WebMedia 2017

XXIII Simpósio Brasileiro de Sistemas Multimídia e Web

Realização SBC Sociedade Brasileira de Computação

17 a 20 Outubro

Gramado – RS – Brasil

Minicursos

Organizadores

Eduardo Barrére (UFJF)
Leandro Krug Wives (UFRGS)
Valter Roesler (UFRGS)
José Valdeni de Lima (UFRGS)
Roberto Willrich (UFSC)

Organização

UFRGS Universidade Federal do Rio Grande do Sul

.inf Instituto de Informática UFRGS

CINTED Centro Interdisciplinar de Novas Tecnologias na Educação

PPGIE Programa de Pós-Graduação em Informática na Educação

Patrocínio

nic.br cgi.br

CAPES

CNPq Conselho Nacional de Desenvolvimento Científico e Tecnológico

Cooperação

sigweb

acm In-Cooperation

SIGMM

Editora
Sociedade Brasileira de Computação (SBC)

XXIII Simpósio Brasileiro de Sistemas Multimídia e Web

De 17 a 20 de Outubro de 2017

Gramado – RS – Brasil

MINICURSOS

Sociedade Brasileira de Computação (SBC)

Organizadores

Eduardo Barrére (UFJF)

Leandro Krug Wives (UFRGS)

Valter Roesler (UFRGS)

José Valdeni de Lima (UFRGS)

Roberto Willrich (UFSC)

Realização

Sociedade Brasileira de Computação (SBC)

Em cooperação com

ACM/SIGWEB e ACM/SIGMM

Organização

Universidade Federal do Rio Grande do Sul (UFRGS)

Instituto de Informática da UFRGS

Centro Interdisciplinar de Novas Tecnologias Na Educação (CINTED)

Programa de Pós-Graduação em Informática na Educação (PPGIE)

FICHA CATALOGRÁFICA

Simpósio Brasileiro de Sistemas Multimídia e Web: Minicursos (23º: 2017: Gramado, RS).

Anais do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web: Minicursos, 17 a 20 de outubro, 2017, Gramado, Rio Grande do Sul.
228p.
E-book.
ISBN: 978-85-7669-382-6.

Evento promovido pela Sociedade Brasileira de Computação (SBC), Porto Alegre, RS. Organização: UFRGS.

1. Multimídia. 2. Computação Ubíqua e Móvel. 3. Web e Redes Sociais. 4. Vídeo colaboração. 5. Educação. I. Título

XXIII WebMedia

Simpósio Brasileiro de Sistemas Multimídia e Web
17 a 20 de outubro de 2017
Gramado, Rio Grande do Sul, Brasil

Comitês

Coordenação do WebMedia 2017

Valter Roesler (UFRGS) – *Coordenador Geral*
José Valdeni de Lima (UFRGS) – *Vice-coordenador Geral*
Celso Alberto Saibel Santos (UFES) – *Coordenador Geral do Comitê de Programa*
Roberto Willrich (UFES) – *Coordenador Adjunto do Comitê de Programa*

Coordenadores dos Minicursos

Eduardo Barrére (UFJF) – *Coordenador Geral*
Leandro Krug Wives (UFRGS) – *Coordenador Local*

Comitê de seleção de minicursos

Adriano C. Machado Pereira (UFMG)
Alessandra Macedo (USP)
Carlos de Salles Soares Neto (UFMA)
Fernando Antonio Mota Trinta (UFC)
Flávio Sousa (UFC)
Jairo Francisco de Souza (UFJF)
Leandro Ives (UFRGS)
Marcelo Ferreira Moreno (UFJF)
Marco Cristo (UFAM)
Mário Teixeira (UFMA)

Coordenação da Comissão Especial de Sistemas Multimídia e Web

Manoel Carvalho Marques Neto (IFBA) – *Coordenador*
Adriano C. Machado Pereira (UFMG) – *Vice-Coordenador*

Comitê Gestor

Álan Lívio (PUC-Rio)
Carlos de Salles Soares Neto (UFMA)
Celso Alberto Saibel Santos (UFES)
Fábio de Jesus Lima Gomes (IFPI)
Guido Lemos de Souza Filho (UFPB)
José Valdeni de Lima (UFRGS)
Luiz Fernando Gomes Soares (PUC-Rio) – *in memoriam*
Maria da Graça Campos Pimentel (USP)
Roberto Willrich (UFSC)
Valter Roesler (UFRGS)

Organização

Universidade Federal do Rio Grande do Sul (UFRGS)
Instituto de Informática da UFRGS
Centro Interdisciplinar de Novas Tecnologias Na Educação (CINTED)
Programa de Pós-Graduação em Informática na Educação (PPGIE)

Promoção

SBC – Sociedade Brasileira de Computação

Patrocinadores

NIC.br – Núcleo de Informação e Coordenação do Ponto BR
CGI.br – Comitê Gestor da Internet no Brasil
CAPES – Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CNPq – Conselho Nacional de Desenvolvimento Científico e Tecnológico

WebMedia'17 é realizado em cooperação com a ACM SIGMM e ACM SIGWEB

ACM SIGMM – Association for Computing Machinery Special Interest Group on Multimedia
ACM SIGWEB – Association for Computing Machinery Special Interest Group on Hypertext and the Web

Prefácio

Tradicionalmente, o Simpósio Brasileiro de Sistemas Multimídia e Web oferece à sua comunidade minicursos de curta duração relacionados a temas que norteiam os últimos avanços na área de multimídia e web. Este também é o caso do XXIII Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia 2017), onde os minicursos têm a duração de 4 horas e permitem que participantes recebam informações sobre novas tecnologias e tópicos atuais de pesquisa em áreas correlatas ao evento. Assim, minicursos aparecem como uma excelente oportunidade para familiarização dos congressistas com novos temas de pesquisa que podem vir a ser úteis em suas vidas profissionais.

