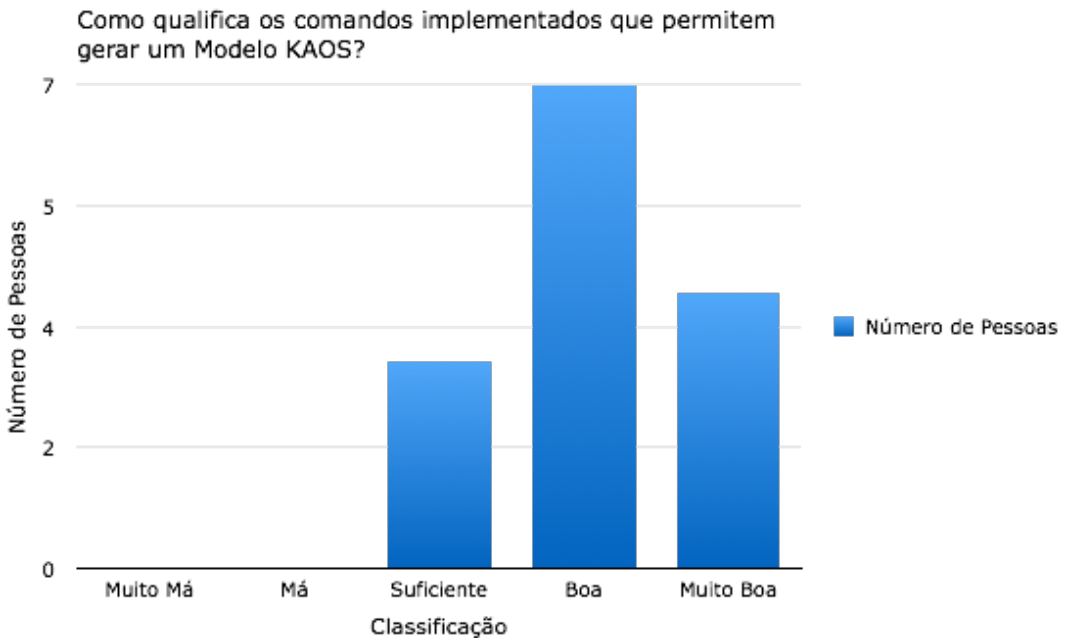


Figura 7.10: Resultados da qualidade dos comandos do Modelo KAOS.

Sobre a pergunta "Como qualifica os comandos implementados que permitem gerar um Modelo Conceptual?" as opiniões ficaram mais divididas entre o suficiente e bom, com ambas as classificações terem os valores de 43% e a classificação de muito boa com 14%, como se pode verificar na Figura 7.11. Os dois utilizadores cegos consideraram a qualidade dos comandos como suficiente e boa.

Houve algumas anotações em relação aos comandos deste modelo. A primeira é que o comando "add association" é muito grande e deveria ser repartida em "find c1", "find c2" e "add association one to many" e a segunda opinião é sobre a inclusão do tipo *optional* quando se define a multiplicidade da associação pode confundir como a omissão desse parâmetro. Em relação à primeira crítica, a semântica do comando "find" neste contexto apenas foca o último elemento especificado no parâmetro em todos os modelos, logo ir-se-ia perder a sua homogeneidade. Em referência à crítica do tipo *optional* nas multiplicidades, decidiu-se manter essa notação porque é a palavra mais comum quando se faz referência a zero ou mais ocorrências numa multiplicidade.

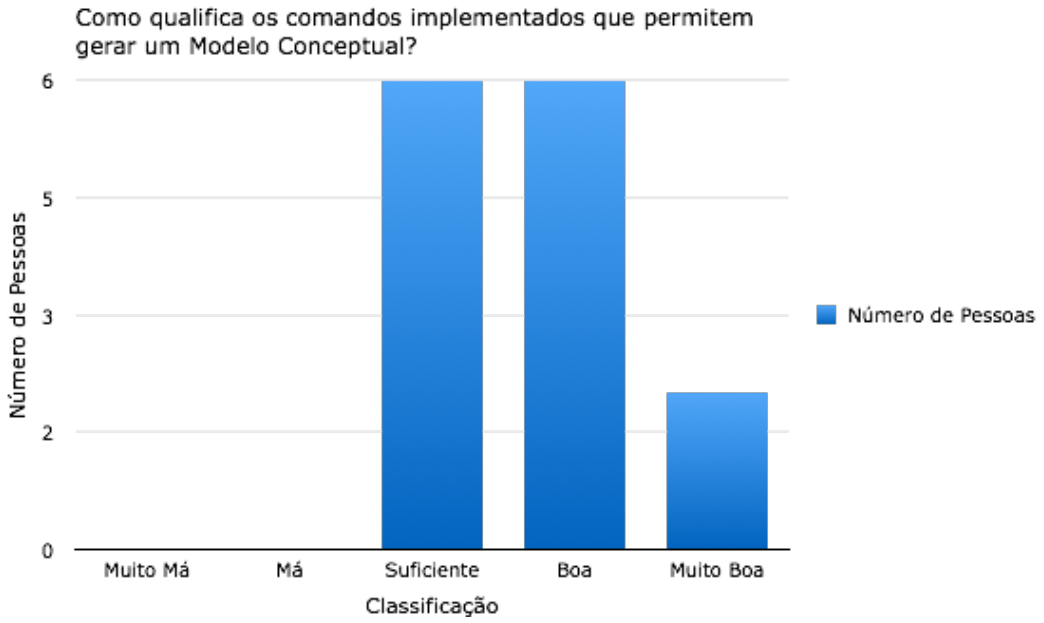


Figura 7.11: Resultados da qualidade dos comandos do Modelo Conceptual.

Relativamente aos resultados do Modelo de *Features*, os resultados da Figura 7.12 mostram que 43% considerou a qualidade como boa e 57% considerou que a qualidade é muito boa. Os utilizadores cegos qualificaram a qualidade como muito boa, referenciando que todos os comandos ajudaram na modelação que pretendiam executar na aplicação. Não houve *feedback* negativo em relação ao Modelo de *Features*, mas sim referências positivas à forma como foram desenhadas todas as operações.

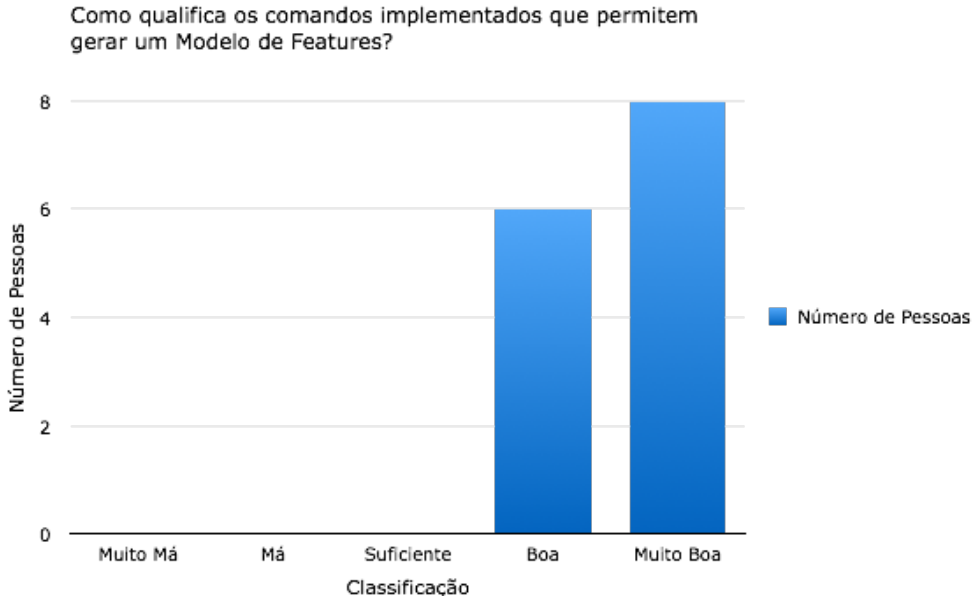


Figura 7.12: Resultados da qualidade dos comandos do Modelo de *Features*.

A última questão desta secção foi questionada a qualidade do *feedback* apresentado em cada operação. A Figura 7.13 mostra que 21% dos utilizadores consideraram suficiente a

qualidade demonstrada, 50% achou a qualidade boa e 29% entendeu que a qualidade do *feedback* é muito boa.

No que diz respeito a alterações, grande maioria dos utilizadores (incluindo os cegos) apenas fizeram reparos na qualidade da voz, ou seja, que a fala deveria ser mais suavizada e não tão robótica. Durante o ciclo de implementação tentou-se integrar as vozes do projecto MBROLA[3] com o *plugin* FreeTTS (responsável pela sintetização de voz), contudo sem sucesso. Não obstante disso, é algo a corrigir no futuro.

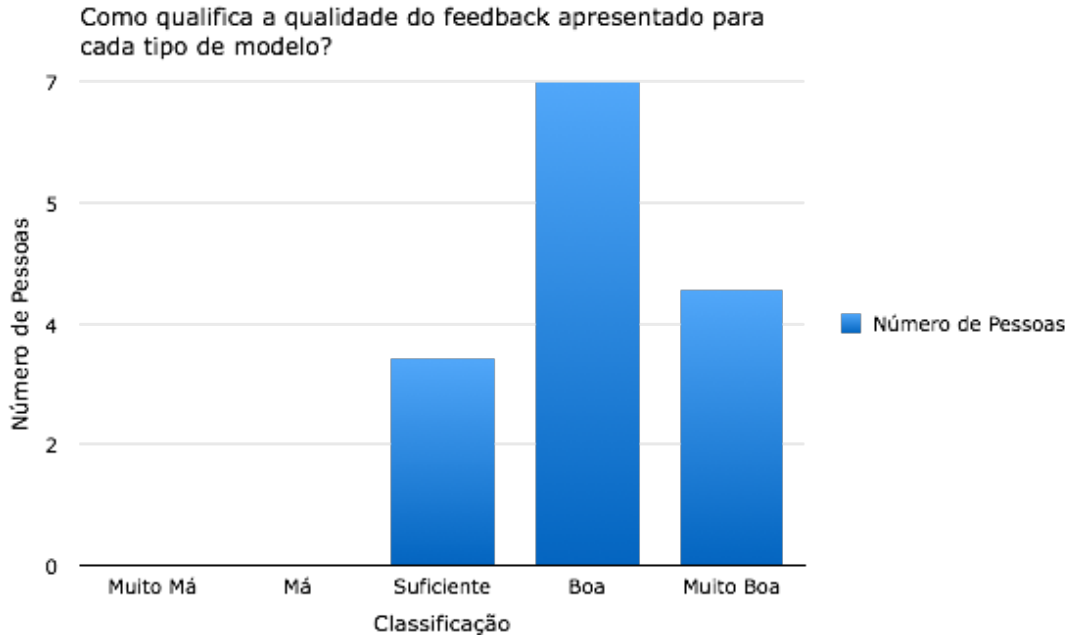


Figura 7.13: Resultados da qualidade do *feedback*.

7.4.4 Aspectos gerais do VoiceToModel

Nesta última subsecção do questionário, pretende-se que o utilizador refira detalhes de sistemas de reconhecimentos de voz já utilizados e que comente aspectos gerais da ferramenta VoiceToModel.

A Figura 7.14 mostra que 21%, 2 utilizadores cegos e 1 participante vidente, já usaram aplicações de reconhecimento de voz, nomeadamente o Google Talk[4] e S Voice[5] da Samsung. Os restantes 79% nunca usaram a ferramenta, o que levou a que os resultados tivessem maior volume tanto na duração das tarefas, como no número de operações realizadas devido à inexperiência. Para além do sistema de reconhecimento de voz não ser 100% eficaz, muito dos utilizadores não cegos não reagiram da melhor forma às adversidades, como por exemplo não suspender o sistema quando reclamam dos problemas de

[3]MBROLA: http://tcts.fpms.ac.be/synthesis/mbrola.html (Acesso feito em Setembro de 2014)
[4]Google Talk: http://www.google.com/talk/whatsnew_more.html (Acesso feito em Setembro de 2014)
[5]S Voice: http://goo.gl/MGiDD5 (Acesso feito em Setembro de 2014)

forma oral, o que originou mais erros de interpretações dos comandos devido à inexperiência do uso deste tipo de tecnologias. Por outro lado, os utilizadores cegos tiveram uma boa resposta ao sistema VoiceToModel graças ao *know-how* já obtido.

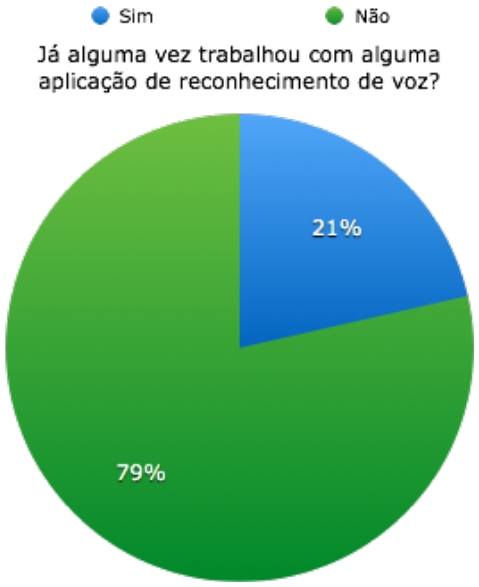


Figura 7.14: Resultados do número de aplicações de reconhecimento de voz já usadas.

Na Figura 7.15 o número de resposta é consensual, 100% dos utilizadores usaram pela primeira vez uma ferramenta que permite a modelação com foco na acessibilidade para pessoas com limitações físicas, cegas ou até mesmo para qualquer pessoa que não tenha qualquer tipo de deficiência.

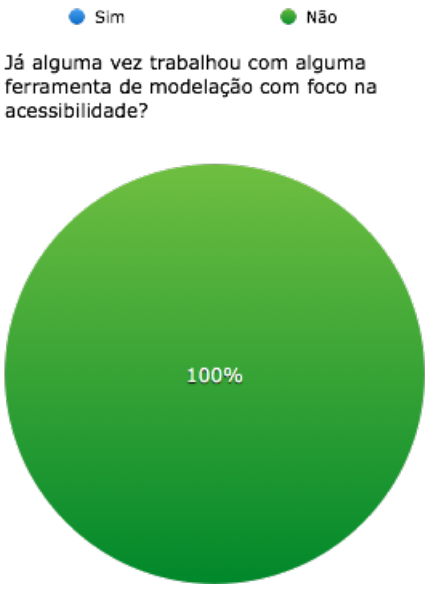


Figura 7.15: Resultados do uso de aplicações de modelação com foco na acessibilidade.

A questão "Quais são os pontos fortes da ferramenta VoiceToModel, na sua opinião?" tem o objectivo de recolher os pontos fortes da aplicação na perspectiva do utilizador. Os

pontos fortes que foram descritos são:

- *Feedback* do modelo de forma fácil/adequado;
- Permite aos utilizadores (com ou sem alguma incapacidade quer a nível motor, quer a nível sensorial) poderem criar modelos através de comandos de voz;
- Criar os modelos sem a necessidade de os fazer "à mão", o que poderá levar ao aumento da produtividade;
- Grande variedade de comandos que permitem um "relativamente fácil" desenho de modelos KAOS, conceptual e de *features*;
- Comandos claros e simples de entender;
- A noção de apontador - comando "find" - é muito interessante e útil;
- Comando "undo command" muito útil para quando uma acção desencadeada não foi a desejada.

O mestre em Engenharia Informática cego frisou que, quando realizou cadeiras do curso que usavam o UML (e.g., diagrama de classes, *use cases*, diagramas de interacção), combinava com o professor da cadeira que notação deveria usar e com o auxílio do bloco de notas descrevia em notação textual o que deveria fazer com ferramentas CASE. Evidenciou que esta ferramenta poderia ser integrada nas aulas para que pessoas com dificuldades físicas ou que sejam cegas pudessem modelar o sistema sem se preocuparem com a notação gráfica e, desta forma, facilitando a integração no trabalho em equipa. Em relação aos pontos fracos, os utilizadores expressaram as seguintes críticas:

- Não há interrupção do *feedback*, ou seja, o utilizador deve esperar pela resposta ser toda expressa (falada pelo sistema) mesmo se ele já ouviu o que gostaria de saber;
- O reconhecimento de voz ainda não é perfeito, ocorrendo falhas na interpretação dos comandos expressados pelo utilizador;
- Problemas com o sotaque.

O FreeTTS, responsável pela sintetização de voz, não permite na versão que o sistema VoiceToModel actualmente integra, a possibilidade de fazer *stop* quando se está a ouvir um *feedback* de uma operação que foi executada. O máximo que é possível fazer nesta versão seria baixar o volume através de um atalho do teclado, contudo o utilizador teria que aguardar que o sistema acabasse de sintetizar o *feedback* completo. Todavia é algo a ter em consideração para trabalho futuro, caso seja integrada uma nova versão do sintetizador de voz.

Como já foi mencionado em várias secções, as opções tecnológicas estão limitadas para serviços *open source* e com base nessas ferramentas foram integradas as melhores *frameworks* que existem ao dispor. A aplicação VoiceToModel está desenhada com os componentes separados, como se pode ver na secção 6.3, logo o componente de reconhecimento de voz pode ser facilmente alterável.

Não foi possível a integração de outros tipos de modelos acústicos que permitissem à aplicação VoiceToModel percepcionar outros sotaques porque era necessário mais tempo de implementação e testes, e tal não foi possível devido ao tempo disponível durante a elaboração da dissertação.

A questão "Que alterações faria para o VoiceToModel?" possibilitou aos utilizadores darem as suas opiniões no que diz respeito a modificações. As sugestões foram:

- Alteração da API de reconhecimento de voz;
- Alteração da voz do sintetizador para um mais suave;
- Navegação através do teclado ou menus para pessoas cegas;
- Adequar a recepção da voz do utilizador de acordo com a sua nacionalidade ou sotaque.

No que diz respeito à alteração da API de reconhecimento e sintetização de voz, como foi explicado anteriormente, a modificação é acessível graças à forma como a aplicação foi desenhada (secção 6.3). A questão do sistema conseguir adaptar o sotaque do utilizador também já foi explicado que por falta de tempo não foi possível integrar outros modelos acústicos que permitissem o suporte de vários sotaques.

Por fim, a sugestão de integrar o estilo de interacção através de atalhos do teclado ou menus também é interessante e deve ser analisado como trabalho futuro de modo a reforçar o conceito de acessibilidade.

A Figura 7.16, que referência a questão "Acha que as pessoas com limitações físicas ou cegas irão sentir-se capazes de usar a ferramenta no futuro (mudando ou não a API de reconhecimento de voz) e gerar modelos de acordo com a especificação pedida?", mostra que 93% dos utilizadores consideram que a ferramenta está bem especificada para ser usada e 7% tem opinião contrária.