Para o Webmedia 2017, o processo de seleção de minicursos foi feito a partir de uma chamada pública divulgada na lista eletrônica de emails da SBC, bem como ampla divulgação no site oficial do evento e em redes sociais. Foram recebidas 20 propostas de minicursos, avaliadas por comitê composto de professores com conhecimento nos temas abordados. Cada minicurso foi avaliado por pelo menos dois avaliadores, que pontuaram notas para quesitos como relevância para o evento, expectativa do público, atualidade e conteúdo de cada minicurso. Ao final, seis propostas foram selecionadas, cujos conteúdos abordados constituem os capítulos deste livro.

O primeiro capítulo tem como tema Objetos de Aprendizagem, destacando o fato de eles serem entidades que podem ser usadas, reutilizadas ou referenciadas durante o processo de aprendizagem, permitindo ao estudante individualizar sua experiência de aprendizagem com mecanismos de navegação não-linear e adaptação de conteúdo. Os autores abordam, como tema central, as recomendações pedagógicas e tecnológicas envolvidas na autoria de Objetos de Aprendizagem multimídia.

O capítulo dois apresenta um toolkit para desenvolvimento multiplataforma de aplicações que viabiliza o desenvolvimento, implantação e manutenção de soluções de software de forma rápida e com um baixo custo, chamado Qt. Este toolkit tem sido utilizado com sucesso no desenvolvimento de aplicações multimídia avançadas através de uma linguagem declarativa simples, denominada QML (Qt Modeling Language). A QML disponibiliza um conjunto de componentes de alto nível para manipulação de áudio e vídeo, animações, mapas, câmeras e sensores, como exemplo. Os autores abordam no transcorrer do capítulo as principais funcionalidades do Qt/QML voltadas ao desenvolvimento de aplicações multimídia multiplataforma.

No terceiro capítulo, os autores apresentam o desenvolvimento de jogos para pesquisadores que desejem analisar a experiência do usuário por meio da coleta de dados em suas pesquisas, mas encontram dificuldades no desenvolvimento de suas aplicações. A primeira parte do capítulo apresenta a teoria sobre a concepção de jogos destacando as definições e exemplos sobre a coleta de dados envolvendo diretrizes e usabilidade, a segunda e última parte apresenta uma abordagem para o desenvolvimentos de jogos por meio da ferramenta Construct 2, utilizando recursos multimídia (áudio, vídeos e fotos) e variáveis (quantitativas e qualitativas) visando o armazenamento dos dados gerados du-

rante a interação do usuário.

O quarto capítulo aborda a Engenharia Dirigida por Modelos (Model-driven Engineering - MDE), a qual considera os modelos como os principais artefatos no desenvolvimento de um software. Modelos são geralmente construídos usando linguagens específicas de domínio, como a UML e a XML. Essas linguagens, por sua vez, são definidas por metamodelos próprios. Nesse contexto, esse capítulo tem como objetivo apresentar os fundamentos de MDE, assim como os principais frameworks e linguagens disponíveis para o seu suporte, com foco na construção de aplicações ubíquas. A ideia é que o leitor entre em contato com os conhecimentos necessários para construir uma ferramenta de modelagem gráfica que possibilite construir modelos em conformidade com um metamodelo em particular. Esses modelos podem então ser usados para documentar e manter sistemas de diferentes domínios.

O capítulo cinco é baseado no fato de que a necessidade de recuperar informações em conteúdo multimídia aumenta a demanda por sistemas que usam o reconhecimento automático de fala. Esses sistemas de reconhecimento de voz permitem que o computador intérprete sinais de áudio, gerando transcrições textuais aproximadas. Eles são baseados em modelos probabilísticos que criam um modelo robusto e correto para a fala humana. A estrutura do capítulo apresenta inicialmente uma arquitetura de sistemas de reconhecimento de fala e a descrição de seus componentes básicos (modelo acústico, modelo de linguagem, lexical e decodificador). Em seguida é abordado o processo de treinamento de modelos acústicos e de linguagem Para finalizar, é apresentado como esses sistemas podem ser usados em algumas aplicações.

Por fim, o sexto capítulo tem como foco Mobile Cloud Computing (MCC) e Computação Sensível ao Contexto. A MCC procura utilizar recursos da computação em nuvem para melhorar o desempenho de aplicações móveis e reduzir o consumo de energia dos dispositivos, enquanto a Computação Sensível ao Contexto busca formas eficazes de criar aplicações que reajam às mudanças de contexto do ambiente. O capítulo tem como objetivo apresentar os principais conceitos, soluções e tecnologias relacionadas à integração de MCC e sensibilidade ao contexto. São apresentados diferentes cenários motivacionais, exemplos de aplicações, bem como um guia prático de como desenvolver uma aplicação multimídia sensível ao contexto utilizando o framework CAOS. Além disso, são discutidos desafios e oportunidades de pesquisa relacionados à integração entre os dois tópicos.

Esperamos que este livro seja útil para todos aqueles interessados e praticantes da área de Sistemas Multimídia e Web.

Gramado, outubro de 2017.

Eduardo Barrére (UFC) – *Coordenador Geral*
Leandro Krug Wives (UFRGS) – *Coordenador Local*
Coordenadores dos Minicursos do WebMedia'17

Sumário

Capítulo 1

Creating Multimedia Learning Objects

Carlos de Salles Soares Neto[1], Thacyla de Sousa Lima[1],
André Luiz de B. Damasceno[2] e Antonio José G. Busson[2]

[1]Laboratório TeleMídia-MA, Departamento de Informática, Universidade Federal do Maranhão (DEINF/UFMA), Brasil

[2]Laboratório TeleMídia, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Brasil

csalles@deinf.ufma.br, thacyla@laws.deinf.ufma.br,
andre@telemidia.puc-rio.br, busson@telemidia.puc-rio.br

Abstract

Learning Objects (LOs) are entities that can be used, reused, or referred during the teaching process. LOs allow students to individualize their learning experience with non-linear browsing mechanisms and content adaptation. The main goal of this tutorial is to discuss both the pedagogical and technological recommendations involved in the authoring of multimedia LOs.

Resumo

Objetos de Aprendizagem (OAs) são entidades que podem ser usadas, reusadas ou referenciadas durante o processo de aprendizagem. OAs permitem a estudantes individualizarem sua experiência de aprendizagem com mecanismos de navegação não-linear e adaptação de conteúdo. O objetivo central deste tutorial é discutir ambas recomendações pedagógicas e tecnológicas envolvidas na autoria de OAs multimídia.