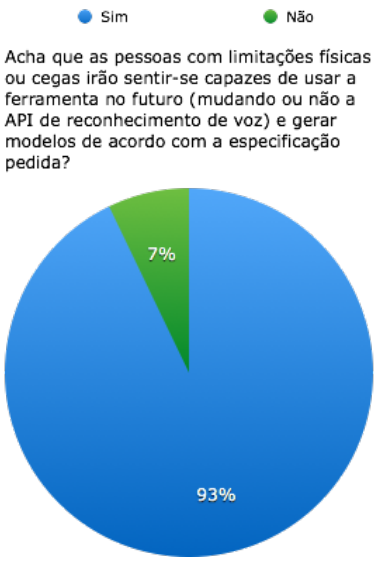


Figura 7.16: Resultados da opinião se o VoiceToModel está pronto para os utilizadores.

O motivo pela qual o utilizador acha que a aplicação não está preparada para que pessoas com deficiências possam usar a aplicação é por causa do sistema de reconhecimento de voz. Como já mencionado várias vezes, a aplicação VoiceToModel é uma prova de conceito e não um produto, e a API de reconhecimento de voz pode ser facilmente substituída por uma mais eficiente. Os restantes 13 participantes, incluindo os utilizadores cegos, concordam que a aplicação já é competente o suficiente para gerar modelos de requisitos através da voz.

Na última questão (comentários e observações) os utilizadores partilharam que as pessoas com limitações físicas em princípio deverão conseguir utilizar este programa adequadamente; a curva de aprendizagem inicial pode ser demasiado grande, podendo demorar bastante tempo a interiorizar os comandos; e de uma forma geral, o tema é muito interessante e certamente o avanço de sua investigação irá abrir novos caminhos tanto a pessoas com alguma deficiência quanto a utilizadores comuns que podem usar a ferramenta para aumentar sua produtividade.

7.5 Ameaças à avaliação

Há que ter em conta que, num processo de avaliação, podem ocorrer factores que podem por em causa os resultados e as conclusões obtidas. Alguns desses factores podem dever à escassez de indivíduos inseridos no mundo industrial, o que seria uma mais-valia na avaliação realizada para esta dissertação, porque permitiria fornecer mais resultados em categorias diferentes. No entanto, quase todos os utilizadores presentes na avaliação conhecem o ambiente Eclipse e os conhecimentos necessários para modelar.

A avaliação apenas incluiu 2 pessoas cegas, e numa situação ideal, dever-se-ia incluir mais pessoas com este género de deficiências e também com outras limitações físicas. Contudo já foi feito um enorme esforço, em colaboração com a ACAPO, para ter acesso a 2 pessoas com conhecimentos técnicos em Informática e outra com um mestrado concluído na área da Engenharia Informática.

A questão do caso de estudo apresentado para os utilizadores também pode constituir uma ameaça devido à sua simplicidade. No entanto, esta escolha deve-se ao facto dos utilizadores que não são cegos enfrentarem pela primeira vez uma aplicação com um sistema de reconhecimento de voz e, para pessoas cegas, que não têm o conhecimento completo para modelar não ficarem confusas com o tamanho da informação que receberam de uma só vez (conhecimentos dos modelos e dos comandos). Como se previa que a curva de aprendizagem não iria ser curta, dado que era primeira vez que iriam usar a ferramenta, apresentou-se um cenário simples com o intuito de analisar se os utilizadores conseguem modelar o sistema com base nos comandos implementados.

A avaliação apenas contém 14 testes realizados, o que pode ser visto como uma ameaça dado ao número reduzido de avaliações. Apesar dos números reduzidos, os testes que foram realizados apresentam resultados promissores e de aceitação.

7.6 Sumário

Neste capítulo começou-se por referir o planeamento para a avaliação experimental tais como o tipo de utilizadores que iriam usar a ferramenta, o estudo de alternativas de ferramentas que pudessem ser incluídas na avaliação para servir como comparação, a especificação das tarefas e uma possível limitação para a avaliação.

Depois de definido o planeamento, foram estabelecidas métricas para análise de resultados. Também foram apresentadas as opiniões dadas pelos utilizadores com base no questionário e, finalmente, foram descritos alguns factores que podem ser considerados como ameaças à validação dos resultados obtidos na avaliação.

Com esta avaliação foi possível verificar, numa visão geral, que a ferramenta é bastante aceite, útil e cumpre o objectivo do utilizador modelar o sistema usando o VoiceToModel não só para utilizadores videntes, como também para utilizadores cegos que são a grande motivação da dissertação.

8

Conclusão

Para concluir a dissertação de mestrado, este capítulo vai analisar o desenvolvimento durante a elaboração do trabalho e a apresentação de um conjunto de ideias para que a investigação desta área seja mais enriquecedora para o futuro.

8.1 Resumo

Nesta dissertação de mestrado foi proposto o desenvolvimento de uma *framework* que usa a voz como *input* para estruturar os modelos de requisitos, através de três Linguagens para Domínio Específico para a especificação de modelos KAOS, conceptuais e de *features*.

Começou-se por analisar, no capítulo 2, a metodologia usada no processo de Engenharia de Requisitos, mais especificamente no contexto da elicitação de requisitos. Também foi feito um estudo sobre os modelos de requisitos orientados a objectivos, objectos e o modelo de variabilidade. Para usar a voz como *input* na modelação, foram discutidos no capítulo 3 várias tecnologias de reconhecimento e sintetização de voz, tendo sido integradas as *frameworks* Sphinx-4, Google Speech API e FreeTTS. De forma a unir os conceitos dos modelos de requisitos e as tecnologias de reconhecimento de voz foi feita uma análise, com base no capítulo 4 de algumas técnicas existentes no Desenvolvimento Orientado a Modelos, nomeadamente a metamodelação, que permite definir os modelos de requisitos com base numa sintaxe abstracta.

Também foi feita uma investigação a trabalhos relacionados, apresentados no capítulo 5, avaliando e discutindo a acessibilidade em ferramentas CASE, os principais desafios enfrentados por engenheiros de requisitos com deficiências (visuais ou motoras) em

projectos de *software* e quais os modelos de requisitos que utilizam estratégias de acessibilidade.

Depois da análise sobre as técnicas, tecnologias e trabalhos existentes, o capítulo 6 descreve a abordagem VoiceToModel implementada durante a elaboração da dissertação, isto é, como foi construída a arquitectura, o modelo de interacção, a descrição dos comandos de cada modelo de requisitos e as técnicas usadas durante a implementação.

Com base na avaliação, descrita no capítulo 7, foi possível concluir que os utilizadores tiveram uma boa aceitação com a aplicação e que, seguindo esta metodologia, é possível integrar pessoas cegas ou com outro tipo de deficiências na modelação de um projecto informático e estruturar os modelos de acordo com as suas expectativas.

8.2 Contribuições

No início deste trabalho foram estabelecidas duas contribuições, que se consideram as principais: inclusão de engenheiros de requisitos com problemas físicos (especialmente cegos) na fase de elicitação de requisitos; e prover uma maior colaboração no processo de elicitação (mesmo para aqueles sem deficiência). As avaliações no experimento comprovaram o que foi previamente definido, isto é, tanto os utilizadores cegos, como aqueles que têm visão, deixaram boas críticas a uma ferramenta que implementa uma abordagem de acessibilidade para a modelação de requisitos.

Os utilizadores cegos tiveram uma boa interacção com a ferramenta e ficaram satisfeitos com o *feedback* que a aplicação retornava em cada operação executada. A noção de expressar comando, executar, e retornar *feedback* audível não só mantinha o utilizador cego ao corrente da actividade de modelação, como também serializava a informação e gerava de forma automática os requisitos em notação gráfica. Assim qualquer outro utilizador pode realizar modificações no modelo, seja no editor gráfico ou através de comandos via voz. Todos estes aspectos promovem a integração de um membro com limitações físicas (especialmente cegos) no desenvolvimento de um projecto informático.

8.3 Limitações

Como já foi mencionado em várias secções, uma das principais limitações prende-se ao facto das APIs de reconhecimento de voz terem falhas no reconhecimento de palavras. Isso deve-se ao facto que nesta dissertação de mestrado apenas tivesse havido acesso a ferramentas *open source*. Outra limitação é o facto da API de reconhecimento de voz da Google apenas disponibilizar 50 pedidos diários para usar o seu serviço. Contudo, a ferramenta foi implementada num estilo baseado em camadas, ou seja, é possível modificar a camada de reconhecimento de voz sem necessidade de restruturar o programa. Desta forma é possível trocar de APIs para um sistema mais fiável.

Outra limitação é o facto de não incluir pessoas na indústria ou a escassez de indivíduos com deficiências. Essa escassez deve-se ao facto de ter sido complicado encontrar

pessoas com conhecimentos na área de informática e que tivessem a disponibilidade para realizar o experimento.

8.4 Trabalho futuro

Dado que este tipo de abordagem usada no contexto do Desenvolvimento Orientado a Modelos é relativamente recente, existem várias sugestões que podem potenciar esta ferramenta e o conceito de acessibilidade. Uma forma de melhorar a ferramenta passa pela automatização da geração de código para os comandos. A primeira alternativa seria com base em anotações, tendo já nesta dissertação ter sido definida a sintaxe das anotações, a geração de código que depois estaria sujeito a refinamento. Outra alternativa seria construir uma linguagem (e.g., máquina de estados) que daria a possibilidade de estabelecer o tipo de modelo de interacção e a respectiva geração de código.

Também será interessante estender os modelos de requisitos definidos nesta dissertação, isto é, implementar mais conceitos e propriedades com base nos comandos que foram estabelecidos. Um modelo cognitivo de requisitos encaixaria neste contexto, graças à sua simplicidade e capacidade de integrar os *stakeholders* no processo de elicitação e uso do VoiceToModel. Outra sugestão seria a implementação de comandos para obter métricas mais complexas, dado que os comandos actuais apenas fornecem informação da dimensão do respectivo modelo.

No que diz respeito ao conceito de acessibilidade, seria interessante mapear os comandos via voz para uma perspectiva de menus e do teclado, que é outros dos paradigmas do modelo de interacção.

Bibliografia

[Ara+05] J. Araújo, E. Baniassad, P. Clements, A. Moreira, A. Rashid e B. Tekinerdogan. "Early aspects: The current landscape". Em: *Technical Notes, CMU/SEI and Lancaster University* (2005).

[Asa+12] M. Asadi, E. Bagheri, B. Mohabbati e D. Gašević. "Requirements engineering in feature oriented software product lines: an initial analytical study". Em: *Proceedings of the 16th International Software Product Line Conference - Volume 2*. SPLC '12. Salvador, Brazil: ACM, 2012. DOI: 10.1145/2364412.2364419. URL: http://doi.acm.org/10.1145/2364412.2364419.

[AN04] T. Ayres e B. Nolan. "Voice activated command and control with Java-enabled speech recognition over Wifi". Em: *Proceedings of the 3rd international symposium on Principles and practice of programming in Java*. Las Vegas, Nevada: Trinity College Dublin, 2004. ISBN: 1-59593-171-6. URL: http://dl.acm.org/citation.cfm?id=1071565.1071586.

[Bas92] V. R. Basili. *Software modeling and measurement: the Goal/Question/Metric paradigm*. Rel. téc. Techreport UMIACS TR-92-96, University of Maryland at College Park, College Park, MD, USA, 1992.

[BG01] J. Bézivin e O. Gerbé. "Towards a precise definition of the OMG/MDA framework". Em: *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*. IEEE. 2001, pp. 273–280.

[Béz+05] J. Bézivin, G. Hillairet, F. Jouault, I. Kurtev e W. Piers. "Bridging the ms/dsl tools and the eclipse modeling framework". Em: *Proceedings of the International Workshop on Software Factories at OOPSLA*. Vol. 5. 2005.

[BE98] P. Blenkhorn e D. G. Evans. "Using speech and touch to enable blind people to access schematic diagrams". Em: *Journal of Network and Computer Applications* 21.1 (1998), pp. 17–29.

[Bud04] F. Budinsky. *Eclipse modeling framework: a developer's guide*. Addison-Wesley Professional, 2004.

[CR94] V. Caldiera e H. D. Rombach. "The goal question metric approach". Em: *Encyclopedia of software engineering* 2.1994 (1994), pp. 528–532.

[Chr+96] Christensen, Mark, Chang e Carl. "Blueprint for the Ideal Requirements Engineer". Em: *IEEE Softw.* (1996). DOI: 10.1109/MS.1996.506457. URL: http://dx.doi.org/10.1109/MS.1996.506457.

[CMU13] CMUSphinx. *Building Language Model*. 2013. URL: http://cmusphinx.sourceforge.net/wiki/tutoriallm (acedido em 20/09/2014).

[Coh+05] R. F. Cohen, R. Yu, A. Meacham e J. Skaff. "PLUMB: Displaying Graphs to the Blind Using an Active Auditory Interface". Em: *Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility*. Assets '05. Baltimore, MD, USA: ACM, 2005, pp. 182–183. ISBN: 1-59593-159-7. DOI: 10.1145/1090785.1090820. URL: http://doi.acm.org/10.1145/1090785.1090820.

[Cre13] P. Creek. *Eclipse Design Patterns*. 2013. URL: http://www.programcreek.com/2013/02/eclipse-design-patterns-composite-in-workspace/ (acedido em 20/09/2014).

[CHE05] K. Czarnecki, S. Helsen e U. Eisenecker. "Formalizing cardinality-based feature models and their specialization". Em: *Software Process: Improvement and Practice* 10.1 (2005), pp. 7–29.

[DLF93] A. Dardenne, A. van Lamsweerde e S. Fickas. "Goal directed requirements acquisition". Em: *Science of Computer Programming* 20 (1993). ISSN: 0167-6423. DOI: http://dx.doi.org/10.1016/0167-6423(93)90021-G. URL: http://www.sciencedirect.com/science/article/pii/016764239390021G.

[DBB52] K. Davis, R. Biddulph e S. Balashek. "Automatic recognition of spoken digits". Em: *The Jornal of the Acoustical Society of America* 24 (1952), p. 637.

[DS99] L. Dawson e P. Swatman. "The use of object-oriented models in requirements engineering: a field study". Em: *Proceedings of the 20th international conference on Information Systems*. Charlotte, North Carolina, USA: Association for Information Systems, 1999, pp. 260–273. URL: http://dl.acm.org/citation.cfm?id=352925.352949.

[Dir04] A. Dirksen. "Speech synthesis and electronic dictionaries". Em: *On Speech and Language* (2004), p. 57.

[Dut97] T. Dutoit. *An introduction to text-to-speech synthesis*. Vol. 3. Springer, 1997.

[End01] J. Enden. "Java speech API". Em: *Jini and Advanced Features of Java (2 cu). Seminar at the Department of Computer Science*. Citeseer. 2001, pp. 48–66.

[FS02] J. M. Francioni e A. C. Smith. "Computer Science Accessibility for Students with Visual Disabilities". Em: *SIGCSE Bull.* 34.1 (fev. de 2002), pp. 91–95. ISSN: 0097-8418. DOI: 10.1145/563517.563372. URL: http://doi.acm.org/10.1145/563517.563372.

[Gro13] S. Group. *Chaos Manifesto 2013*. 2013. URL: http://versionone.com/assets/img/files/ChaosManifesto2013.pdf (acedido em 20/09/2014).

[HN10] Z. Hua e W. L. Ng. "Speech recognition interface design for in-vehicle system". Em: *Proceedings of the 2nd International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. AutomotiveUI '10. Pittsburgh, Pennsylvania: ACM, 2010. ISBN: 978-1-4503-0437-5. URL: http://doi.acm.org/10.1145/1969773.1969780.

[Hun10] A. Hunter. *Ecplise - Graphical Modeling Framework*. 2010. URL: http://www.eclipse.org/modeling/gmp/?project=gmf-notation#gmf-notation (acedido em 20/09/2014).

[IT07] R. IT. *A KAOS Tutorial, version 1.0*. 2007. URL: http://www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf (acedido em 20/09/2014).

[IT14] R. IT. *Objectiver*. 2014. URL: http://www.objectiver.com (acedido em 20/09/2014).

[JK06] F. Jouault e I. Kurtev. "Transforming Models with ATL". Em: *Satellite Events at the MoDELS 2005 Conference*. Ed. por J.-M. Bruel. Vol. 3844. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 128–138. ISBN: 978-3-540-31780-7. DOI: 10.1007/11663430_14. URL: http://dx.doi.org/10.1007/11663430_14.

[Kan+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak e A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Rel. téc. Carnegie-Mellon University Software Engineering Institute, 1990.

[Kas+05] C. Kastner, T. Thum, G. Saake, J. Feigenspan, T. Leich, F. Wielgorz e S. Apel. *An Eclipse plug-in for Feature-Oriented Software Development*. 2005. URL: http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/ (acedido em 20/09/2014).

[Kee07] S. Keele. *Guidelines for performing systematic literature reviews in software engineering*. Rel. téc. Technical report, EBSE Technical Report EBSE-2007-01, 2007.

[Ken96] A. R. Kennel. "Audiograf: a diagram-reader for the blind". Em: *Proceedings of the second annual ACM conference on Assistive technologies*. ACM. 1996, pp. 51–56.

[Kin+04] A. King, P. Blenkhorn, D. Crombie, S. Dijkstra, G. Evans e J. Wood. "Presenting UML Software Engineering Diagrams to Blind People". Em: *Computers Helping People with Special Needs*. Ed. por K. Miesenberger, J. Klaus, W. Zagler e D. Burger. Vol. 3118. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, pp. 522–529. ISBN: 978-3-540-22334-4. DOI: 10.1007/978-3-540-27817-7_76. URL: http://dx.doi.org/10.1007/978-3-540-27817-7_76.

[KWB03] A. Kleppe, J. Warmer e W. Bast. *MDA Explained: The Model Driven Architecture : Practice and Promise*. The Addison-Wesley object technology series. Addison-Wesley, 2003. ISBN: 9780321194428.

[Kol+13] D. Kolovos, L. Rose, A. García-domínguez e R. Paige. *The Epsilon Book*. 2013.

[KS98] G. Kotonya e I. Sommerville. *Requirements engineering: processes and techniques*. Worldwide series in computer science. John Wiley & Sons, 1998. ISBN: 9780471972082.

[LP03] S. Lahtinen e J. Peltonen. "Enhancing usability of UML CASE-tools with speech recognition". Em: *Human Centric Computing Languages and Environments, 2003. Proceedings. 2003 IEEE Symposium on*. IEEE. 2003, pp. 227–235.

[LA01] V. Lamsweerde e Axel. "Goal Oriented Requirements Engineering: A Guided Tour". Em: *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*. Washington, DC, USA: IEEE Computer Society, 2001. URL: http://dl.acm.org/citation.cfm?id=882477.883624.

[Lap05] A. Lapouchnian. *Goal-Oriented Requirements Engineering: An Overview of the Current Research*. Rel. téc. 2005.

[Led+01] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle e P. Volgyesi. "The generic modeling environment". Em: *Workshop on Intelligent Signal Processing, Budapest, Hungary*. Vol. 17. 2001.

[Lem99] S. Lemmetty. *History and Development of Speech Synthesis*. 1999. URL: http://www.acoustics.hut.fi/publications/files/theses/lemmetty_mst/chap2.html (acedido em 20/09/2014).

[Lev66] V. I. Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals". Em: *Soviet Physics Doklady*. Vol. 10. 1966, p. 707.

[LBD02] T. Lodderstedt, D. Basin e J. Doser. "SecureUML: A UML-Based Modeling Language for Model-Driven Security". English. Em: *UML 2002 - The Unified Modeling Language*. Ed. por J.-M. Jezequel, H. Hussmann e S. Cook. Vol. 2460. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 426–441. ISBN: 978-3-540-44254-7. DOI: 10.1007/3-540-45800-X_33. URL: http://dx.doi.org/10.1007/3-540-45800-X_33.

[Lud07] S. Ludi. "Introducing accessibility requirements through external stakeholder utilization in an undergraduate requirements engineering course". Em: *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society. 2007, pp. 736–743.

[Mag14] N. Magic. *No Magic, Inc - Unified Modeling Language (UML), SYSML, UPDM, SOA, Business Process Modeling Tools*. 2014. URL: http://www.nomagic.com (acedido em 20/09/2014).

[MH05] R. Matulevicius e P. Heymans. "Analysis of KAOS Meta-model". Em: *Namur University: Computer Science Department: Belgium* (2005).

[Mel+02] S. Mellor, K. Scott, A. Uhl e D. Weise. "Model-Driven Architecture". English. Em: *Advances in Object-Oriented Information Systems*. Ed. por J.-M. Bruel e Z. Bellahsene. Vol. 2426. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, pp. 290–297. ISBN: 978-3-540-44088-8. DOI: 10.1007/3-540-46105-1_33. URL: http://dx.doi.org/10.1007/3-540-46105-1_33.

[MHM10] D. Moody, P. Heymans e R. Matulevičius. "Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation". English. Em: *Requirements Engineering* 15.2 (2010), pp. 141–175. ISSN: 0947-3602. DOI: 10.1007/s00766-010-0100-1. URL: http://dx.doi.org/10.1007/s00766-010-0100-1.

[Nob96] J. Noble. "Scribble: a diagram editor with a minimal interface". Em: *Computer-Human Interaction, 1996. Proceedings., Sixth Australian Conference on*. Nov. de 1996, pp. 162–168. DOI: 10.1109/OZCHI.1996.560006.

[NE00] B. Nuseibeh e S. Easterbrook. "Requirements engineering: a roadmap". Em: *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. Limerick, Ireland: ACM, 2000, pp. 35–46. ISBN: 1-58113-253-0. DOI: 10.1145/336512.336523. URL: http://doi.acm.org/10.1145/336512.336523.

[OMG11] OMG. *Meta Object Facility (MOF) Core Specification Version 2.4.1*. OMG Available Specification. Object Management Group, 2011.

[OMG09] O. M. G. (OMG). *Unified Modeling Language (UML) Infrastructure specification, version 2.2*. 2009. URL: http://www.omg.org/spec/UML/2.2/Infrastructure/PDF (acedido em 20/09/2014).

[Ora14] Oracle. *Java Speech API Frequently Asked Questions*. 2014. URL: http://www.oracle.com/technetwork/java/jsapifaq-135248.html (acedido em 20/09/2014).

[Pan+12] L. T. E. Pansanato, A. L. M. Bandeira, L. G. d. Santos e D. d. P. Pereira. "Projeto D4ALL: acesso e manipulação de diagramas por pessoas com deficiência visual". Em: *Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems*. Brazilian Computer Society. 2012, pp. 33–36.

[PSR12] L. T. Pansanato, C. E. Silva e L. Rodrigues. "Uma Experiência de Inclusão de Estudante Cego na Educação Superior em Computação". Em: 2012.

[PR96] F. C. N. Pereira e M. Riley. "Speech Recognition by Composition of Weighted Finite Automata". Em: *CoRR* cmp-lg/9603001 (1996).

[Pre09] R. Pressman. *Software Engineering: A Practitioner's Approach*. 7ª ed. New York, NY, USA: McGraw-Hill, Inc., 2009. ISBN: 0073375977, 9780073375977.

[Pro14] M. Project. *Mobiusing Advanced Technologies For Care At Home*. 2014. URL: http://www.match-project.org.uk/resources/tutorial/Speech_Language/Speech_Synthesis/Syn_6.html (acedido em 20/09/2014).

[RKR13] T. ur Rehman, M. N. A. Khan e N. Riaz. "Analysis of Requirement Engineering Processes, Tools/Techniques and Methodologies". Em: *I.J. Information Technology and Computer Science* (2013). DOI: 10.5815/ijitcs.2013.03.05.

[Rud13] T. Rudd. *Using Python to Code by Voice*. 2013. URL: http://pyvideo.org/video/1735/using-python-to-code-by-voice (acedido em 20/09/2014).

[SE05] M. Sabetzadeh e S. Easterbrook. "Traceability in Viewpoint Merging: A Model Management Perspective". Em: *Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering*. TEFSE '05. Long Beach, California: ACM, 2005, pp. 44–49. ISBN: 1-59593-243-7. DOI: 10.1145/1107656.1107667. URL: http://doi.acm.org/10.1145/1107656.1107667.

[Sch06] D. C. Schmidt. "Guest Editor's Introduction: Model-Driven Engineering". Em: *Computer* 39.2 (2006), pp. 25–31. ISSN: 0018-9162. DOI: http://doi.ieeecomputersociety.org/10.1109/MC.2006.58.

[Sep90] A. September. "IEEE Standard Glossary of Software Engineering Terminology". Em: *IEEE Std 610.12-1990* (dez. de 1990), pp. 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[SAW15] F. Soares, J. Araujo e F. Wanderley. "VoiceToModel: An Approach to Generate Requirements Models from Speech Recognition Mechanisms". Em: *Proceedings of the 30th Annual ACM Symposium On Applied Computing*. ACM. 2015.

[Sph14] Sphinx. *A speech recognizer written entirely in the Java programming language*. 2014. URL: http://cmusphinx.sourceforge.net/sphinx4/ (acedido em 20/09/2014).

[TGA02] G. H. Travassos, D. Gurov e E. Amaral. "Introdução a Engenharia de Software Experimental – Relatório Técnico: RT-ES-590/02". Em: *Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ* (2002).

[Tuf03] E. R. Tufte. *Envisioning Information*. Graphics Press (1990), 2003.

[Uni96] C. M. University. *What is speech recognition?* 1996. URL: http://www.speech.cs.cmu.edu/comp.speech/Section6/Q6.1.html (acedido em 20/09/2014).

[VKV00] A. Van Deursen, P. Klint e J. Visser. "Domain-Specific Languages: An Annotated Bibliography." Em: *Sigplan Notices* 35.6 (2000), pp. 26–36.

[Vaz+12] D. Vazhenina, I. Kipyatkova, K. Markov e A. Karpov. "State-of-the-art speech recognition technologies for Russian language". Em: *Proceedings of the 2012 Joint International Conference on Human-Centered Computer Environments*. HCCE '12. Aizu-Wakamatsu et al.: ACM, 2012, pp. 59–63. ISBN: 978-1-4503-1191-5. URL: http://doi.acm.org/10.1145/2160749.2160763.

[Von91] W. Von Kempelen. *Mechanismus der menschlichen Sprache*. Degen, 1791.

[W3C00] W3C. *JSpeech Grammar Format*. 2000. URL: http://www.w3.org/TR/jsgf/ (acedido em 20/09/2014).

[WLK02] W. Walker, P. Lamere e P. Kwok. "FreeTTS: a performance case study". Em: (2002).

[Wal+04] W. Walker, P. Lamere, P. Kwok, B. Raj, R. Singh, E. Gouvea, P. Wolf e J. Woelfel. *Sphinx-4: A Flexible Open Source Framework for Speech Recognition*. Rel. téc. Mountain View, CA, USA, 2004.

[WK03] J. Warmer e A. Kleppe. *The Object Constraint Language: Getting Your Models Ready for MDA*. 2ª ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN: 0321179366.

[WK05] P. L. Willie Walker e P. Kwok. *FreeTTS 1.2 - A speech synthesizer written entirely in the Java programming language*. 2005. URL: http://freetts.sourceforge.net/docs/index.php (acedido em 20/09/2014).

[wis14] wiseGEEK. *What are the Advantages of Speech Recognition?* 2014. URL: http://www.wisegeek.com/what-are-the-advantages-of-speech-recognition.htm (acedido em 20/09/2014).

[Woh+00] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell e A. Wesslén. *Experimentation in Software Engineering: An Introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000. ISBN: 0-7923-8682-5.

[Zav97] P. Zave. “Classification of research efforts in requirements engineering”. Em: *ACM Comput. Surv.* 29.4 (dez. de 1997), pp. 315–321. ISSN: 0360-0300. DOI: 10.1145/267580.267581. URL: http://doi.acm.org/10.1145/267580.267581.

Anexo dos comandos

A.1 Comandos do SimpleKAOS

Tabela A.1: Comandos do SimpleKAOS.

Comando	Descrição	Feedback	Nota	Complexidade
add root <goal>	Adiciona um *goal root*.	*Root <goal> has been added successfully*.		O(1)
add goal refinement OR <goal2> to <goal1>	Adiciona um *goal* com refinamento OR, sendo <goal1> o nome do *goal* pai e <goal2> o nome do *goal* a adicionar.	*Goal <goal2> with refinement OR has been added successfully*.		O(|G|+|R|)
add goal refinement OR <goal2>	Adiciona um *goal* com refinamento OR, sendo <goal2> o nome do *goal* a adicionar.	*Goal <goal2> with refinement OR has been added successfully*.	O parâmetro <goal2> é definido usando anteriormente os comandos de exemplo: - find <goal1> - find by position <number> - add root/goal refinement AND/OR <goal1>	O(1)

add goal refinement AND <goal2> to <goal1>	Adiciona um *goal* com refinamento AND, sendo <goal1> o nome do *goal* pai e <goal2> o nome do *goal* a adicionar.	*Goal <goal2> with refinement AND has been added successfully*.		O(|G|+|R|)
add goal refinement AND <goal2>	Adiciona um *goal* com refinamento AND, sendo <goal2> o nome do *goal* a adicionar.	*Goal <goal2> with refinement AND has been added successfully*.	O parâmetro <goal2> é definido usando anteriormente os comandos de exemplo: - find <goal1> - find by position <number> - add root/goal refinement AND/OR <goal1>	O(1)
add agent <agent> to <goal2>	Adiciona e associa um agente a um *goal*, sendo <goal2> o nome do *goal* e <agent> o nome do agente.	*Agent <agent> has been added successfully*.		O(|G|+|R|)
add agent <agent>	Adiciona e associa um agente a um *goal*.	*Agent <agent> has been added successfully*.	O parâmetro <goal2> é definido usando anteriormente os comandos de exemplo: - find <goal2> - find by position <number> - add root/goal refinement AND/OR <goal2>	O(1)
change goal name <old> to <new>	Muda o nome de um *goal*.	*Goal <old> was changed successfully to <new>*.		O(|G|+|R|)

change goal name <new>	Muda o nome de um *goal*.	*Goal <old> was changed successfully to <new>.*	O parâmetro <old> tem que ser especificado no comando anterior, por exemplo: - find <old> - find by position <number> - add root/goal refinement AND/OR <old>	O(1)
change agent name <old> to <new>	Muda o nome de um agente.	*Agent <old> was changed successfully to <new>.*		O(|A|)
change refinement OR to AND <goal>	Muda o tipo de um refinamento para OR>AND	*Goal <goal> with refinement OR was changed to AND successfully.*		O(|G|+|R|)
change refinement OR to AND	Muda o tipo de um refinamento para OR>AND	*Goal <goal> with refinement OR was changed to AND successfully.*	O parâmetro <goal> tem que ser especificado no comando anterior, por exemplo: - find <goal> - find by position <number> - add goal refinement OR <goal>	O(|G|+|R|)
change refinement AND to OR <goal>	Muda o tipo de um refinamento para AND>OR	*Goal <goal> with refinement AND was changed to OR successfully.*		O(|G|+|R|)
change refinement AND to OR	Muda o tipo de um refinamento para AND>OR	*Goal <goal> with refinement AND was changed to OR successfully.*	O parâmetro <goal> tem que ser especificado no comando anterior, por exemplo: - find <goal> - find by position <number> - add goal refinement AND <goal>	O(|G|+|R|)
delete goal <goal>	Apaga um *goal*	*Goal <goal> was deleted successfully.*		O(|G|+|R|)

delete goal	Apaga um *goal*	*Goal <goal> was deleted successfully*.	O parâmetro <goal> tem que ser especificado no comando anterior, por exemplo: - find <goal> - find by position <number> - add root/goal refinement AND/OR <goal> - change <goal>	O(|G|+|R|)
delete agent <agent>	Apaga um agente	*Agent <agent> was deleted successfully*.		O(|A|)
delete refinement OR <goal>	Apaga um refinamento OR de um *goal*	*Goal <goal> with refinement OR was deleted successfully*.		O(|G|+|R|)
delete refinement OR	Apaga um refinamento OR de um *goal*	*Goal <goal> with refinement OR was deleted successfully*.	O parâmetro <goal> tem que ser especificado no comando anterior, por exemplo: - find <goal> - find by position <number> - add goal refinement OR <goal> - change refinement AND to OR <goal>	O(|G|+|R|)
delete refinement AND <goal>	Apaga um refinamento AND de um *goal*	*Goal <goal> with refinement AND was deleted successfully*.		O(|G|+|R|)
delete refinement AND	Apaga um refinamento AND de um *goal*	*Goal <goal> with refinement AND was deleted successfully*.	O parâmetro <goal> tem que ser especificado no comando anterior, por exemplo: - find <goal> - find by position <number> - add goal refinement AND <goal> - change refinement OR to AND <goal>	O(|G|+|R|)

find <goal>	Foca um *goal* <goal> de modo a que futuros comandos como add/-change/delete possam omitir parâmetros	*Goal <name> found successfully.*		O(|G|+|R|)
find by position <number>	Foca um *goal* <name> dada a sua posição de modo a que futuros comandos como add/-change/delete possam omitir parâmetros	*Goal <name> with the position <number> found successfully.*	Exemplo: *root* tem posição 1.	O(|G|+|R|)
say total of goals	Indica o número total de *goals* que o Modelo KAOS tem.	*The KAOS Model has <number> goals.*		O(|G|+|R|)
say total of agents	Indica o número total de agentes que o Modelo KAOS tem.	*The KAOS Model has <number> agents.*		O(|G|+|R|)
say total of sub goals <goal>	Indica o número de *sub goals* que um *goal* tem.	*Goal <goal> has <childs> sub goals.*		O(|G|)
say total of sub goals	Indica o número de *sub goals* que um *goal* tem.	*Goal <goal> has <childs> sub goals.*	O parâmetro <goal> tem que ser especificado no comando anterior, por exemplo: - find <goal> - find by position <number> - add/change/delete	O(|G|)
say total of agents <goal>	Indica o número de agentes que um *goal* tem.	*Goal <goal> has <childs> agents.*		O(|G|+|R|+|A|)
say sub goals name <goal>	Indica o nome dos *sub goals* que um *goal* tem.	Goal <goal> has a refinement AND/OR with the following sub goals <sub_goals>.		O(|G|+|R|)

say sub goals name	Indica o nome dos *sub goals* que um *goal* tem.	*Goal <goal> has a refinement AND/OR with the following sub goals <sub_goals>.*	O parâmetro <goal> tem que ser especificado no comando anterior, por exemplo: - find <goal> - find by position <number> - add/change/delete	O(|G|)
say agents name <goal>	Indica o nome de agentes que um *goal* tem.	*Goal <name> has the following agents: <agents>.*		O(|G|+|R|+|A|)
say actual goal	Notifica qual o *goal* actual que está focado.	*The actual goal is <goal>.*		O(1)
sleep mode on	Sistema fica em *stand-by*.	VoiceToModel *will be on stand-by*.		O(1)
sleep mode off	Sistema fica novamente activo para reconhecer comandos.	VoiceToModel *is ready*.		O(1)
repeat last feedback	Repete o *feedback* da última operação realizada.	Conteúdo da mensagem do último comando.		O(1)
validate model	Informa se o modelo é valido de acordo com as regras de especificação forma. Caso não seja, o utilizador é informado dos *warnings*/erros.	Conteúdo proveniente das retrições estabelecidas.		O(|G|+|R|)
undo command	Desfaz o último comando realizado.	Conteúdo proveniente da as retrições estabelecidas.		O(1)
describe commands	Descrever os comandos principais para o controlador de projectos.	Conteúdo da lista de comandos.		O(1)

help mode on	Permite activar o modo ajuda quando o utilizador não diz correctamente um comando.	*Help mode* ON.		O(1)
help mode off	Desactiva o modo ajuda.	*Help mode* OFF.		O(1)
back to project controller	Volta para o controlador de projectos.	*Back to project controller*.		O(1)

A.2 Comandos do Modelo Conceptual

Tabela A.2: Comandos do Modelo Conceptual.