1.1. Introdução

O desenvolvimento de Tecnologias da Informação e Comunicação (TICs) tem impactado todos os setores da sociedade, incluindo o setor educacional. No ensino superior, a aplicação de TICs na forma de e-learning tem mudado as maneiras de ensinar e aprender. Os benefícios para a sua adoção incluem o aumento da colaboração e cooperação, a possibilidade de atingir diferentes estudantes em grande número, as melhorias pedagógicas e o uso de recursos multimídia (slides, vídeos ou jogos) como Objetos de Aprendizagem (OA).

Nesses ambientes, professores e alunos podem escolher as aplicações mais apropriadas que possuem tempo e locais flexíveis e são reusáveis, adaptativas e personalizáveis.

Dentre as metodologias que mais se destacam nesse cenário está o Aprendizado Baseado em Vídeo (ABV). O ABV não é um fenômeno recente, porém tem ganhado foco como resultado das novas formas de ensinar e o crescimento de ambientes de e-learning como MOCCs (Massive Open Online Courses). Entretanto, a simples incorporação de vídeos em ambientes de e-learning não é suficiente para aumentar o aprendizado. OAs baseados em hipervideos ou Vídeos Interativos (VI) oferecem o controle de acesso ao conteúdo individual, uma melhor experiência de aprendizado e uma maior satisfação do aluno.

Além das funcionalidades de interação, VIs também podem ser enriquecidos com conteúdo multimídia adicional, isto é, faz-se uso de outros elementos como gráficos, desenhos e clipes de áudio sincronizados com o vídeo principal para proporcionar alívio de cansaço ao aluno. Há estudos que mostram que o emprego de OAs como vídeos interativos, enriquecidos com conteúdo multimídia, possuem um melhor desempenho de aprendizagem quando comparado com a abordagem clássica de vídeo Brecht (2012). Entretanto, para o planejamento e elaboração de OAs multimídia interativos se faz necessária uma equipe de profissionais de diferentes áreas de conhecimento, o que resulta em maior tempo de desenvolvimento e no alto custo de produção. Programadores são necessários para codificar tais OAs. Designers ajudam a estabelecer uma identidade visual entre diferentes OAs. Pedagogos traçam e medem objetivos didáticos. No centro dessa equipe está o professor conteudista, aquele que detém o conhecimento do assunto.

Fazendo uma comparação com a Web e sua evolução, no início as páginas eram desenvolvidas apenas por programadores ou profissionais com conhecimento específico de linguagem de marcação. Já nos tempos recentes, grande parte do conteúdo Web é feito por não-programadores Paternò (2013). Isso engloba, por exemplo, perfis em redes sociais com vídeos, textos, fotos e toda sorte de conteúdo multimídia, o que é feito por usuários sem conhecimento específico ou treinamento. Envolve também jornalistas que criam e gerenciam blogues sem a necessidade da ajuda de um profissional especializado em programação. Seguramente o processo de transferência da autoria de conteúdo na Web saindo das mãos de programadores para o próprio usuário final é um dos diversos fatores que ajudam a explicar a explosão global em seu uso.

O que se espera é que esse cenário seja replicado para a autoria de aplicações multimídia, como por exemplo objetos de aprendizagem que sirvam como recurso didático, englobando determinado conteúdo de uma disciplina. Somado a isso, também é desejável permitir ao professor criar sozinho conteúdo através de uma ferramenta de autoria de alto nível. A intenção, no entanto, não é a de se tornar dispensável ou de substituir a equipe multidisciplinar. Pretende-se, no entanto, permitir que o professor tenha a liberdade e capacidade de poder criar objetos de aprendizagem sozinho para usar em sua turma. Alguns exemplos são aplicações hipermídia que permitem que o usuário possa escolher entre quais vídeos assistir, ou quais informações dentro do conteúdo sendo assistido ele gostaria de ver. Cada usuário diferente personaliza o conteúdo e o assiste de acordo com suas próprias preferências.

Objetos de aprendizagem (ou OAs) Polsani (2006) são qualquer entidade digital que

possa ser usada e reusada no processo de ensino e aprendizagem apoiado por computador. OAs multimídia, por sua vez, são aqueles que utilizam objetos de diferentes monomídias, como imagem, áudio, vídeo, texto, todos eles sincronizados entre si em uma mesma aplicação. Há desafios pedagógicos para se construir OAs que sejam realmente úteis para o aprendizado, estejam eles sendo empregados no apoio ao ensino presencial, ou sejam eles elementos centrais em um curso de Ensino à Distância (EaD). Há ainda os desafios tecnológicos, que podem interpor custo e tempo excessivos na preparação dos OAs.

Este minicurso tem como foco discutir tanto as recomendações pedagógicas quanto tecnológicas envolvidas na construção de OAs multimídia. Em especial, o minicurso prepara o participante para que possa criar OAs baseados em hipervídeo, uma categoria específica de OAs em que o vídeo é o elemento central do conteúdo multimídia, porém com a possibilidade do aluno individualizar sua experiência navegando no conteúdo de forma interativa. Para tanto, serão discutidas boas práticas e apresentado um guia de recomendações que expressam a experiência adquirida pelos autores nos últimos anos não apenas na construção de OAs diretamente mas também no acompanhamento de professores de diferentes áreas o fazendo.

Ferramentas de software são apresentadas no minicurso para a preparação de conteúdo educacional, desde informações sobre como capturar e editar os vídeos e imagens, até a preparação da aplicação multimídia final. Um destaque do minicurso é o treinamento com a ferramenta brasileira Cacuriá, também de autoria dos proponentes do minicurso, que permite, através de seu ambiente de autoria, a construção de vídeos interativos para Web e TV Digital sem a necessidade de conhecimentos prévios de programação.

Por meio do emprego da ferramenta de autoria multimídia Cacuriá Damasceno et al. (2014a,b, 2015), o professor tem a possibilidade de construir OAs baseados em vídeos Busson et al. (2017). A metáfora central de autoria do Cacuriá envolve a organização de objeto de aprendizagem como uma sequência de cenas, cada uma delas possuindo um vídeo, imagens, texto e áudio sincronizados com a linha temporal da cena Busson et al. (2016). Toda a manipulação da cena é direta, usando uma abordagem de autoria WYSIWYG (What You See Is What You Get) de tal forma que o autor tem o resultado final da aplicação (Objeto de Aprendizagem) sendo criado diretamente na tela. A ferramenta de autoria Cacuriá foi desenvolvida pelo Laboratório de Sistemas Avançados da Web (LAWS) da Universidade Federal do Maranhão Neto et al. (2015), em parceria com a Mediabox Technologies e com apoio da Rede Nacional de Ensino e Pesquisa (RNP).

A presente proposta apresenta uma gama de aplicações práticas possíveis para o enriquecimento no processo de ensino e aprendizagem com o Cacuriá. Com a interatividade disponível sobre os conteúdos audiovisuais é possível criar aulas e lições explorando a não-linearidade da apresentação. Assim os autores, sendo eles professores ou criadores de objetos de aprendizagem, podem fornecer âncoras para conteúdos complementares ao OA. A possibilidade de interação com a apresentação propicia um ambiente investigativo, mais intrigante e interessante aos alunos, que poderiam decidir o seu próprio roteiro através da apresentação.

Por essas razões, acreditamos que a proposta de minicurso seja de interesse tanto de professores quanto de alunos. Para os professores, é cada vez mais importante que os mesmos criem a aptidão necessária para criar videoaulas interativas. O processo de

aprendizagem em sala de aula invertida, por sua vez, requer que o próprio aluno crie conteúdo multimídia educacional sobre os tópicos que está aprendendo.

O texto deste minicurso está organizado da seguinte forma:

1. Introdução: O minicurso é iniciado com a apresentação do conceito e da importância de objetos de aprendizagem na educação. Depois é mostrado como a multimídia pode apoiar a autoria, o armazenamento e a distribuição de OAs.

2. Projeto de OAs: Boas práticas e recomendações para o planejamento de OAs multimídia. Apresenta-se como relacionar a matriz curricular com a modelagem dos OAs. Diagramas são sugeridos para facilitar o planejamento.

3. Utilizando a ferramenta Cacuriá: Neste tópico é mostrado como instalar a ferramenta de autoria Cacuriá e a abstração utilizada para criação de OAs. É detalhado o processo de criação de OAs mostrando como inserir, remover e sincronizar mídias (vídeo, texto, imagem e formas) entre si. Depois são abordados os métodos utilizados pela ferramenta para criação de conteúdo linear (sem interatividade) e não-linear (com interatividade). Finalmente se trata da publicação e exportação de OAs para HTML5 e NCL.

4. Conclusão: considerações finais do minicurso.

1.2. Projeto de Objetos de Aprendizagem

O processo de desenvolvimento tradicional de software a grosso modo pode ser decomposto em duas grandes etapas: uma inicial que envolve a modelagem e uma final que envolve a implementação. Outra visão similar é a de entender o desenvolvimento de software como um processo de análise-síntese. Durante a análise é detalhado o artefato de software que se deseja construir, o que inclui, por exemplo, obter informações sobre as necessidades ou requisitos que o novo software precisa atender, projetar diversos diagramas de apoio que traçam uma descrição do artefato que precisa ser construído em vários níveis de abstração, entre outras tarefas. Durante a síntese, o software é implementado seguindo uma metodologia de desenvolvimento específica e refletindo ao máximo o planejado durante a etapa de análise.

A forma como o projeto de software é tradicionalmente conduzido é o ponto de partida para se idealizar o projeto de objetos de aprendizagem multimídia. Nesta seção, são discutidas boas práticas adquiridas com a experiência no desenvolvimento ou acompanhamento de terceiros criando OAs multimídia.

1.2.1. Matriz Disciplinar

A matriz é o documento que expressa o escopo e forma do conteúdo que será trabalhado por uma disciplina. O documento comumente tem como cabeçalho os seguintes itens: i) Dados de Identificação; ii) Ementa; iii) Objetivos; iv) Referências. O corpo do documento, por sua vez, detalha o planejamento das atividades a serem executadas durante o decorrer do curso.

Nos dados de identificação há informações sobre o curso, professor, tutores e carga horária, entre outras. A ementa relaciona os itens do conteúdo a serem pedagogicamente trabalhados. O objetivo geral é o propósito central daquela disciplina, o que normalmente vem acompanhado de objetivos específicos que detalham propósitos colaterais daquele curso. Entre as referências deve-se explicitar as referências básicas a serem adotadas pelos alunos e referências complementares que apoiam tópicos específicos do curso.

A Figura 1.1 apresenta um exemplo de Matriz de Disciplina (cortesia do NEAD/UFMA), especificamente com exemplos de atividades organizadas por semana. Como se vê, o documento define a Modalidade (Presencial, Ambiente Virtual de Aprendizagem etc), Conteúdos, Atividades, Materiais Didáticos e forma de Avaliação.