Comando	Descrição	Feedback	Nota	Complexidade
add class <class>	Adiciona uma classe.	*Class <name> has been added successfully*.		O(1)
add attribute <attribute> to <class>	Adiciona um atributo de uma classe, sendo <class> o nome da classe e <attribute> o nome do atributo a ser adicionado na classe.	*Attribute <attribute> in class <class> has been added successfully*.		O(|CL|)
add attribute <attribute>	Adiciona um atributo de uma classe, sendo <attribute> o nome do atributo a ser adicionado na classe.	*Attribute <attribute> in class <class> has been added successfully*.	O parâmetro <class> é definido usando anteriormente os comandos de exemplo: - find <class> - add class <class> - change class name <class>	O(1)
add association (many | one | optional) to (many | one | optional) <class1> to <class2>	Adiciona uma associação entre duas classes, sendo <class1> o nome da primeira classe e <class2> o nome da segunda classe.	*Association between <class1> and <class2> has been added successfully*.		O(|CL|)

add association (many | one | optional) to (many | one | optional) <class2>	Adiciona uma associação entre duas classes, sendo <class2> o nome da segunda classe.	*Association between <class1> and <class2> has been added successfully.*	O parâmetro <class1> é definido usando anteriormente os comandos de exemplo: - find <class1> - add class <class1>	O(|CL|)
change class name <old> to <new>	Muda o nome de uma classe.	*Class <old> was changed successfully to <new>.*		O(|CL|)
change class name <new>	Muda o nome de uma classe.	*Class <old> was changed successfully to <new>.*	O parâmetro <old> é definido usando anteriormente os comandos de exemplo: - find <old> - add class <old>	O(1)
change attribute name <old> to <new>	Muda o nome de um atributo.	*Attribute <old> in class <class> was changed successfully to <new>.*	O parâmetro <class> é definido usando anteriormente os comandos de exemplo: - find <class>	O(|AT|)
change association (many | one | optional) to (many | one | optional) <class1> to <class2>	Muda a multiplicidade entre duas classes, sendo <class1> o nome da primeira classe e <class2> o nome da segunda classe.	*Multiplicity association between <class1> and <class2> has been changed successfully.*		O(|CL|+|AS|)
change association (many | one | optional) to (many | one | optional) <class2>	Muda a multiplicidade entre duas classes, sendo <class1> o nome da primeira classe e <class2> o nome da segunda classe.	*Multiplicity association between <class1> and <class2> has been changed successfully.*	O parâmetro <class1> é definido usando anteriormente os comandos de exemplo: - find <class1> - add association	O(|CL|+|AS|)
delete class <class>	Apaga uma classe	*Class <class> was deleted successfully.*		O(|CL|)

delete class	Apaga uma classe	*Class <class> was deleted successfully*.	O parâmetro <class> tem que ser especificado no comando anterior, por exemplo: - find <class> - add class <class> - change class name <class>	O(1)
delete attribute <attribute> to <class>	Apaga um atributo de uma classe, sendo <class> o nome da classe em questão e <attribute> o nome do atributo a ser apagado.	*Attribute <attribute> has been deleted successfully*.		O(|CL|+|AT|)
delete association <class1> to <class2>	Apaga uma associação entre duas classes, sendo <class1> o nome da primeira classe e <class2> o nome da segunda classe.	*Association between <class1> and <class2> has been deleted successfully*.		O(|CL|+|AS|)
delete association <class2>	Apaga uma associação entre duas classes, sendo <class2> o nome da segunda classe.	*Association between <class1> and <class2> has been deleted successfully*.	O parâmetro <class1> tem que ser especificado no comando anterior, por exemplo: - find <class1>	O(|CL|+|AS|)
find <class>	Foca a classe <class> de modo a que futuros comandos como add/-change/delete possam omitir parâmetros	*Class <class> found successfully*.		O(|CL|)
say number of classes	Indica o número total de classes que o modelo tem.	*Conceptual Model has <number> classes*.		O(|CL|)
say number of attributes <class>	Indica o número de atributos que uma classe tem.	*Class <class> has <number> attributes*.		O(|CL|+|AT|)

say number of attributes	Indica o número de atributos que uma classe tem.	*Class <class> has <number> attributes*.	O parâmetro <class> tem que ser especificado no comando anterior, por exemplo: - find <class> - change class name <class>	O(|AT|)
say classes name	Diz o nome de todas as classes do modelo.	*Conceptual Model has the following classes name: <names>*.		O(|CL|)
say attributes name <class>	Indica o nome de todas as classes do modelo.	*Conceptual Model has the following attribute name: <names>*.		O(|CL|+|AT|)
say attributes name	Indica o nome de todas as classes do modelo.	*Conceptual Model has the following attribute name: <names>*.	O parâmetro <class> tem que ser especificado no comando anterior, por exemplo: - find <class> - change class name <class>	O(|CL|+|AT|)
say associations <class>	Indica as associações que uma classe tem.	*Conceptual Model has the following associations name: <names>*.	O parâmetro <class> tem que ser especificado no comando anterior, por exemplo: - find <class>	O(|CL|+|AS|)
say associations	Indica as associações que uma classe tem.	*Conceptual Model has the following associations name: <names>*.	O parâmetro <class> tem que ser especificado no comando anterior, por exemplo: - find <class> - change class name <class>	O(|AS|)
say actual class	Notifica qual a classe actual que está focado.	*The actual goal is <goal>*.		O(1)
sleep mode on	Sistema fica em *stand-by*.	VoiceToModel *will be on stand-by*.		O(1)
sleep mode off	Sistema fica novamente activo para reconhecer comandos.	VoiceToModel *is ready*.		O(1)
repeat last feedback	Repete o *feedback* da última operação realizada.	Conteúdo da mensagem do último comando.		O(1)

validate model	Informa se o modelo é valido de acordo com as regras de especificação forma. Caso não seja, o utilizador é informado dos *warnings*/erros.	Conteúdo proveniente das retrições estabelecidas.		O(|CL|+|AS|)
undo command	Desfaz o último comando realizado.	Conteúdo proveniente da as retrições estabelecidas.		O(1)
describe commands	Descrever os comandos principais para o controlador de projectos.	Conteúdo da lista de comandos.		O(1)
help mode on	Permite activar o modo ajuda quando o utilizador não diz correctamente um comando.	*Help mode* ON.		O(1)
help mode off	Desactiva o modo ajuda.	*Help mode* OFF.		O(1)
back to project controller	Volta para o controlador de projectos.	*Back to project controller*.		O(1)

A.3 Comandos do Modelo de Features

Tabela A.3: Comandos do Model de *Features*.

Comando	Descrição	Feedback	Nota	Complexidade
add root <feature>	Adiciona uma *feature root*.	*Root <feature> has been added successfully*.		O(1)
add mandatory <feature2> to <feature1>	Adiciona uma *feature* obrigatória, sendo <feature1> o nome da *feature* pai e <feature2> o nome da *feature* a adicionar.	*Mandatory feature <feature2> has been added successfully*.		O(|F|+|G|)

add mandatory <feature2>	Adiciona uma *feature* obrigatória, sendo <feature2> o nome da *feature* a adicionar.	*Mandatory feature <feature2> has been added successfully.*	O parâmetro <feature1> é definido usando anteriormente os comandos de exemplo: - find <feature1> - find by position <number> - add root/mandatory/optional <feature1>	O(1)
add optional <feature2> to <feature1>	Adiciona uma *feature* opcional, sendo <feature1> o nome da *feature* pai e <feature2> o nome da *feature* a adicionar.	*Optional feature <feature2> has been added successfully.*		O(|F|+|G|)
add optional <feature2>	Adiciona uma *feature* opcional, sendo <feature2> o nome da *feature* a adicionar.	*Optional feature <feature2> has been added successfully.*	O parâmetro <feature1> é definido usando anteriormente os comandos de exemplo: - find <feature1> - find by position <number> - add root/mandatory/optional <feature1>	O(1)
add feature <feature2> to <feature1>	Adiciona uma *feature* a um grupo OR/*Alternative*, sendo <feature1> o nome da *feature* pai e <feature2> o nome da *feature* a adicionar.	*Feature <feature2> has been added successfully.*		O(|F|+|G|)

add feature <feature2>	Adiciona uma *feature* a um grupo OR/*Alternative*, sendo <feature2> o nome da *feature* a adicionar.	*Feature <feature2> has been added successfully*.	O parâmetro <feature1> é definido usando anteriormente os comandos de exemplo: - find <feature1> - find by position <number> - add group OR/*Alternative* <feature1>	O(1)
add group OR <feature>	Associa um grupo OR a uma *feature*, sendo <feature> o nome da *feature* a adicionar.	*Group OR of feature <feature> has been added successfully*.		O(|F|+|G|)
add group OR	Associa um grupo OR a uma *feature*.	*Group OR of feature <feature> has been added successfully*.	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add root/mandatory/optional/feature <feature>	O(|F|)
add group Alternative <feature>	Associa um grupo *Alternative* a uma *feature*, sendo <feature> o nome da *feature* a adicionar.	*Group Alternative of feature <feature> has been added successfully*.		O(|F|+|G|)

add group Alternative	Associa um grupo *Alternative* a uma *feature*.	*Group Alternative of feature <feature> has been added successfully*.	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add root/mandatory/optional/feature <feature>	O(|F|)
add constraint requires <feature1> to <feature2>	Adiciona uma restrição requires da <feature1> para a <feature2>.	*Constraint <feature1> requires <feature2> has been added successfully*.		O(|F|+|G|)
add constraint requires <feature2>	Adiciona uma restrição requires para a <feature2>.	*Constraint <feature1> requires <feature2> has been added successfully*.	O parâmetro <feature1> é definido usando anteriormente os comandos de exemplo: - find <feature1> - find by position <number> - add root/mandatory/optional/feature <feature1>	O(|F|+|G|)
add constraint excludes <feature1> to <feature2>	Adiciona uma restrição excludes da <feature1> para a <feature2>.	*Constraint <feature1> excludes <feature2> has been added successfully*.		O(|F|+|G|)
add constraint excludes <feature2>	Adiciona uma restrição excludes para a <feature2>.	*Constraint <feature1> excludes <feature2> has been added successfully*.	O parâmetro <feature1> é definido usando anteriormente os comandos de exemplo: - find <feature1> - find by position <number> - add root/mandatory/optional/feature <feature1>	O(|F|+|G|)
change name <old> to <new>	Muda o nome de uma *feature*.	*Feature <old> was changed successfully to <new>*.		O(|F|+|G|)

change name <new>	Muda o nome de uma *feature*.	*Feature <old> was changed successfully to <new>.*	O parâmetro <old> é definido usando anteriormente os comandos de exemplo: - find <old> - find by position <number> - add root/mandatory/optional/feature <old>	O(1)
change type of feature <feature>	Muda o tipo de uma *feature* para Obrigatória->Opcional ou Opcional->Obrigatória.	*Feature <feature> was changed to mandatory/optional.*		O(|F|+|G|+|C|)
change type of feature	Muda o tipo de uma *feature* para Obrigatória->Opcional ou Opcional->Obrigatória.	*Feature <feature> was changed to mandatory/optional.*	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add mandatory/optional <feature>	O(|F|+|G|+|C|)
change type of group <feature>	Muda o tipo de um grupo para OR->Alternative ou Alternative->OR.	*Group of feature <feature> was changed to Alternative/OR successfully.*		O(|F|+|G|)
change type of group	Muda o tipo de um grupo para OR->Alternative ou Alternative->OR.	*Group of feature <feature> was changed to Alternative/OR successfully.*	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add group Alternative/OR <feature>	O(|F|+|G|)

change type of constraint <feature1> to <feature2>	Muda o tipo de uma restrição para Requires->Excludes ou Excludes->Requires.	*Constraint was changed to requires/excludes successfully.*		O(|C|)
change type of constraint <feature2>	Muda o tipo de uma restrição para Requires->Excludes ou Excludes->Requires.	*Constraint was changed to requires/excludes successfully.*	O parâmetro <feature1> é definido usando anteriormente os comandos de exemplo: - find <feature1> - find by position <number> - add constraint requires/excludes <feature1> to <feature2>	O(|C|)
delete feature <feature>	Apaga uma *feature*.	*Feature <feature> was deleted successfully.*		O(|F|+|G|+|C|)
delete feature	Apaga uma *feature*.	*Feature <feature> was deleted successfully.*	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add root/mandatory/optional/feature <feature> - change type of feature	O(|F|+|G|+|C|)
delete group <feature>	Apaga um grupo OR/*Alternative*.	*Group was deleted successfully.*		O(|F|+|G|+|C|)

delete group	Apaga um grupo OR/*Alternative*.	*Group was deleted successfully*.	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add group OR/Alternative <feature> - change type of group	O(|F|+|G|+|C|)
delete constraint <feature1> to <feature2>	Apaga uma constraint entre a <feature1> e <feature2>.	*Constraint was deleted successfully*.		O(|C|)
delete constraint <feature2>	Apaga uma constraint entre a <feature1> e <feature2>.	*Constraint was deleted successfully*.	O parâmetro <feature1> é definido usando anteriormente os comandos de exemplo: - find <feature1> - find by position <number> - add constraint requires/excludes <feature> - change type of constraint	O(|C|)
find <feature>	Foca uma *feature* <feature> de modo a que futuros comandos como add/-change/delete possam omitir parâmetros	*Feature <feature> found successfully*.		O(|F|+|G|)

find by position <number>	Foca um *feature* <feature> dada a sua posição de modo a que futuros comandos como add/-change/delete possam omitir parâmetros	*Feature <feature> with the position <number> found successfully.*	Exemplo: *root* tem posição 1.	O(|F|+|G|)
say total of features	Indica o número total de *features* que o Modelo de *Features* tem.	*Feature Model has <features> features.*		O(|F|+|G|)
say total of mandatory features	Indica o número total de *features* obrigatórias que o Modelo de *Features* tem.	*Feature Model has <features> mandatory features.*		O(|F|+|G|)
say total of optional features	Indica o número total de *features* opcionais que o Modelo de *Features* tem.	*Feature Model has <features> optional features.*		O(|F|+|G|)
say total of constraint	Indica quantas constraints requires e excludes que o Modelo de *Features* tem.	*Feature Model has <number> requires and <number> excludes.*		O(|C|)
say number of sub features <feature>	Indica o número de filhos que uma *feature* tem.	*Feature <name> has <number> sub features.*		O(|F|+|G|)
say number of sub features	Indica o número de filhos que uma *feature* tem.	*Feature <name> has <number> sub features.*	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add/change	O(|F|+|G|)

say sub features name <feature>	Indica o nome dos filhos que uma *feature* tem.	*Feature <name> has (a group <group> with) the following features: <features>.*		O(|F|+|G|)
say sub features name	Indica o nome dos filhos que uma *feature* tem.	*Feature <name> has (a group <group> with) the following features: <features>.*	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add/change	O(|F|+|G|)
say constraints <feature>	Indica o tipo de restrições e para onde aponta.	*Feature <name> has constraint requires to Features <features> and constraint excludes to Features <features>.*		O(|C|)
say constraints	Indica o tipo de restrições e para onde aponta.	*Feature <name> has constraint requires to Features <features> and constraint excludes to Features <features>.*	O parâmetro <feature> é definido usando anteriormente os comandos de exemplo: - find <feature> - find by position <number> - add/change	O(|C|)
say actual feature	Notifica qual a *feature* actual que está focado.	*The actual feature is <feature>.*		O(1)
sleep mode on	Sistema fica em *stand-by*.	VoiceToModel *will be on stand-by*.		O(1)
sleep mode off	Sistema fica novamente activo para reconhecer comandos.	VoiceToModel *is ready*.		O(1)
repeat last feedback	Repete o *feedback* da última operação realizada.	Conteúdo da mensagem do último comando.		O(1)

validate model	Informa se o modelo é valido de acordo com as regras de especificação forma. Caso não seja, o utilizador é informado dos *warnings*/erros.	Conteúdo proveniente das retrições estabelecidas.		O(|F|+|G|+|C|)
undo command	Desfaz o último comando realizado.	Conteúdo proveniente da as retrições estabelecidas.		O(1)
describe commands	Descrever os comandos principais para o controlador de projectos.	Conteúdo da lista de comandos.		O(1)
help mode on	Permite activar o modo ajuda quando o utilizador não diz correctamente um comando.	*Help mode* ON.		O(1)
help mode off	Desactiva o modo ajuda.	*Help mode* OFF.		O(1)
back to project controller	Volta para o controlador de projectos.	*Back to project controller*.		O(1)

A.4 Comandos do Controlador de Projectos

Tabela A.4: Comandos do Controlador de Projectos.

Comando	Descrição	Feedback	Nota	Complexidade
start program	Inicia o sistema VoiceToModel.	*Start* VoiceToModel. *Workspace has* <number> *projects*.		O(1)
end program	Desliga o sistema VoiceToModel.	*End* VoiceToModel.		O(1)
create project	Cria um projecto em branco e associa um número ao projecto.	*Project number <number> has been created successfully*.	Exemplo: Project_1	O(1)

create feature model	Cria um modelo de *features* e associa um número ao modelo.	*Feature Model number <number> has been created successfully.*	Exemplo: FeatureModel_1 É necessário executar anteriormente o comando create/open project	O(1)
create conceptual model	Cria um modelo conceptual e associa um número ao modelo.	*Conceptual Model number <number> has been created successfully.*	Exemplo: ConceptualModel_1 É necessário executar anteriormente o comando create/open project	O(1)
create kaos model	Cria um modelo KAOS e associa um número ao modelo.	*KAOS Model number <number> has been created successfully.*	Exemplo: KAOSModel_1 É necessário executar anteriormente o comando create/open project	O(1)
open project <number>	Abre um projecto dado o seu número.	*Project number <number> is open.*		O(1)
open feature model <number>	Abre um modelo de *features* dado o seu número e possibilita ao utilizador iniciar a modelação.	*Feature Model number <number> is open.*	É necessário executar antes o comando open project.	O(1)
open conceptual model <number>	Abre um modelo conceptual dado o seu número e possibilita ao utilizador iniciar a modelação.	*Conceptual Model number <number> is open.*	É necessário executar antes o comando open project.	O(1)
open kaos model <number>	Abre um modelo KAOS dado o seu número e possibilita ao utilizador iniciar a modelação.	*KAOS Model number <number> is open.*	É necessário executar antes o comando open project.	O(1)
delete project <number>	Apaga um projecto dado o seu número	*Project number <number> has been deleted successfully.*		O(1)
delete feature model <number>	Apaga um modelo de *features* dado o seu número.	*Feature Model number <number> has been deleted successfully.*	É necessário executar antes o comando open project.	O(1)

delete conceptual model <number>	Apaga um modelo conceptual dado o seu número.	*Conceptual Model number <number> has been deleted successfully*.	É necessário executar antes o comando open project.	O(1)
delete kaos model <number>	Apaga um modelo KAOS dado o seu número.	*KAOS Model number <number> has been deleted successfully*.	É necessário executar antes o comando open project.	O(1)
say current model	Indica qual o projecto actual.	*The active project is number <number> with the <current_model> number <number>*.		O(1)
say project details	Informa quantos modelos tem o projecto e os seus respectivos identificadores.	*Project number <number> has <number> KAOS Models with the numbers <numbers>, <number> Conceptual Models with the numbers <numbers> and <number> Feature Models with the numbers <numbers>*.		O(1)
say number os projects	Indica qual o número total de projectos que há no *workspace*.	*Workspace has <number> projects*.		O(1)
sleep mode on	Sistema fica em *stand-by*.	VoiceToModel *will be on stand-by*.		O(1)
sleep mode off	Sistema fica novamente activo para reconhecer comandos.	VoiceToModel *is ready*.		O(1)
repeat last feedback	Repete o *feedback* da última operação realizada.	Conteúdo da mensagem do último comando.		O(1)
describe commands	Descrever os comandos principais para o controlador de projectos.	Conteúdo da lista de comandos.		O(1)
help mode on	Permite activar o modo ajuda quando o utilizador não diz correctamente um comando.	*Help mode* ON.		O(1)
help mode off	Desactiva o modo ajuda.	*Help mode* OFF.		O(1)