NÚCLEO DE EDUCAÇÃO A DISTÂNCIA - NEAD

A Universidade que Cresce com Inovação e Inclusão Social

COORDENAÇÃO PEDAGÓGICA DE HIPERMIDIA PARA A APRENDIZAGEM-CPHA

MATRIZ DA DISCIPLINA

Data Carga Horária	Modalidade	Conteúdos	Atividades	Materiais didáticos	Avaliação
1ª Semana 14/08 a 19/08 Total: 12h	AVA	- Histórico e evolução dos computadores. - Conceitos e princípios de funcionamento;	Assistir Vídeos e leitura do material da unidade; **Atividade 1**: Leitura do SLIDE UNID - I - INTRODUÇÃO A INFORMÁTICA, CONCEITOS E PRINCÍPIOS DE FUNCIONAMENTO DOS COMPUTADORES. **Atividade 2**: Construir um texto (4 páginas no máximo) enfatizando a cronologia dos principais fatos que marcaram a evolução dos computadores no mundo, no Brasil e no Maranhão.	**Material da UNID- I** **E-Book** – Introdução a Informática **Vídeos 1** – Evolução dos computadores https://youtu.be/mFdUqgwzbVs	**1ª Avaliação:** Fórum 1: valor: 2,5 Atividade 2: valor: 5,0
	Encontro Presencial 26/08 4h Professor e Tutor Presencial no Polo	Plano de Ensino. Apresentação da Ementa com os Procedimentos Metodológicos e Avaliativos.	Aula dialogada para apresentação da disciplina, utilização de slides e vídeos apresentando o conteúdo da aula. Utilização de Laboratório de informática para aplicação de conhecimentos e utilização de softwares.	**Slides** – Introdução a Informática, conceitos e princípios de funcionamento dos computadores **Vídeos 1** – Evolução dos computadores https://youtu.be/mFdUqgwzbVs	**1ª Avaliação:** Será considerada a participação individual e coletiva.

Figura 1.1. Exemplo de Matriz Disciplinar. Fonte: Cortesia do NEAD/UFMA.

A Matriz Disciplinar é o documento base que pode ser adotado para iniciar o processo de construção de novos objetos de aprendizagem multimídia. Para essa finalidade, o documento é útil principalmente em apoiar o planejamento do roteiro para os vídeos e os tópicos que devem ser trabalhados.

Um resultado direto da análise da Matriz Disciplinar é a modularização dos tópicos da ementa de acordo com as atividades planejadas, o que conduz a uma tabela que resume o conteúdo que será produzido como OA. Um exemplo é a Tabela 1.1 extraída de Portela (2017).

Tabela 1.1. Exemplo de planejamento de módulos e tópicos. Adaptado de Portela (2017).

N. Módulo	Módulo	N. Tópico	Tópico
1	Fundamentos	1	Apresentação do curso
		2	Conceito de algoritmo
		3	Conceito de lógica de programação
2	Tipos de dados	4	Conceito de dados
		5	Inteiro
		6	Real
		7	Caractere
		8	Lógico
		9	Cadeia de Caracteres

1.2.2. Planejamento Hierárquico do Conteúdo

O planejamento de OAs não é similar ao de vídeos. Isso porque vídeos possuem um roteiro linear, com início, meio e fim definidos. Uma aula em vídeo tem sempre a mesma sequência linear de tempo que o aluno precisa seguir. Por outro lado, um OA permite que o aluno navegue de diversas formas diferentes.

Sugere-se, portanto, que o conteúdo seja ordenado hierarquicamente como um diagrama de atividades UML de forma que represente a navegação ideal para o aprendizado da disciplina, com suas possibilidades de escolha. A Figura 1.2 mostra um exemplo de diagrama de sequência de conteúdo para os primeiros dois módulos de um curso de programação de computadores.

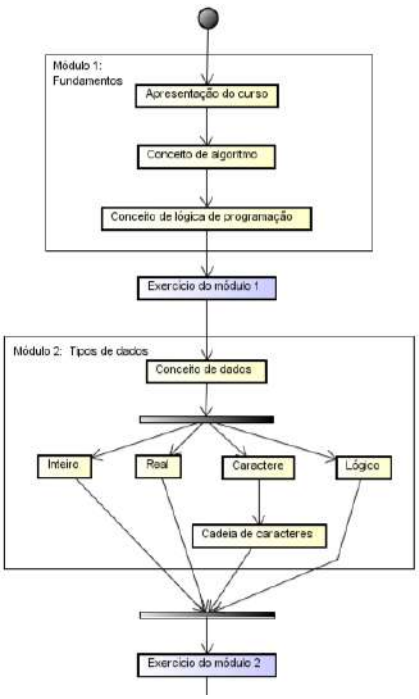


Figura 1.2. Exemplo de planejamento de módulos e tópicos. Fonte: Portela (2017).

Posteriormente, cada tópico (videoaula) pode ser descrito superficialmente, como mostrado no exemplo da Tabela 1.2. Para cada videoaula, foram identificadas videoaulas anteriores que poderiam ser mencionadas na videoaula em questão. A coluna Hiperlink da tabela exibe esses relacionamentos. O hiperlink seria representado, por exemplo, através de um botão na videoaula que proporciona ao aluno retornar a uma videoaula anterior sem perder seu progresso na atual.

Tabela 1.2. Hiperlinks entre os tópicos da videoaula. Adaptado de Portela (2017).

N°	**Título da videoaula**	**Descrição do conteúdo**	***Hiperlink***
39	Conceito de estrutura de repetição	Explicar o conceito desse tipo de estrutura e mostrar exemplos para evidenciar sua importância na elaboração de algoritmos. Apresentar problema para ser resolvido nas videoaulas seguintes. Apresentar os tipos básicos de estruturas de repetição.	
40	Repita	Explicar o significado, sintaxe e uso da estrutura. Resolver o problema da primeira videoaula.	39
41	Enquanto		
42	Para		
43	Laços infinitos	Apresentar o conceito e mostrar exemplos.	39, 40, 41, 42
44	Laços animados	Apresentar situações onde é necessário o uso de laços aninhados. Mostrar exemplos.	

1.3. Ferramenta Cacuriá

A ferramenta Cacuriá foi desenvolvida pelo Grupo de Trabalho "Vídeos sob Demanda como Objetos de Aprendizagem"(GT-VoA) que fez parte do programa de GTs da RNP (Rede Nacional de Ensino e Pesquisa) a partir do ano de 2012.

O projeto que deu origem à ferramenta Cacuriá visou permitir a criação e exibição colaborativa de objetos de aprendizagem para ensino à distância como conteúdo multimídia interativo integrado ao repositório de vídeos do serviço de videoaula da RNP. O projeto foi dividido em duas etapas. Na primeira delas foi proposto o desenvolvimento da ferramenta de autoria Cacuriá, a qual teve como foco permitir a usuários finais criar objetos de aprendizagem enriquecendo tais vídeos com conteúdo multimídia adicional (imagens, formas e textos), usando a mesma estrutura de distribuição já existente. A segunda etapa do projeto propôs um software integrado ao serviço de vídeo sob demanda da RNP que torna transparente a exibição para o usuário do conteúdo multimídia interativo enriquecido.