B

Anexo - Anotações

B.1 Emfatic Modelo SimpleKAOS

Listagem B.1: *Emfatic source* do Modelo SimpleKAOS.

```
@gmf(foo="bar")
@namespace(uri="http://simplekaosmodel.org/models/kaos/1.0",
prefix="simplekaosmodel")
package simplekaosmodel;

@gmf.diagram(foo="bar")
@v2m.grammar(name="simpleKAOS")
@v2m.say(aditionalName="total of goals")
@v2m.say(aditionalName="total of agents")
@v2m.sleepMode
@v2m.repeatLastFeedback
@v2m.validateModel
@v2m.undoCommand
@v2m.describeCommands
@v2m.helpMode
@v2m.backToProjectController
class KAOSModel {
  ref Goal[1] root;
  val Node[*] nodes;
  val Link[*] links;
}

abstract class Node {
  attr String name;
}
```

```
26
27 abstract class Object extends Node {
28 }
29
30 abstract class GoalRefinement {
31   @gmf.link(tool.name="AND/ORToGoal", tool.small.bundle="VoiceToModel",
32   tool.small.path="Icons/SimpleKAOS/and_or_to_goal.png")
33   ref Goal[*] goals;
34 }
35
36 @gmf.node(figure="polygon", polygon.x="0 10 80 70 0", polygon.y="35 0 0 35 35",
37 tool.small.bundle="VoiceToModel", tool.small.path="Icons/SimpleKAOS/goal.png",
38 label.icon="false", color="215,230,238", border.width="1", border.style="solid",
39 border.color="0,0,0", label="name", size="80,35", phantom="true")
40 @v2m.add(aditionalName="root", maxParameters="1", pointsTo="name")
41 @v2m.add(aditionalName="goal refinement OR", maxParameters="2",
42 pointsTo="name, name")
43 @v2m.add(aditionalName="goal refinement AND", maxParameters="2",
44 pointsTo="name, name")
45 @v2m.change(aditionalName="goal name", maxParameters="2",
46 pointsTo="name, name")
47 @v2m.change(aditionalName="refinement OR to AND", maxParameters="1",
48 pointsTo="name")
49 @v2m.change(aditionalName="refinement AND to OR", maxParameters="1",
50 pointsTo="name")
51 @v2m.delete(aditionalName="goal", maxParameters="1", pointsTo="name")
52 @v2m.delete(aditionalName="refinement OR", maxParameters="1", pointsTo="name")
53 @v2m.delete(aditionalName="refinement AND", maxParameters="1", pointsTo="name")
54 @v2m.find(maxParameters="1", pointsTo="name")
55 @v2m.findByPosition(maxParameters="1", pointsTo="position")
56 @v2m.say(aditionalName="number of sub goals", maxParameters="1",
57 pointsTo="name")
58 @v2m.say(aditionalName="number of agents", maxParameters="1", pointsTo="name")
59 @v2m.say(aditionalName="sub goals name", maxParameters="1", pointsTo="name")
60 @v2m.say(aditionalName="agents name", maxParameters="1", pointsTo="name")
61 @v2m.say(aditionalName="actual goal", maxParameters="1", pointsTo="name")
62 class Goal extends Node {
63   @gmf.link(tool.name="GoalToRefinement", source.decoration="arrow",
64   tool.small.bundle="VoiceToModel",
65   tool.small.path="Icons/SimpleKAOS/goal_to_refinement.png")
66   ref Link[*] links;
67 }
68
69 class Link {
70   attr String name;
71 }
72
73 @gmf.node(figure="polygon", polygon.x="0 10 70 80 70 10 0",
74 polygon.y="17 0 0 17 35 35 17", tool.small.bundle="VoiceToModel",
75 tool.small.path="Icons/SimpleKAOS/agent.png", label.icon="false",
```

```
color="255,255,227", border.width="1",
border.style="solid", border.color="0,0,0", label="name", size="80,35",
phantom="true")
@v2m.add(aditionalName="agent", maxParameters="2", pointsTo="name, Goal")
@v2m.change(aditionalName="agent name", maxParameters="2",
pointsTo="name, name")
@v2m.delete(aditionalName="agent", maxParameters="1", pointsTo="name")
class Agent extends Object {
}

@gmf.node(figure="ellipse", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/SimpleKAOS/OR.png", label="name", label.icon="false",
color="144,238,144", border.width="1", border.style="solid",
border.color="0,0,0", label.placement="external", phantom="true")
class OR extends Link, GoalRefinement {
}

@gmf.node(figure="ellipse", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/SimpleKAOS/AND.png", label="name", label.icon="false",
color="246,222,63", border.width="1", border.style="solid", border.color="0,0,0"
, label.placement="external", phantom="true")
class AND extends Link, GoalRefinement {
}

@gmf.node(figure="ellipse", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/SimpleKAOS/AgentLink.png", label="name",
label.icon="false", color="204,0,0", border.width="1", border.style="solid",
border.color="0,0,0", label.placement="external", phantom="true")
class AgentLink extends Link {
  @gmf.link(tool.name="AgentRefToAgent", tool.small.bundle="VoiceToModel",
  tool.small.path="Icons/SimpleKAOS/agentref_to_agent.png")
  ref Agent[1] targetAgent;
}
```

B.2 Emfatic Modelo Conceptual

Listagem B.2: *Emfatic source* do Modelo Conceptual.

```
@gmf(foo="bar")
@namespace(uri="http://conceptualmodel.org/models/conceptual/1.0",
prefix="conceptualmodel")
package conceptualmodel;

@gmf.diagram(foo="bar")
@v2m.grammar(name="conceptualModel")
@v2m.say(aditionalName="number of classes")
@v2m.say(aditionalName="classes name")
@v2m.sleepMode
@v2m.repeatLastFeedback
```

```
@v2m.validateModel
@v2m.undoCommand
@v2m.describeCommands
@v2m.helpMode
@v2m.backToProjectController
class ConceptualModel {
  val Classifier[*] classifiers;
}

abstract class Classifier {
}

@gmf.node(label="name", figure="rectangle", tool.name="Class",
tool.small.bundle="VoiceToModel",
tool.small.path="Icons/ConceptualModel/class.png")
@v2m.add(aditionalName="class", maxParameters="1", pointsTo="name")
@v2m.change(aditionalName="class name", maxParameters="2",
pointsTo="name, name")
@v2m.delete(aditionalName="class", maxParameters="1", pointsTo="name")
@v2m.find(maxParameters="1", pointsTo="name")
@v2m.say(aditionalName="number of attributes", maxParameters="1",
pointsTo="name")
@v2m.say(aditionalName="attributes name", maxParameters="1", pointsTo="name")
@v2m.say(aditionalName="associations", maxParameters="1", pointsTo="name")
@v2m.say(aditionalName="actual class")
class Class extends Classifier {

  @gmf.compartment(layout="list", collapsible="false")
  val Attribute[*] attributes;

  @gmf.affixed
  val Property[*] properties;
  attr String name;
}

@gmf.node(label="name", figure="rectangle", label.icon="false",
tool.name="Attribute", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/ConceptualModel/attribute.png")
@v2m.add(aditionalName="attribute", maxParameters="2", pointsTo="name, Class")
@v2m.change(aditionalName="attribute name", maxParameters="2",
pointsTo="name, name")
@v2m.delete(aditionalName="attribute", maxParameters="2",
pointsTo="name, Class")
class Attribute {
  attr String name;
}

abstract class MultiplicityElement {
  attr int lower;
  attr int upper;
```

```
}

@gmf.link(label="name", source="source", target="target",
tool.name="Association", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/ConceptualModel/association.png")
@v2m.add(aditionalName="association [many | one | optional] to [many | one |"
"optional]", maxParameters="2", pointsTo="source, target")
@v2m.change(aditionalName="association multiplicity [many | one | optional] to"
"[many | one | optional]", maxParameters="2", pointsTo="source, target")
@v2m.delete(aditionalName="association", maxParameters="2",
pointsTo="source, target")
class Association extends RelationShip, Classifier {
  attr String name;
  ref Property[1] source;
  ref Property[1] target;
}

@gmf.node(figure="rectangle", size="10,10", label="lower,upper",
label.placement="external", label.icon="false", label.pattern="{0}..{1}",
tool.name="Multiplicity", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/ConceptualModel/multiplicity.png")
class Property extends MultiplicityElement {
  ref Association association;
  attr AggregationKind aggregation;
}

abstract class RelationShip {
}

enum AggregationKind {
  none = 0;
  share = 1;
  aggregation = 2;
}
```

B.3 Emfatic Modelo de Features

Listagem B.3: *Emfatic source* do Modelo de *Features*.

```
@gmf(foo="bar")
@namespace(uri="http://featuremodel.org/models/feature/1.0",
prefix="featuremodel")
package featuremodel;

@gmf.diagram(foo="bar")
@v2m.grammar(name="featureModel")
@v2m.say(aditionalName="total of features")
@v2m.say(aditionalName="total of mandatory feature")
@v2m.say(aditionalName="total of optional feature")
```

```
@v2m.say(aditionalName="total of constraints")
@v2m.sleepMode
@v2m.repeatLastFeedback
@v2m.validateModel
@v2m.undoCommand
@v2m.describeCommands
@v2m.helpMode
@v2m.backToProjectController
class FeatureModel {
  val Root[1] rootNode;
}

@v2m.add(aditionalName="group OR", maxParameters="1", pointsTo="name")
@v2m.add(aditionalName="group Alternative", maxParameters="1", pointsTo="name")
@v2m.change(aditionalName="name", maxParameters="2", pointsTo="name, name")
@v2m.change(aditionalName="group", maxParameters="1", pointsTo="name")
@v2m.delete(aditionalName="feature", maxParameters="1", pointsTo="name")
@v2m.delete(aditionalName="group", maxParameters="1", pointsTo="name")
@v2m.find(maxParameters="1", pointsTo="name")
@v2m.findByPosition(maxParameters="1", pointsTo="position")
@v2m.say(aditionalName="number of sub features", maxParameters="1",
pointsTo="name")
@v2m.say(aditionalName="sub features name", maxParameters="1", pointsTo="name")
@v2m.say(aditionalName="constraints", maxParameters="1", pointsTo="name")
@v2m.say(aditionalName="actual feature", maxParameters="1", pointsTo="name")
abstract class FeatureNode {
  attr String name;

  @gmf.link(tool.name="LinkToMand/Opt", tool.small.bundle="VoiceToModel",
  tool.small.path="Icons/FeatureModel/member.png")
  val Member[*] members;
  val Constraint[*] constraints;
}

@gmf.node(label="name", figure="rectangle", label.icon="false",
border.width="2", border.style="solid", border.color="0,0,0",
label.placement="internal", size="80,35", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/FeatureModel/root.png")
@v2m.add(aditionalName="root", maxParameters="1", pointsTo="name")
class Root extends FeatureNode {
}

@v2m.change(aditionalName="feature", maxParameters="1", pointsTo="name")
class Solitary extends FeatureNode, Member {
  attr int[1] lower;
  attr int[1] upper;
}


@gmf.node(figure="polygon",
```

```
polygon.x="0 80 80 0 0 35 35 45 44 35 35 44 44 35 35 44 44 35 35 44 44 35 35
44 44 35 35 44", polygon.y="10 10 45 45 10 10 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8
8 9 9 10 10", label.icon="false", border.width="2", border.style="solid",
border.color="0,0,0", label="name", size="80,35", phantom="true",
tool.small.bundle="VoiceToModel",
tool.small.path="Icons/FeatureModel/mandatory.png")
@v2m.add(aditionalName="mandatory", maxParameters="2", pointsTo="name, name")
class Mandatory extends Solitary {
}

@gmf.node(figure="polygon", polygon.x="0 80 80 0 0 35 35 45 45 80",
polygon.y="10 10 45 45 10 10 0 0 10 10", label.icon="false", border.width="2",
border.style="solid", border.color="0,0,0", label="name", size="80,35",
phantom="true", tool.small.bundle="VoiceToModel",
tool.small.path="Icons/FeatureModel/optional.png")
@v2m.add(aditionalName="optional", maxParameters="2", pointsTo="name, name")
class Optional extends Solitary {
}

@gmf.node(label="name", figure="rectangle", label.icon="false",
border.width="2", border.style="solid", border.color="0,0,0",
label.placement="internal", size="80,35", phantom="true",
tool.small.bundle="VoiceToModel",
tool.small.path="Icons/FeatureModel/grouped.png")
@v2m.add(aditionalName="feature", maxParameters="2", pointsTo="name, name")
class Grouped extends FeatureNode, GroupMember {
}

abstract class Member {
}

class Group extends Member {
  attr int[1] lower;
  attr int[1] upper;
  @gmf.link(tool.name="LinkToGrouped", tool.small.bundle="VoiceToModel",
  tool.small.path="Icons/FeatureModel/groupMember.png")
  val GroupMember[*] groupMembers;
}

@gmf.node(label="lower", label.text="", figure="rectangle", label.icon="false",
color="0,0,0",label.placement="external",size="40,8", phantom="true",
tool.small.bundle="VoiceToModel", tool.small.path="Icons/FeatureModel/or.png")
class OR extends Group {
}

@gmf.node(label="lower", label.text="", figure="rectangle", label.icon="false",
border.width="1", border.style="solid", border.color="0,0,0",
label.placement="external",size="40,8", phantom="true",
tool.small.bundle="VoiceToModel",
tool.small.path="Icons/FeatureModel/alternative.png")
```

```
111 class Alternative extends Group {
112 }
113
114 @v2m.change(aditionalName="constraint", maxParameters="2",
115 pointsTo="source, target")
116 @v2m.delete(aditionalName="constraint", maxParameters="2",
117 pointsTo="source, target")
118 class Constraint {
119   attr String name;
120   ref FeatureNode[1] source;
121   ref FeatureNode[1] target;
122 }
123
124 @gmf.link(label="name", label.text="Requires", source="source", target="target",
125 style="dot", width="2", target.decoration="arrow",
126 tool.small.bundle="VoiceToModel",
127 tool.small.path="Icons/FeatureModel/requires.png")
128 @v2m.add(aditionalName="contraint requires", maxParameters="2",
129 pointsTo="source, target")
130 class Requires extends Constraint {
131 }
132
133 @gmf.link(label="name", label.text="Excludes", source="source", target="target",
134 style="dot", width="2", target.decoration="arrow",
135 tool.small.bundle="VoiceToModel",
136 tool.small.path="Icons/FeatureModel/excludes.png")
137 @v2m.add(aditionalName="contraint excludes", maxParameters="2",
138 pointsTo="source, target")
139 class Excludes extends Constraint {
140 }
141
142 abstract class GroupMember {
143 }
144
145 class Reference extends Member, GroupMember {
146   ref Root[1] parentNode;
147 }
```

Anexo - Questionário

Dados pessoais

Esta secção vai permitir saber um pouco sobre si.

Nome

Idade

Género

◯ Masculino
◯ Feminino

Background

Indique qual a sua área de estudo.

Qual o nível do seu inglês?

Acessibilidade

Esta secção vai permitir comentar e classificar aspectos de acessibilidade para gerar modelos de requisitos.

Tarefa 1 - Tempo despendido no Controlador de Projectos e na criação do Modelo KAOS.
_____ minuto(s).

Tarefa 2 - Tempo despendido no Controlador de Projectos e na criação do Modelo Conceptual.
_____ minuto(s).

Tarefa 3 - Tempo despendido no Controlador de Projectos e na criação do Modelo de *Features*.
_____ minuto(s).

Tarefa 4 - Tempo despendido no Controlador de Projectos e na modificação e consulta do Modelo KAOS.
_____ minuto(s).

Tarefa 5 - Tempo despendido no Controlador de Projectos e na modificação e consulta do Modelo Conceptual.
_____ minuto(s).

Tarefa 6 - Tempo despendido no Controlador de Projectos e na modificação e consulta do Modelo de *Features*.
_____ minuto(s).

Qual a sua impressão ao usar a ferramenta?
◯ Muito Má
◯ Má
◯ Suficiente
◯ Boa
◯ Muito Boa

Foi difícil usar o VoiceToModel para o exemplo apresentado na avaliação?
◯ Sim
◯ Não

Os modelos gerados foram os esperados?
◯ Sim
◯ Não

Se na pergunta anterior a resposta seleccionada foi "Não", indique qual o modelo em causa e a(s) dificuldade(s) enfrentadas.

Modelo de interacção

Permite obter *feedback* sobre o modelo de interacção implementado.

Como qualifica os comandos implementados que permitem criar/editar/apagar projectos ou ficheiros de modelos?

◯ Muito Má
◯ Má
◯ Suficiente
◯ Boa
◯ Muito Boa

Que alterações faria?

Como qualifica os comandos implementados que permitem gerar um Modelo KAOS?

◯ Muito Má
◯ Má
◯ Suficiente
◯ Boa
◯ Muito Boa

Que alterações faria?

Como qualifica os comandos implementados que permitem gerar um Modelo Conceptual?

◯ Muito Má
◯ Má
◯ Suficiente
◯ Boa
◯ Muito Boa

Que alterações faria?

Como qualifica os comandos implementados que permitem gerar um Modelo de *Features*?

◯ Muito Má
◯ Má
◯ Suficiente
◯ Boa
◯ Muito Boa

Que alterações faria?

Como qualifica a qualidade do *feedback* apresentado?

◯ Muito Má
◯ Má
◯ Suficiente
◯ Boa
◯ Muito Boa

Que alterações faria no *feedback* apresentado para cada operação?

Aspectos gerais do VoiceToModel

Esta secção vai permitir obter *feedback* sobre os aspectos gerais da ferramenta.

Já alguma vez trabalhou com alguma aplicação de reconhecimento de voz?

◯ Sim
◯ Não

Se seleccionou a resposta "Sim"na pergunta anterior, indique por favor quais são as ferramentas e o que achou da qualidade de accuracy e do modelo de interacção da(s) ferramenta(s) comparativamente ao VoiceToModel.

Já alguma vez trabalhou com alguma ferramenta de modelação com foco na acessibilidade?

◯ Sim
◯ Não

Se respondeu "Sim"a pergunta anterior, indique qual o mecanismo de acessibilidade usado.

Na sua opinião quais são os pontos fortes da ferramenta VoiceToModel?

E os fracos?

Que alterações faria para o VoiceToModel?

Acha que as pessoas com limitações físicas ou cegas irão sentir-se capazes de usar a ferramenta no futuro (mudando ou não a API de reconhecimento de voz) e gerar modelos de acordo com a especificação pedida?

◯ Sim
◯ Não

Se respondeu não na questão anterior, indique por favor, as razões que o conduziram para essa opinião.

Outros comentários/observações que queira partilhar.

D

Anexo dos comandos em braile

D.1 Tarefas

.