A ferramenta Cacuriá permite a conversão de aplicações Cacuriá em código NCL (Nested Context Language) ou HTML5. A linguagem de programação NCL é adotada pela ISDB-Tb (International Services for Digital Broadcast, Terrestial Brazilian) para especificação de aplicações interativas na parte declarativa de seu middleware Ginga. O ITU-T (International Telecommunication Union) definiu NCL como recomendação para serviços IPTV. HTML 5 é a linguagem de marcação para estruturação e apresentação de conteúdo para a World Wide Web, o que possibilita a execução dos OAs produzidos no Cacuriá em AVAs que permitem adição desse tipo de aplicação. A Figura 1.2, a seguir, descreve a arquitetura básica do projeto.

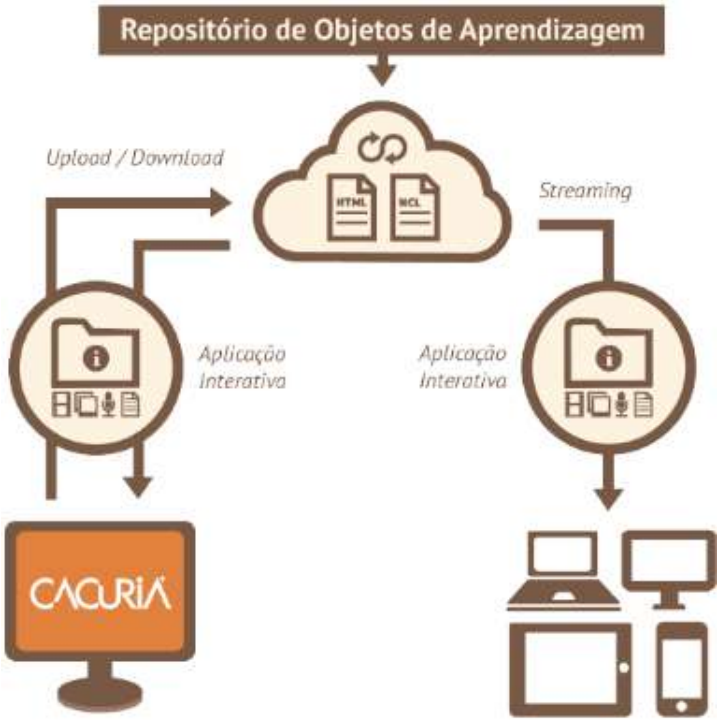


Figura 1.3. Arquitetura de serviço do projeto GT-VoA.

O resto desta seção está organizado como se segue. A Seção 1.3.1 descreve como é feita a instalação da ferramenta Cacuriá. A Seção 1.3.2 apresenta uma visão geral da ferramenta. A seguir, na Seção 1.3.3, há diversos exemplos que apresentam os recursos da ferramenta.

1.3.1. Instalação da ferramenta Cacuriá

A ferramenta possui versões para Windows, MAC e para Linux, os quais podem ser baixados diretamente do site dos desenvolvedores. Selecione todas a opções de instalação das telas do programa. Logo após essa instalação será possível abrir o executável da ferramenta de autoria Cacuriá. A Figura 1.4 mostra telas da instalação.

1.3.2. Visão Geral do Cacuriá

A Figura 1.5 mostra que a ferramenta Cacuriá possui 6 visões. A Visão de Menu (1) é usada para adição de mídias, publicação do OA e desfazer e refazer ações. A Visão de Cenas (2) é usada para manipular as cenas no projeto. Por meio dessa visão também é possível adicionar, remover e editar elos automáticos entre as cenas. A Visão de Leiaute (3) permite visualizar o posicionamento e dimensionamento das mídias de acordo com o tempo. A Visão Temporal (4) é utilizada para executar e manipular o tempo da cena. Nessa visão também é possível visualizar, mover e remover os marcadores temporais de cada mídia da cena. A Visão de Propriedades (5) é usada para visualização e edição das propriedades da mídia em foco e criação de links interativos entre as cenas. A Visão de Biblioteca (6) é utilizada para listar e renomear as mídias da cena atual e do projeto.

Figura 1.4. Instalação da ferramenta Cacuriá.

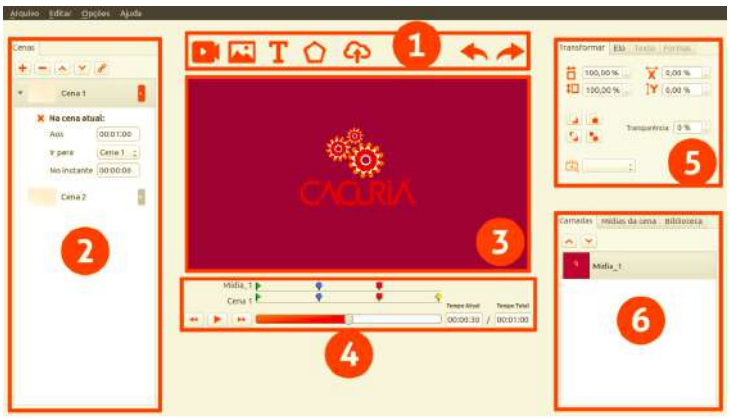
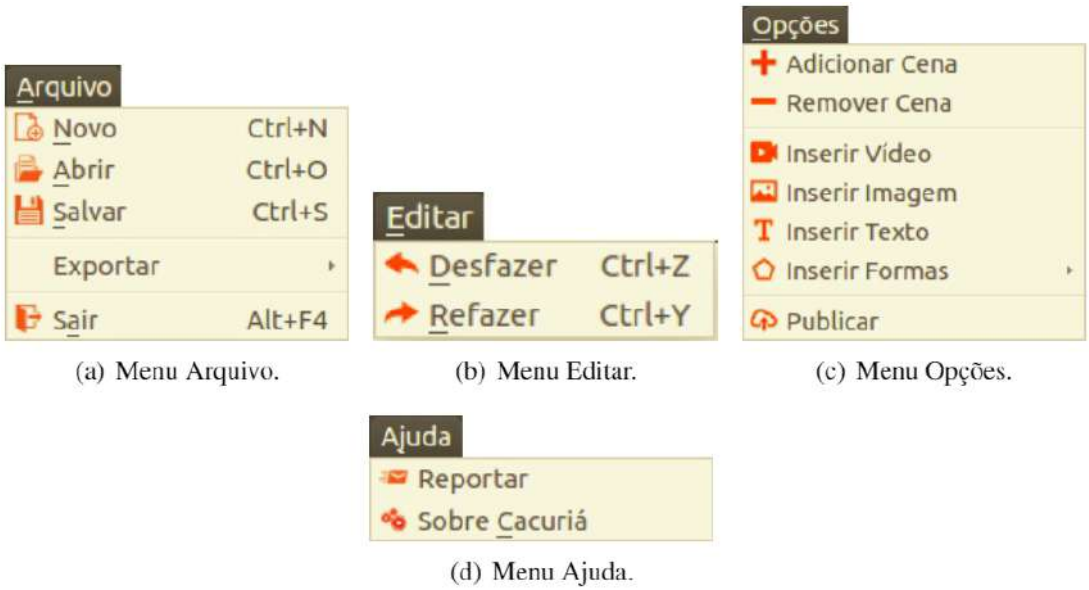