⠝⠾⠍⠑⠗⠕ ⠥⠍

⠨⠞⠁⠗⠑⠋⠁ ⠝⠾⠍⠑⠗⠕ ⠥⠍

⠅⠉⠕⠝⠎⠊⠙⠑⠗⠑ ⠥⠍ ⠎⠊⠎⠞⠑⠍⠁ ⠙⠑
⠑⠍⠁⠊⠇ ⠟⠥⠑ ⠏⠑⠗⠍⠊⠞⠑ ⠑⠝⠧⠊⠁⠗⠂
⠗⠑⠉⠑⠃⠑⠗ ⠑ ⠝⠕⠞⠊⠋⠊⠉⠁⠗ ⠕ ⠥⠞⠊⠤
⠇⠊⠵⠁⠙⠕⠗ ⠎⠕⠃⠗⠑ ⠁ ⠗⠑⠉⠑⠏⠯⠜⠕ ⠙⠕
⠑⠍⠁⠊⠇⠄

⠨⠕ ⠑⠍⠁⠊⠇ ⠎⠬ ⠏⠕⠙⠑ ⠎⠑⠗ ⠋⠑⠊⠞⠕
⠉⠕⠍ ⠕ ⠥⠎⠕ ⠙⠑ ⠞⠑⠭⠞⠕ ⠕⠥ ⠊⠍⠁⠤
⠛⠑⠍⠄ ⠨⠁ ⠗⠑⠉⠑⠏⠯⠜⠕ ⠙⠕⠎ ⠑⠍⠁⠊⠇⠎
⠎⠬ ⠏⠕⠙⠑ ⠎⠑⠗ ⠋⠑⠊⠞⠁ ⠑⠍ ⠉⠁⠎⠁ ⠕⠥
⠝⠕ ⠞⠗⠁⠃⠁⠇⠓⠕⠄

⠨⠍⠕⠙⠑⠇⠕ ⠑⠎⠞⠑ ⠎⠊⠎⠞⠑⠍⠁ ⠥⠎⠁⠝⠤
⠙⠕ ⠕ ⠍⠕⠙⠑⠇⠕ ⠨⠨⠅⠁⠕⠎⠄

⠨⠞⠁⠗⠑⠋⠁ ⠝⠾⠍⠑⠗⠕ ⠙⠕⠊⠎

⠨⠉⠕⠝⠎⠊⠙⠑⠗⠁ ⠕ ⠑⠭⠑⠍⠏⠇⠕ ⠙⠁
⠞⠁⠗⠑⠋⠁ ⠥⠍⠄ ⠨⠍⠕⠙⠑⠇⠕ ⠑⠎⠞⠑ ⠎⠊⠎⠤
⠞⠑⠍⠁ ⠥⠎⠁⠝⠙⠕ ⠕ ⠍⠕⠙⠑⠇⠕ ⠉⠕⠝⠤
⠉⠑⠏⠞⠥⠁⠇⠄

⠃

⠨⠞⠁⠗⠑⠋⠁ ⠝⠾⠍⠑⠗⠕ ⠞⠗⠣⠎

⠨⠉⠕⠝⠎⠊⠙⠑⠗⠁ ⠕ ⠑⠭⠑⠍⠏⠇⠕ ⠙⠁
⠞⠁⠗⠑⠋⠁ ⠥⠍⠄ ⠨⠍⠕⠙⠑⠇⠕ ⠑⠎⠞⠑ ⠎⠊⠎⠤
⠞⠑⠍⠁ ⠥⠎⠁⠝⠙⠕ ⠕ ⠍⠕⠙⠑⠇⠕ ⠙⠑ ⠋⠑⠁⠤
⠞⠥⠗⠑⠎⠄

⠨⠞⠁⠗⠑⠋⠁ ⠝⠾⠍⠑⠗⠕ ⠟⠥⠁⠞⠗⠕

⠨⠉⠕⠝⠎⠊⠙⠑⠗⠁ ⠕ ⠍⠕⠙⠑⠇⠕ ⠛⠑⠗⠁⠙⠕
⠝⠁ ⠞⠁⠗⠑⠋⠁ ⠥⠍⠄ ⠨⠏⠑⠗⠛⠥⠝⠞⠑ ⠁⠕
⠎⠊⠎⠞⠑⠍⠁ ⠟⠥⠁⠝⠞⠕⠎ ⠛⠕⠁⠇⠎ ⠓⠷ ⠝⠕
⠍⠕⠙⠑⠇⠕ ⠑ ⠕ ⠝⠕⠍⠑ ⠙⠕⠎ ⠎⠥⠃
⠛⠕⠁⠇⠎ ⠙⠕ ⠛⠕⠁⠇ ⠑⠍⠁⠊⠇⠄

⠨⠞⠁⠗⠑⠋⠁ ⠝⠾⠍⠑⠗⠕ ⠉⠊⠝⠉⠕

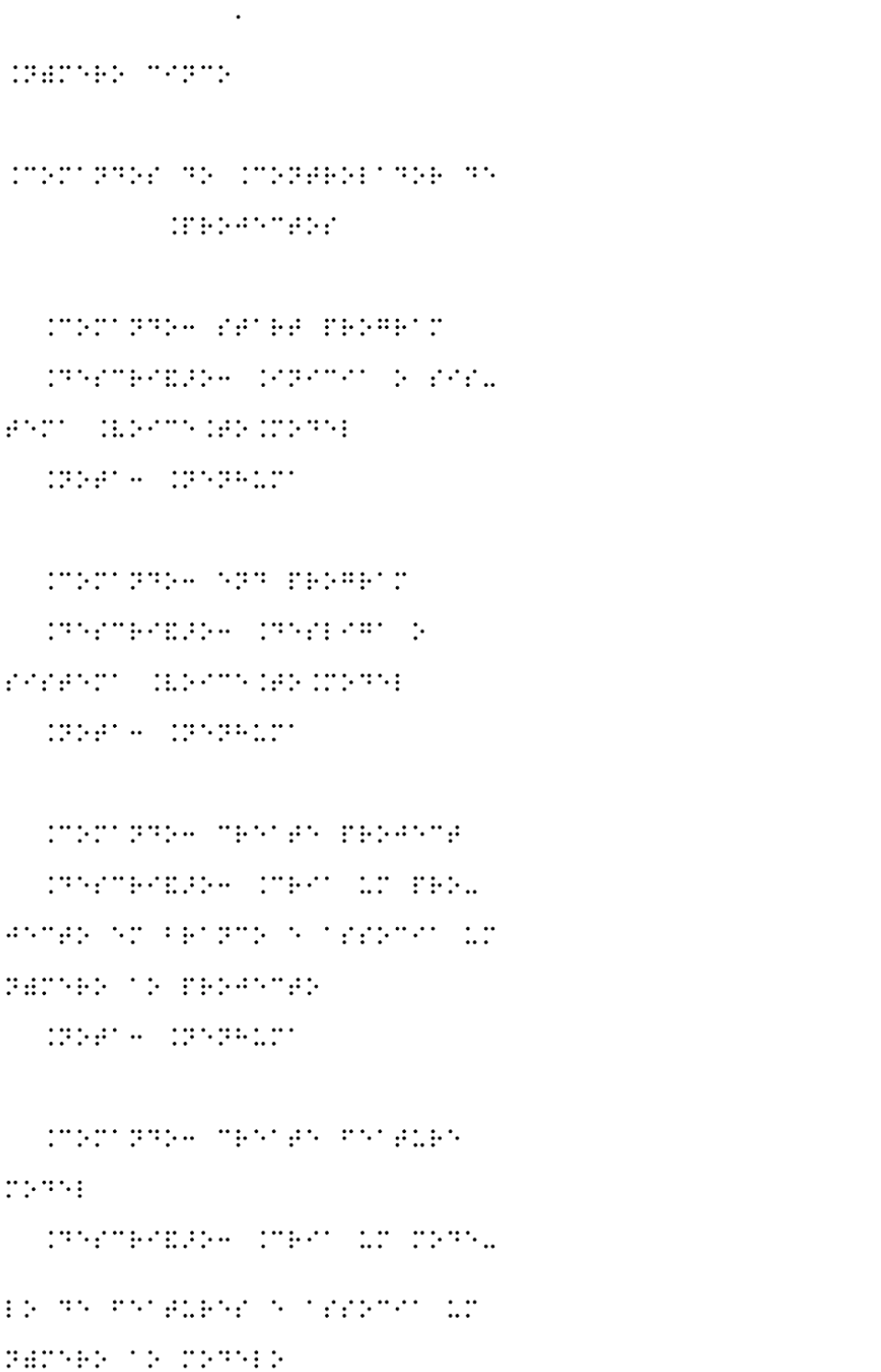
⠨⠉⠕⠝⠎⠊⠙⠑⠗⠁ ⠕ ⠍⠕⠙⠑⠇⠕ ⠛⠑⠗⠁⠙⠕
⠝⠁ ⠞⠁⠗⠑⠋⠁ ⠙⠕⠊⠎⠄ ⠨⠏⠑⠗⠛⠥⠝⠞⠑ ⠁⠕
⠎⠊⠎⠞⠑⠍⠁ ⠟⠥⠁⠝⠞⠁⠎ ⠉⠇⠁⠎⠎⠑⠎ ⠓⠷
⠝⠕ ⠍⠕⠙⠑⠇⠕⠄ ⠨⠎⠕⠇⠊⠉⠊⠞⠑ ⠁⠕ ⠏⠗⠕⠤
⠛⠗⠁⠍⠁ ⠕ ⠝⠕⠍⠑ ⠙⠑ ⠞⠕⠙⠁⠎ ⠁⠎
⠉⠇⠁⠎⠎⠑⠎ ⠟⠥⠑ ⠓⠷ ⠝⠕ ⠍⠕⠙⠑⠇⠕⠄

⠒

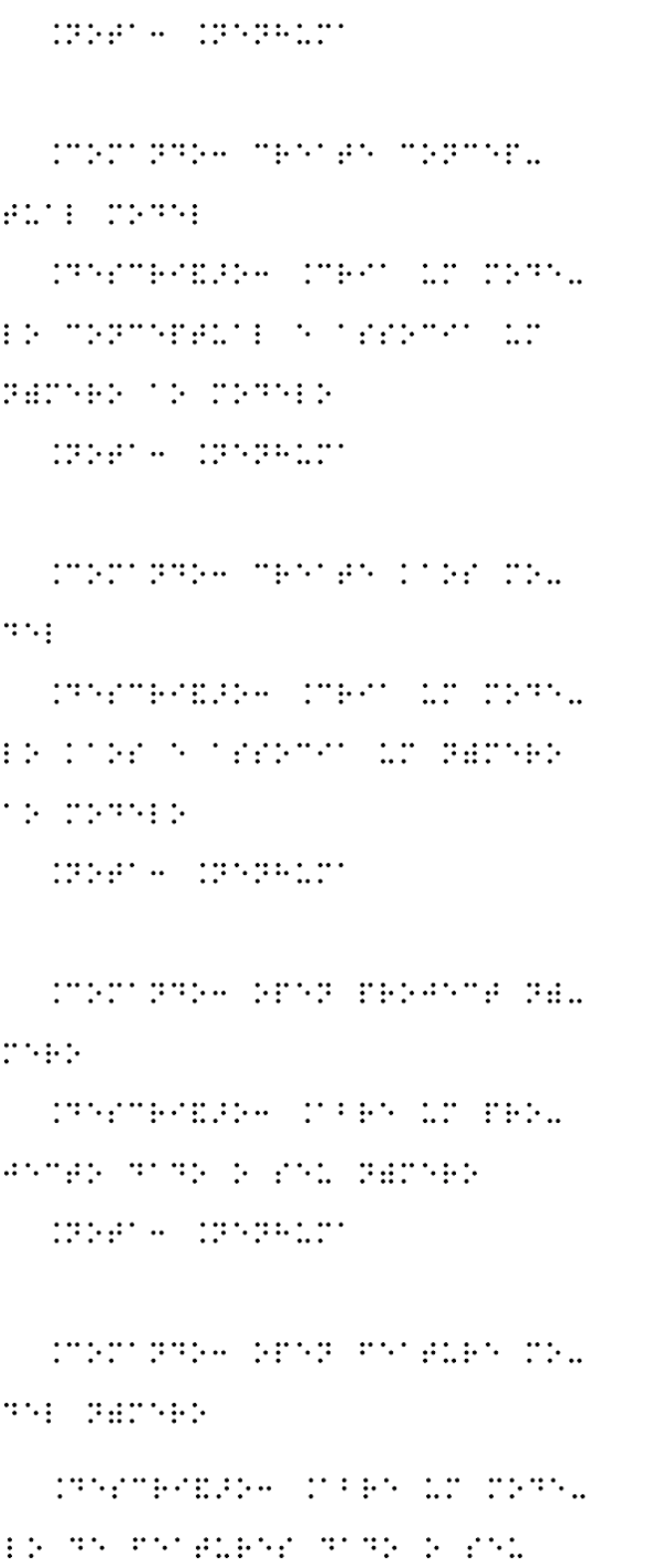
⠨⠞⠁⠗⠑⠋⠁ ⠝⠾⠍⠑⠗⠕ ⠎⠑⠊⠎

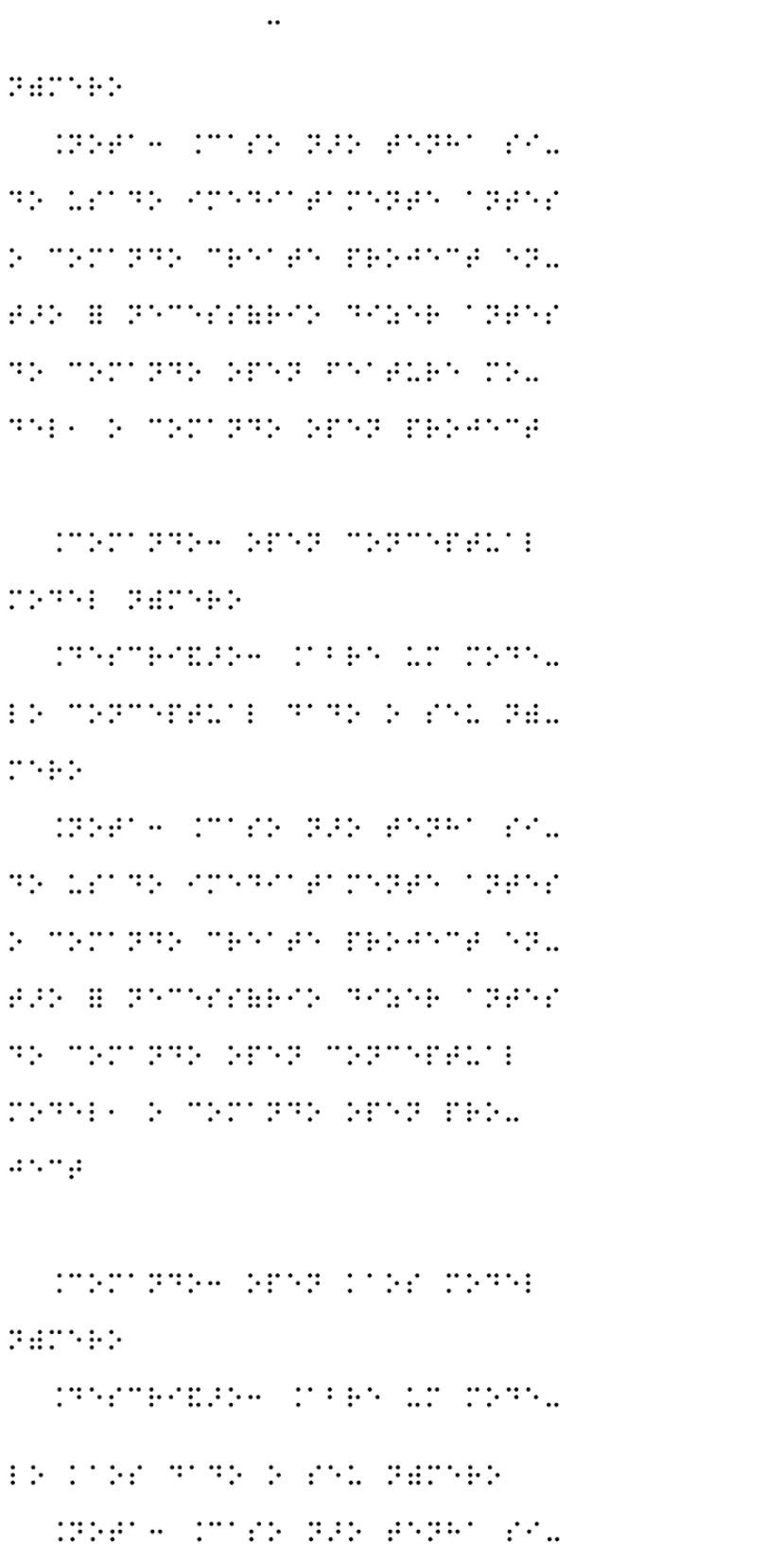
⠨⠉⠕⠝⠎⠊⠙⠑⠗⠁ ⠕ ⠍⠕⠙⠑⠇⠕ ⠛⠑⠗⠁⠙⠕
⠝⠁ ⠞⠁⠗⠑⠋⠁ ⠞⠗⠷⠎⠄ ⠨⠏⠑⠗⠛⠥⠝⠞⠑ ⠁⠕
⠎⠊⠎⠞⠑⠍⠁ ⠕ ⠝⠾⠍⠑⠗⠕ ⠞⠕⠞⠁⠇ ⠙⠑
⠋⠑⠁⠞⠥⠗⠑⠎⠂ ⠋⠑⠁⠞⠥⠗⠑⠎ ⠕⠃⠗⠊⠛⠁⠞⠬⠤
⠗⠊⠁⠎ ⠑ ⠋⠑⠁⠞⠥⠗⠑⠎ ⠕⠏⠉⠊⠕⠝⠁⠊⠎
⠟⠥⠑ ⠓⠷ ⠝⠕ ⠍⠕⠙⠑⠇⠕⠄