Figura 1.5. Tela inicial da ferramenta Cacuriá.

Menus

Na ferramenta também são disponibilizados três menus, mostrados na Figura 1.6: Menu de Arquivo, Menu Editar, Menu Opções e Menu Ajuda. O menu Arquivo permite criar, abrir e salvar projetos. O menu Editar lista as principais operações de edição. O menu

Opções lista os principais comandos para o desenvolvimento do projeto, do gerenciamento de cenas à inserção de mídias. Já o menu Ajuda tem como objetivo auxiliar na resolução de problemas de uso da ferramenta e exibir detalhes do desenvolvimento da mesma.



(a) Menu Arquivo. (b) Menu Editar. (c) Menu Opções.

(d) Menu Ajuda.

Figura 1.6. Menus da ferramenta Cacuriá.

Metáfora de Autoria

A metáfora central de autoria do Cacuriá envolve a organização de objeto de aprendizagem como uma sequência de cenas, cada uma delas possuindo um vídeo, imagens, texto e áudio sincronizados com a linha temporal da cena Busson et al. (2016). Toda a manipulação da cena é direta, usando uma abordagem de autoria WYSIWYG (What You See Is What You Get) de tal forma que o autor tem o resultado final da aplicação (Objeto de aprendizagem) sendo criado diretamente na tela.

No Cacuriá, um novo projeto já possui uma cena incorporada ao objeto de aprendizagem em construção, esta representação é observada no lado esquerdo da ferramenta, na área chamada visão de Cenas, como mostra a Figura 1.7. Além da lista das cenas do projeto, encontra-se ainda na visão de Cenas a barra de ferramentas da Lista de Cenas.

O primeiro botão da barra de ferramenta, da esquerda para direita, é o botão Adicionar Cena, que permite inserir uma nova cena vazia no fim da lista de cenas. O segundo botão é o botão Remover Cena, que permite remover a cena selecionada da lista de Cenas. Adjacente a este está localizado o botão Enviar Cena para cima, que tem como ação alterar a posição da cena selecionada, fazendo-a subir uma posição. O quarto botão, Enviar cena para baixo, também altera a posição da cena, fazendo-a descer uma posição. O último botão é chamado Criar elo automático. Este possibilita configurar transições entre uma ou mais cenas.

Figura 1.7. Visão de Cenas.

Elementos da tela

A Visão de Menu é composta pela barra de ferramentas da Cacuriá, mostrada na Figura 1.8. Por meio dos ícones dessa barra é possível adicionar as mídias no OA. O primeiro ícone é o Vídeo, que permite inserir um vídeo em uma cena. O vídeo aparecerá na área de trabalho da ferramenta. Vale destacar que cada cena só pode ter um vídeo, o que é uma restrição criada para evitar a criação de OAs confusos para o aluno. Assim, quando a cena já possuir um vídeo, o ícone Vídeo é desabilitado. Adjacente a esse tem-se o ícone Imagem, responsável por permitir inserir imagens em uma cena da ferramenta. O terceiro ícone da barra é o ícone Texto, que permite a inserção de conteúdo textual nos OAs. O próximo ícone é o ícone Formas, que possibilita inserir formas distintas já existentes na ferramenta ajudando a composição do projeto. Na barra de ferramentas também está contido o ícone Publicar, que ao ser acionado exporta o objeto de aprendizagem para repositórios. Nesta Visão ainda estão presentes os ícones Desfazer (cancela uma ação do usuário) e Refazer (replica a última ação desfeita).

Figura 1.8. Barra de ferramenta principal da Cacuriá.

Área de Trabalho

A Figura 1.9 mostra a Área de Trabalho da ferramenta Cacuriá. Nesta área o autor do OA pode visualizar e organizar os elementos inseridos por meio dos menus apresentados. A Área de Trabalho é fundamental no sincronismo das mídias e composição das cenas. Na Figura 1.9 também é possível ver a lista de opções exibida ao clicar com o botão direito sob qualquer mídia da Área de Trabalho.

Figura 1.9. Área de trabalho da ferramenta da Cacuriá.

Camadas

Na área de trabalho da cena é possível empilhar itens de mídia fazendo com que uma apareça na frente da outra. Assim, o usuário pode configurar a disposição das mídias alterando suas camadas. Isso pode ser feito pelo menu de opções da mídia, apresentado na Figura 1.9, na Visão de Propriedades e ainda pela Visão de Biblioteca, como será explicado posteriormente.

Elos

A Cacuriá também oferece ao usuário a adição de elos, que possuem o efeito de iniciar uma cena em um determinado instante de tempo. O elo pode pode ser acionado quando a cena chega em um certo instante de tempo, os chamados elos automáticos, ou pela interação do usuário através do clique do mouse sob uma mídia, os chamados elos interativos.