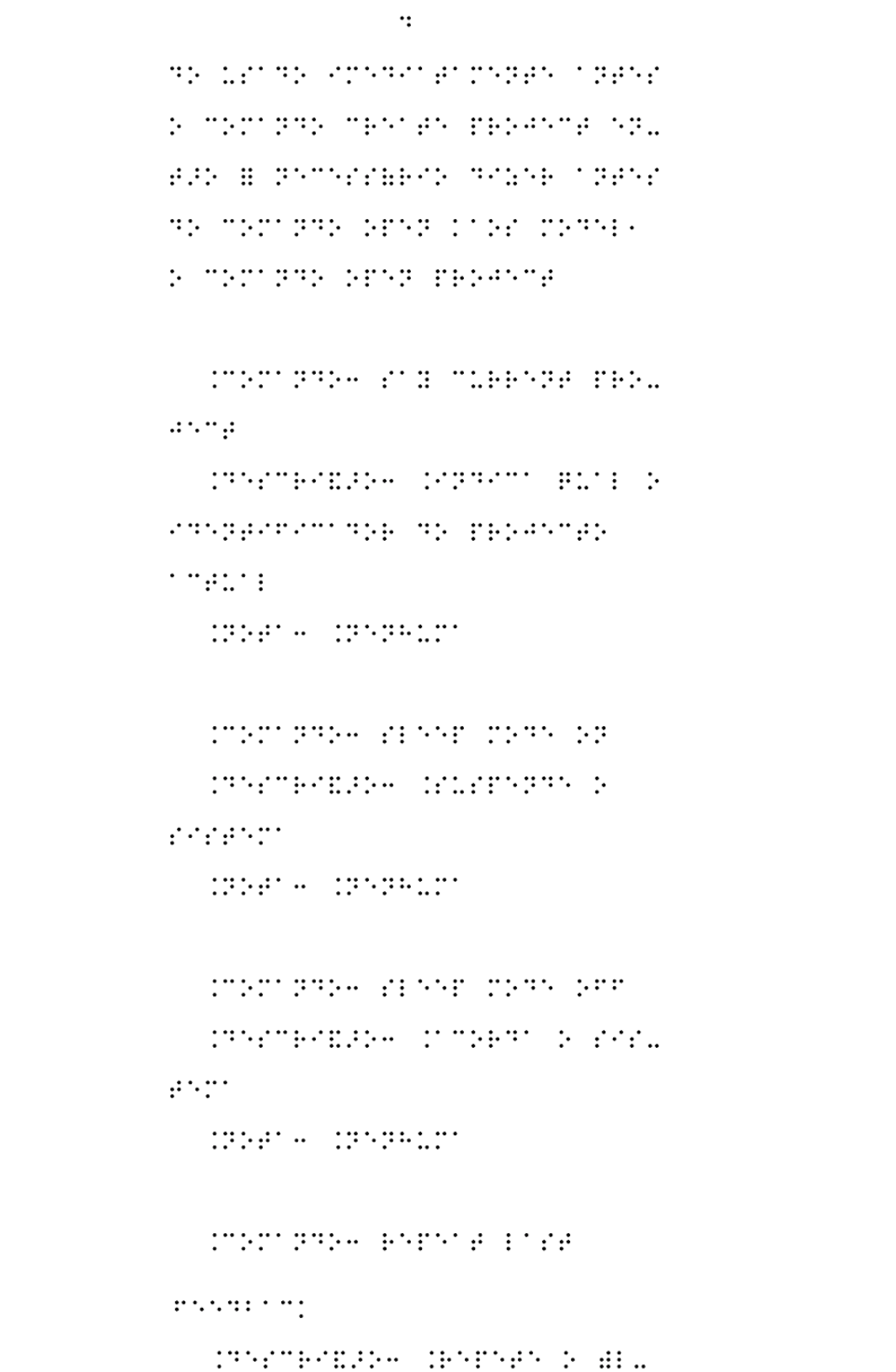
D.2 Comandos do Controlador de Projectos

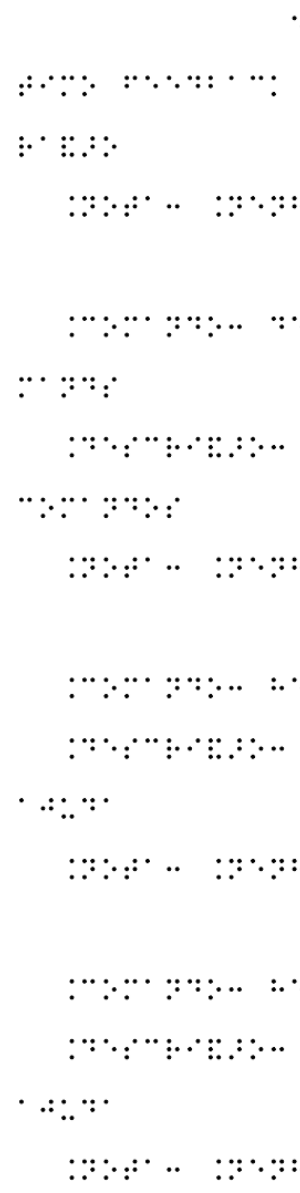


:

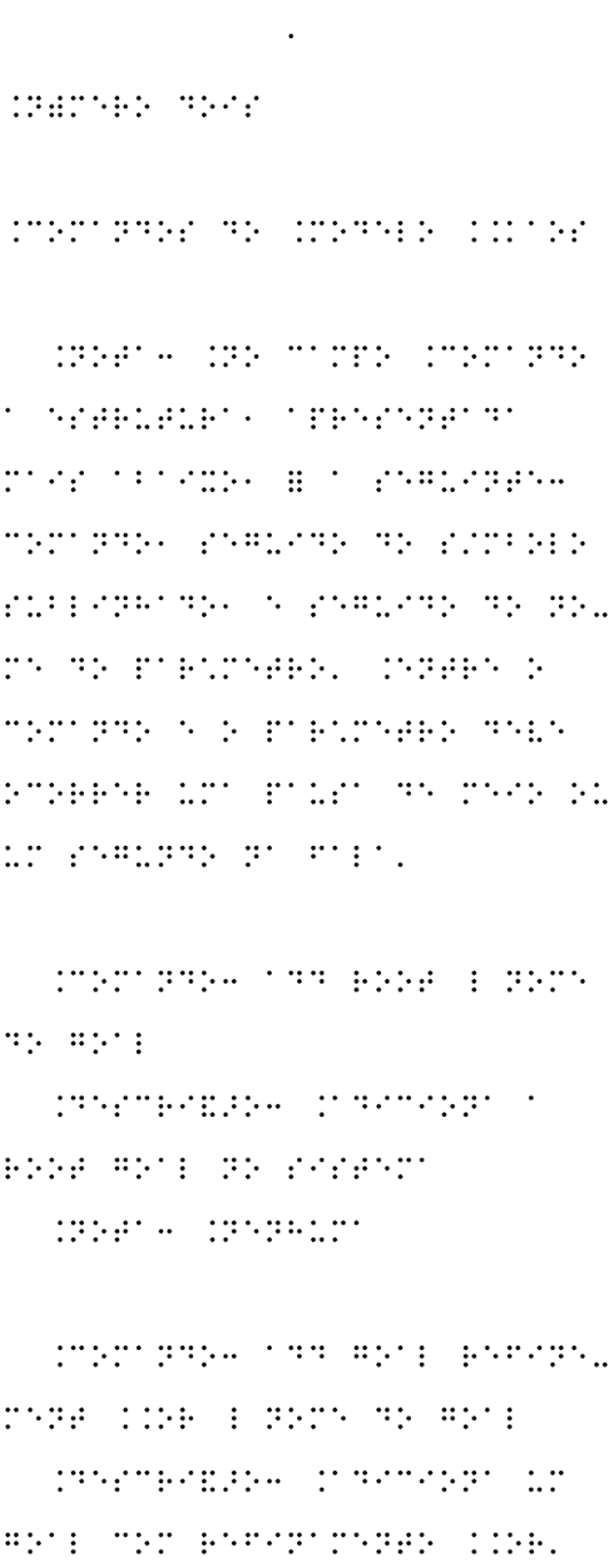


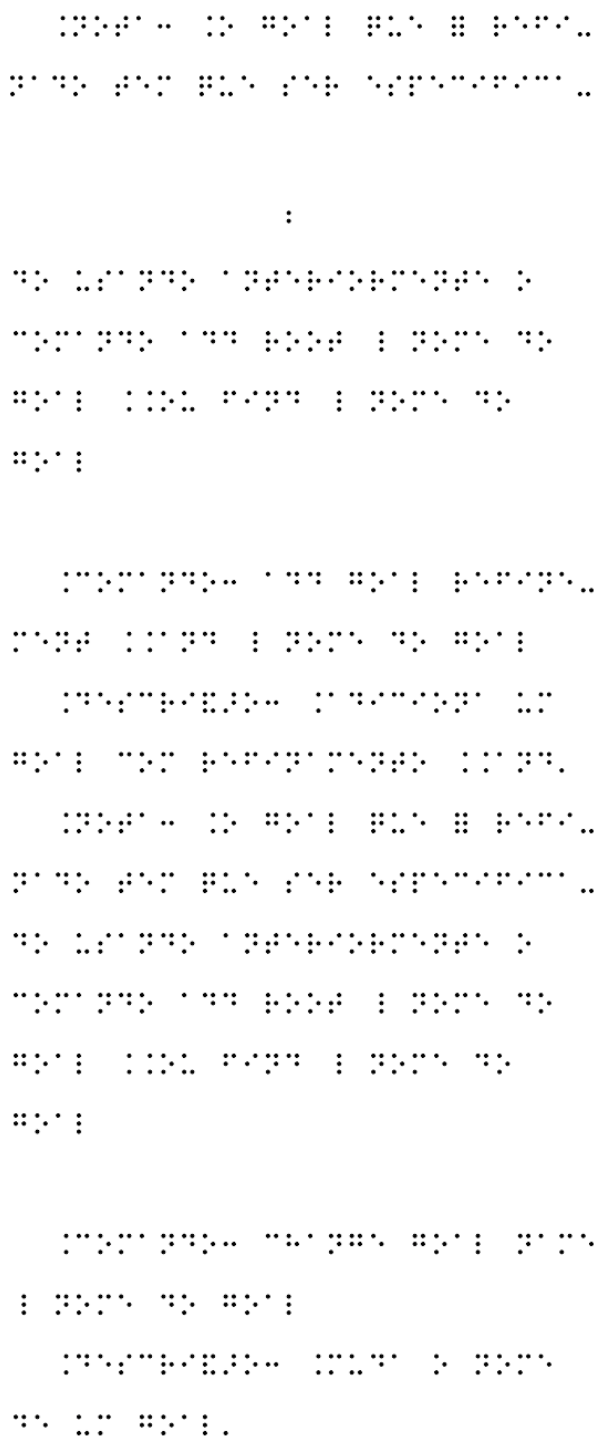


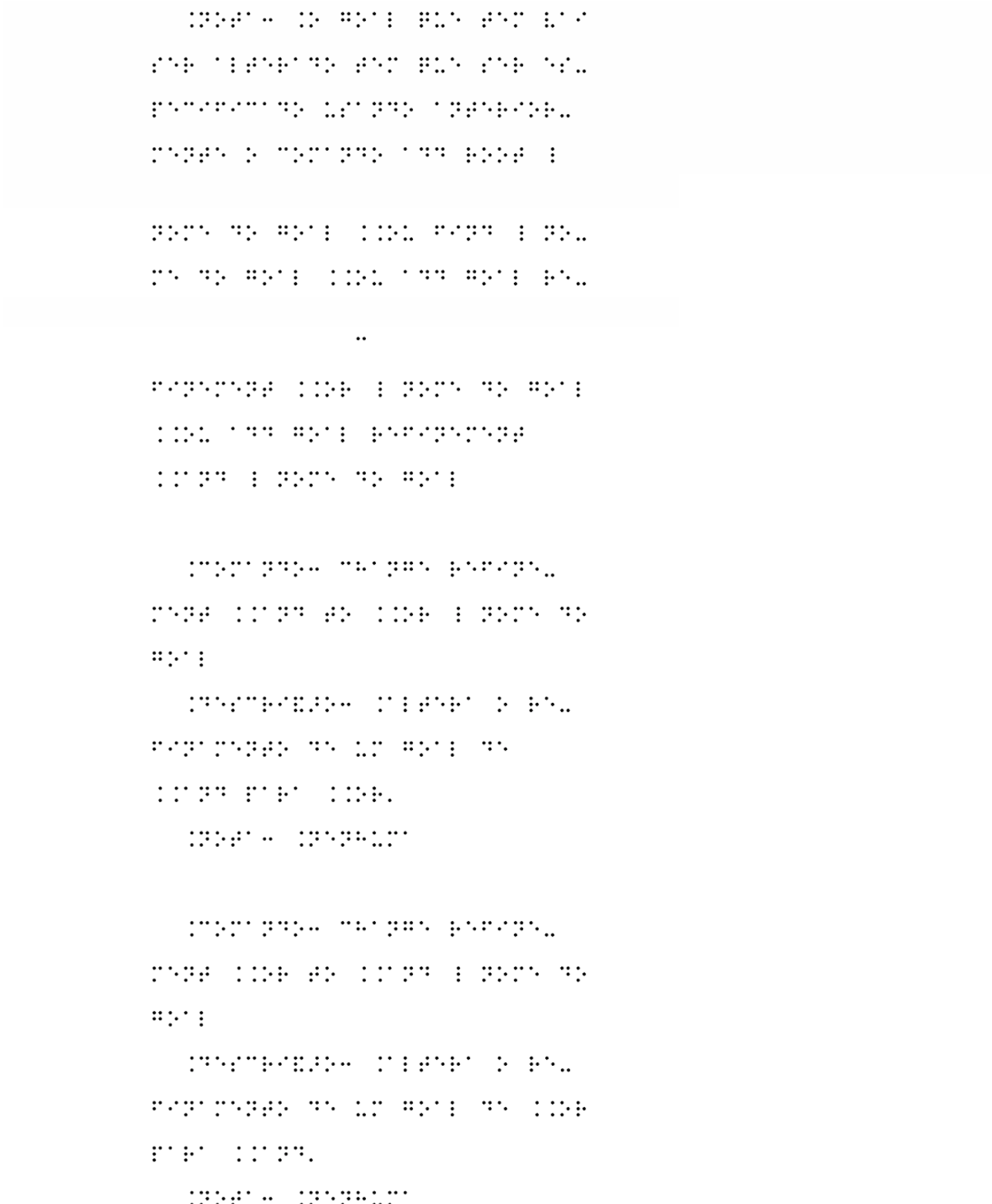


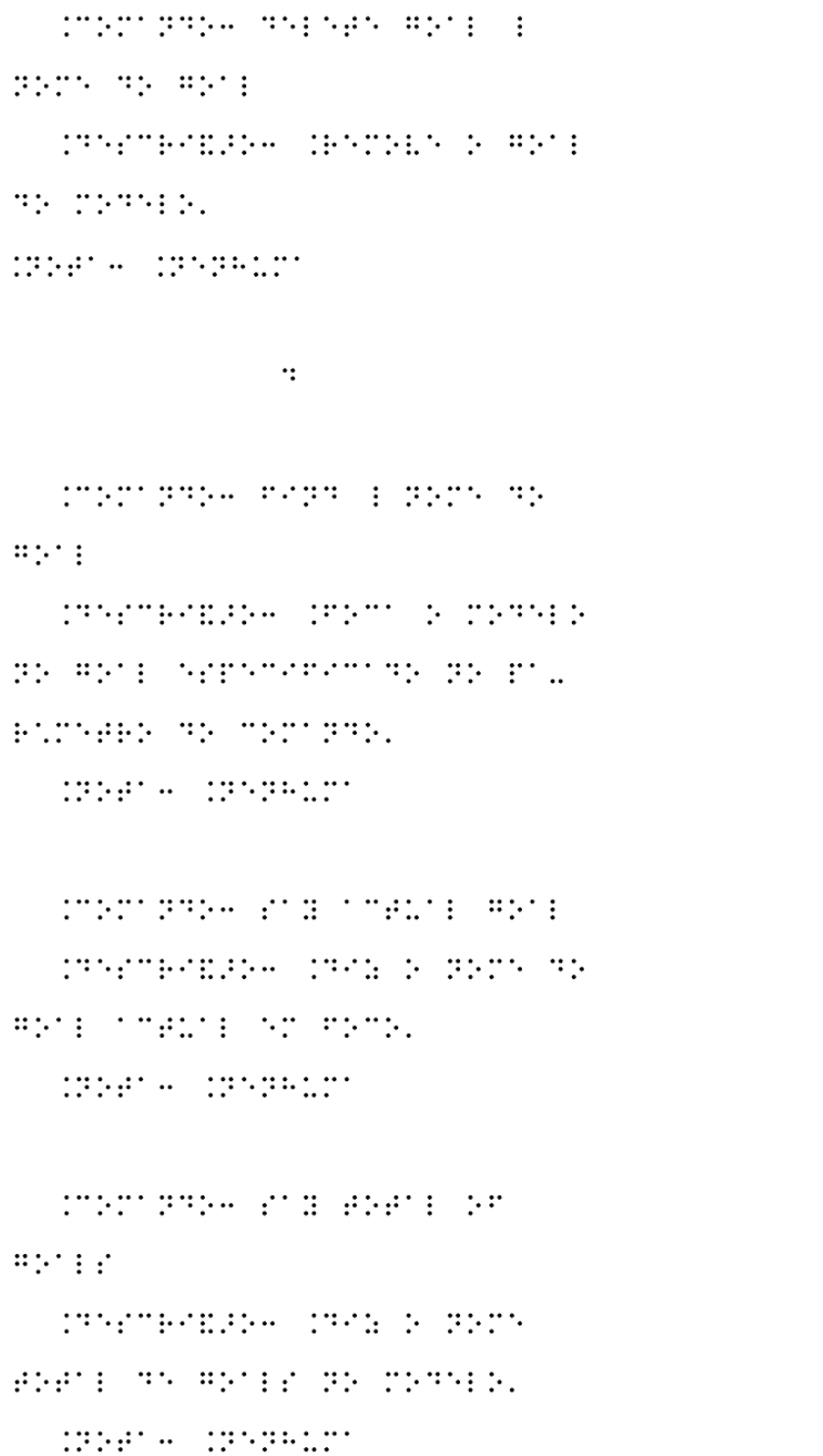


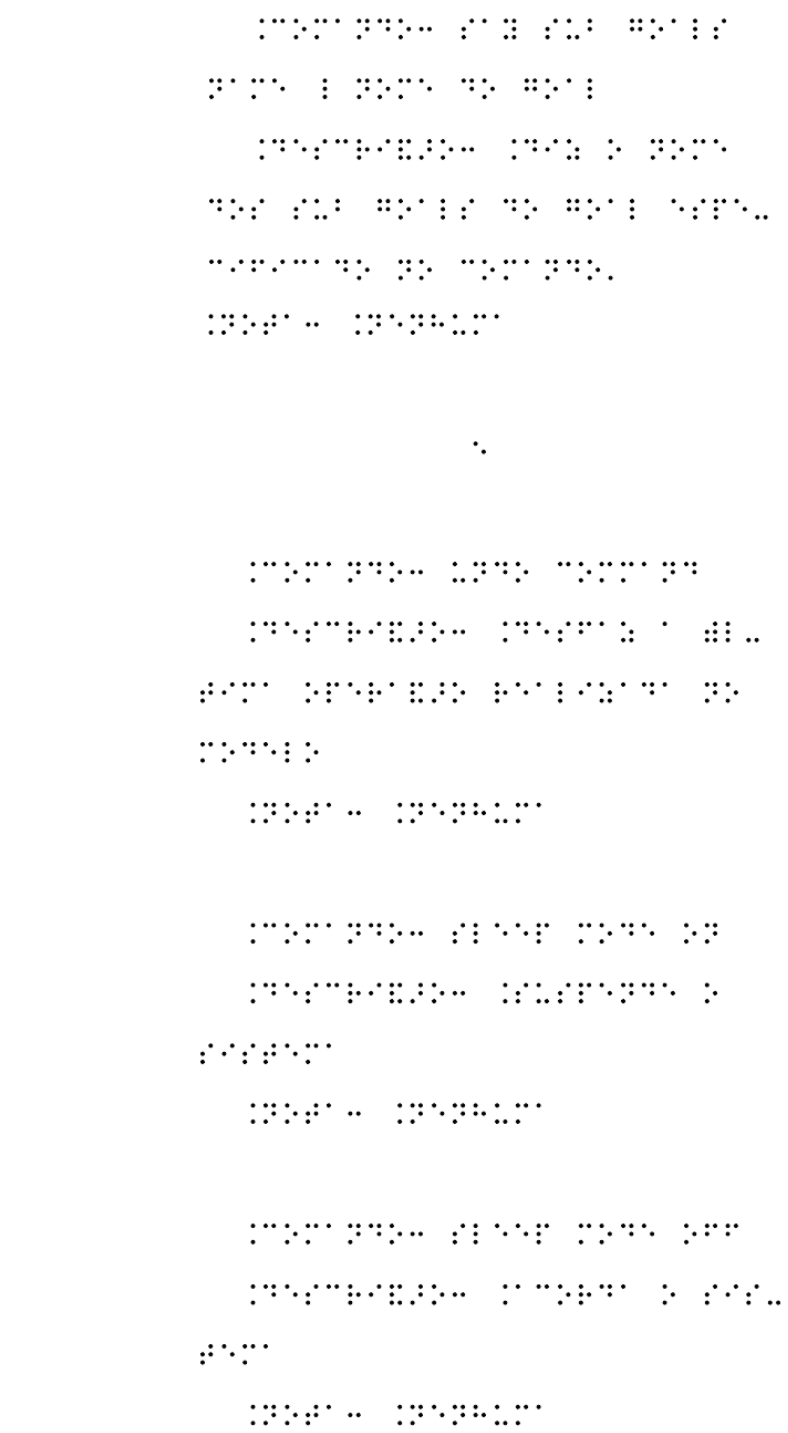
D.3 Comandos do SimpleKAOS

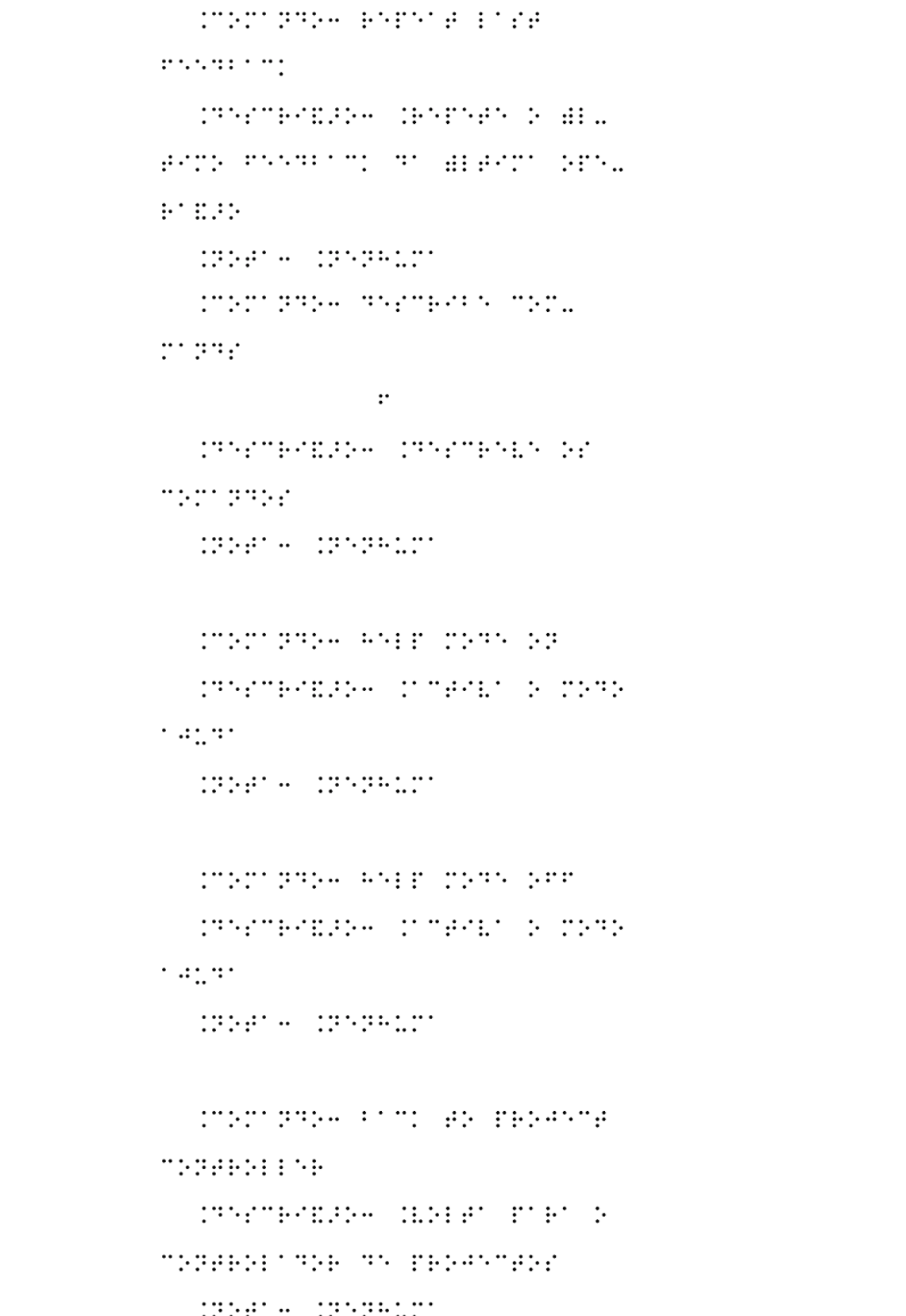




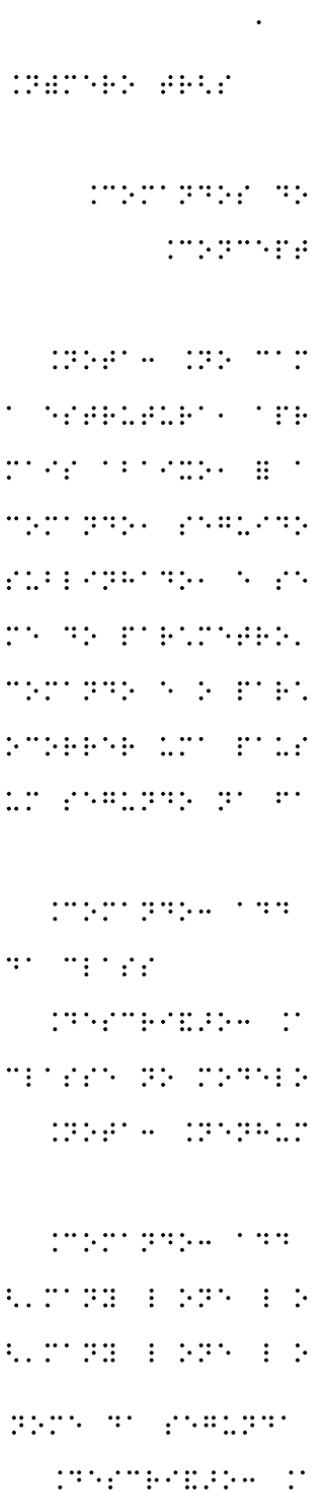


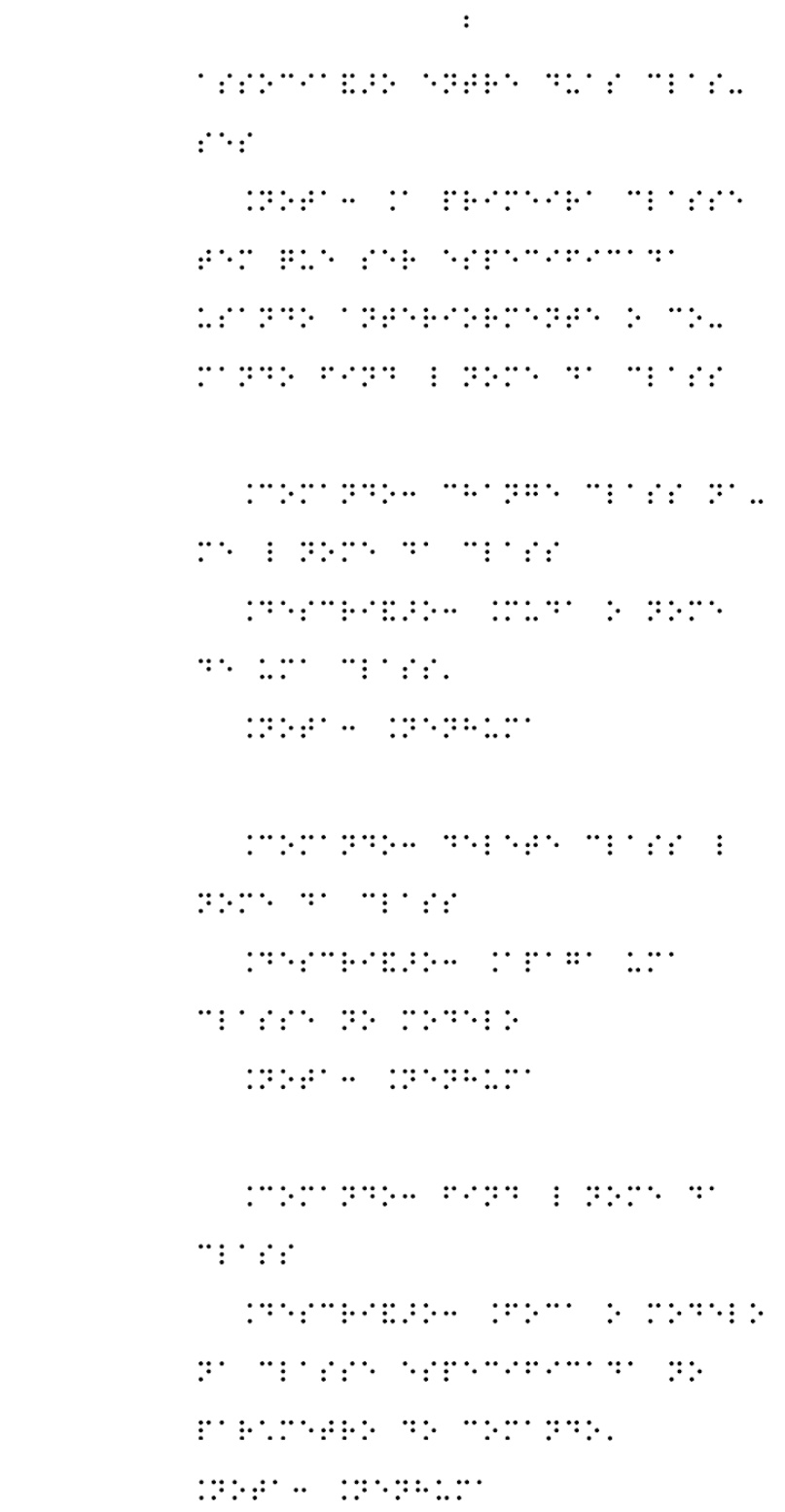


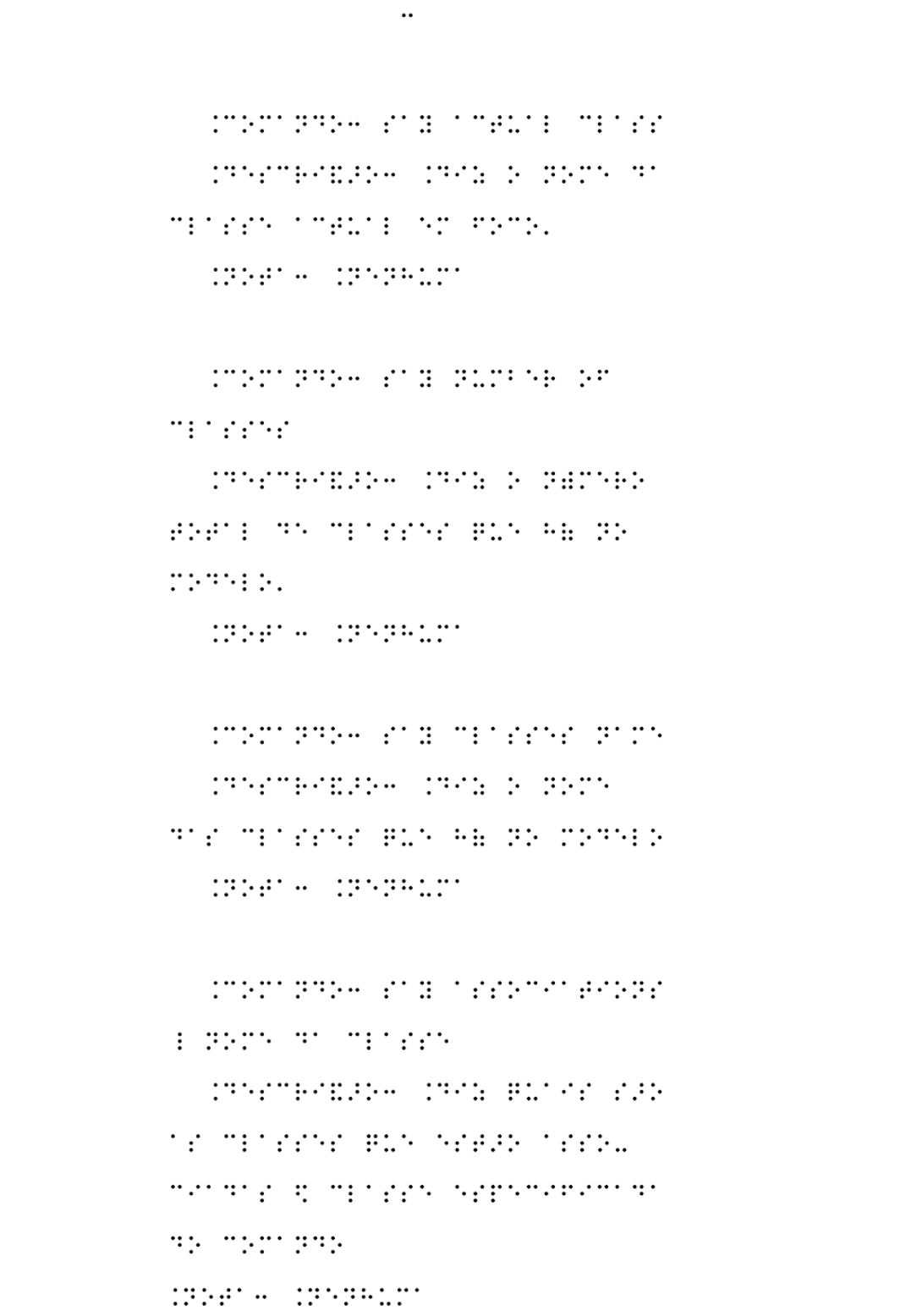


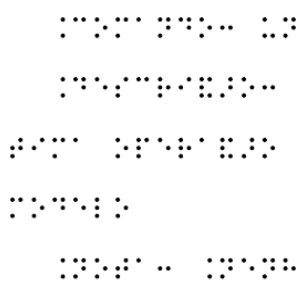


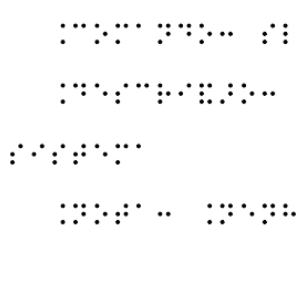
D.4 Comandos do Modelo Conceptual

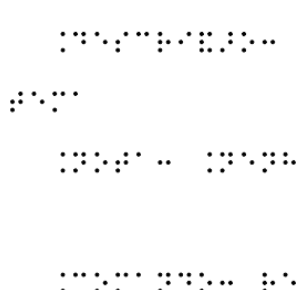


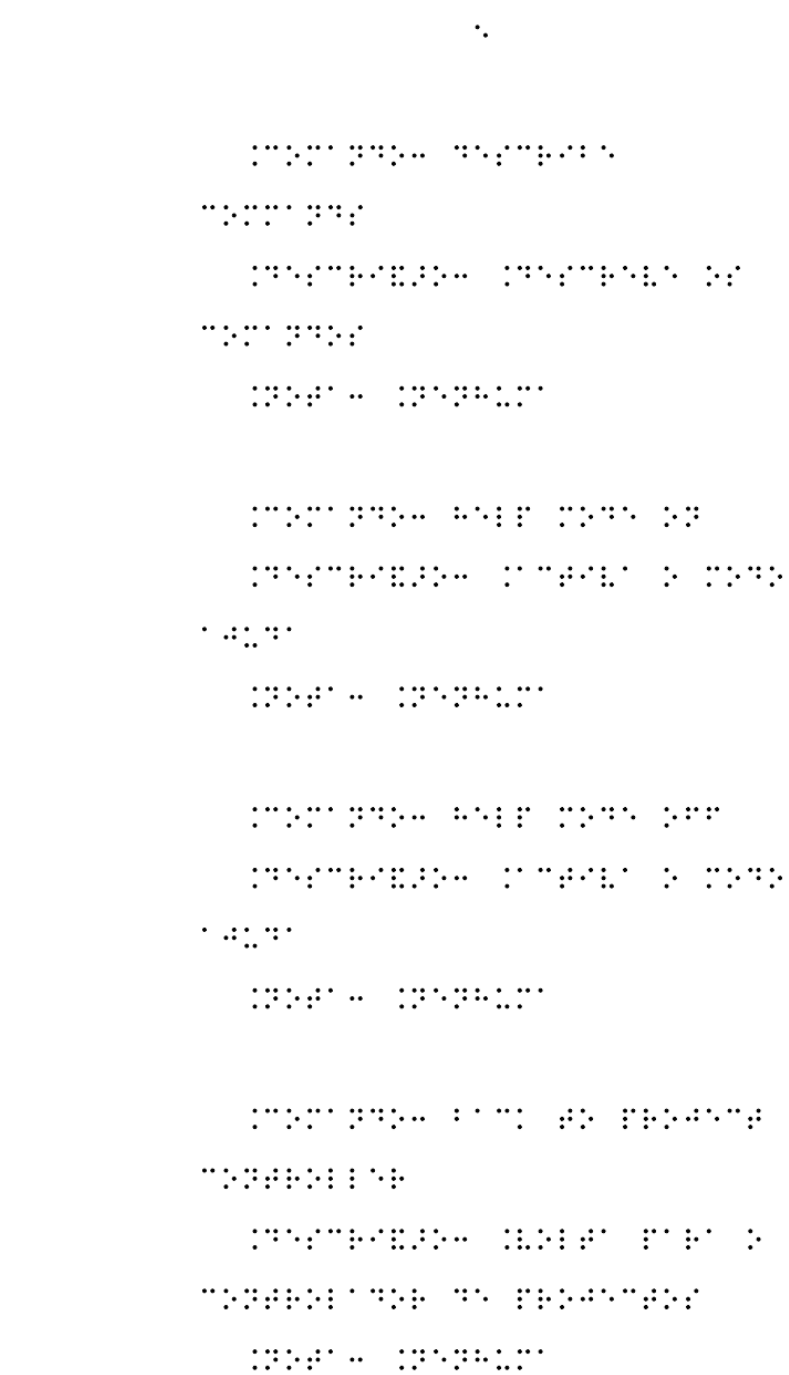












D.5 Comandos do Modelo de Features