Linha do tempo

Na ferramenta Cacuriá, a representação e a marcação do tempo são feitas pela Linha de Tempo, mostrada na Figura 1.10. Na Linha de tempo é possível visualizar e alterar os tempos em que a exibição dos elementos é iniciada e finalizada. Também é destacado quando os elementos sofreram alterações em suas propriedades e quando foram definidos elos automáticos na cena. Isso é feito pela exibição dos marcadores temporais, mostrados na Figura 1.11.

Na Visão da Linha de Tempo estão presentes o botão de *Play*, *Voltar* e *Avançar*. Com o botão *Play* é possível reproduzir e pausar a cena. Os botões Voltar e Avançar são utilizados para pular entre momentos de interação subsequentes.

Nessa visão ainda é possível editar o tempo total da cena e o tempo atual da cena. O tempo atual da cena também pode ser alterado pelo clique do mouse sob o *slider* e pelo clique sob um marcador temporal.

Figura 1.10. Linha de Tempo da ferramenta da Cacuriá.

Figura 1.11. Marcadores temporais.

Biblioteca de Mídias

As mídias inseridas na área de trabalho são representadas na Visão de Biblioteca, como mostra a Figura 1.12. Essa visão é composta pelas abas "Camadas", "Mídias da cena" e "Biblioteca". Na aba Camadas é possível visualizar e alterar a ordem das camadas em que as mídias são exibidas. Na aba Mídia da cena são mostradas as mídias que compõem a cena em foco. Na aba Biblioteca são exibidas todas as mídias que compõem o projeto.

(a) Aba camadas. (b) Aba mídias da cena. (c) Aba biblioteca.

Figura 1.12. Representação de mídias na Biblioteca do Cacuriá.

É possível alterar a ordem de camadas na Visão de Biblioteca pela aba "Camadas". Isso é feito atráves do clique sob os botões "Subir uma camada" e "Descer uma camada", mostrados na Figura 1.13. A aba "Mídias da cena" oferece a opção "Duplicar a mídia", que como o nome já diz duplica a mídia, que está em foco na biblioteca, na cena. O botão com essa funcionalidade é mostrado na Figura 1.13. A aba "Biblioteca", através do clique sob o botão "Importar mídia", possibilita carregar várias mídias para dentro do projeto. Depois de importadas, as mídias podem ser adicionadas na cena por meio de *drag-and-drop* ou através da opção "Inserir mídia", visível na Figura 1.13.

Figura 1.13. Botões da Visão de Biblioteca.

Propriedades de Mídias

Na Visão de Propriedades, destacada na Figura 1.14, é possível editar as propriedades da mídia em foco na Área de Trabalho. Para todas as mídias pode-se editar a altura, largura, posicionamento, transparência e sobreposição de camadas. Para mídias do tipo texto ainda é possível editar propriedades referentes à fonte e para mídias do tipo forma pode-se editar propriedades de preenchimento.

Figura 1.14. Propriedades editáveis de elementos no Cacuriá.

1.3.3. Utilizando a ferramenta

Nesta subseção é demonstrada a construção de objetos de aprendizagem utilizando a ferramenta Cacuriá. Durante este processo serão mostrados e explicados os principais recursos que a Cacuriá oferece.

Olá mundo em vídeo

O objetivo desta seção é demonstrar como são inseridas mídias na ferramenta. Primeiramente, é descrito como é adicionado um vídeo à cena. Para isso, o ícone "Vídeo" da barra de ferramentas deve ser clicado. Como consequência, será aberta uma janela de navegação onde o usuário deve indicar o vídeo que deseja inserir na área de trabalho. A Figura 1.15 ilustra essa sequência de etapas. Para inserir outras mídias basta clicar no ícone correspondente a mesma na barra de ferramentas.

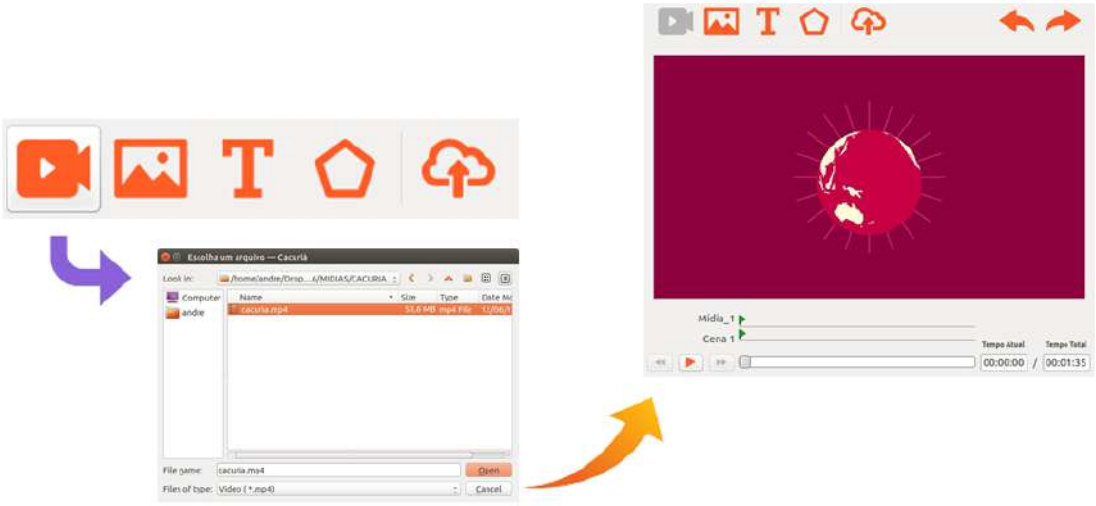


Figura 1.15. Inserção de vídeo na cena.

A Figura 1.16 mostra como fica a Área de Trabalho e a Linha de Tempo. Também é possível perceber que o ícone "Vídeo" fica desabilitado, já que só pode haver um vídeo por cena. Além disso, como consequência da adição do vídeo, o tempo total recebe como valor a duração do vídeo inserido na cena. É importante destacar que no momento que a cena é executada todas as opções são desabilitadas.

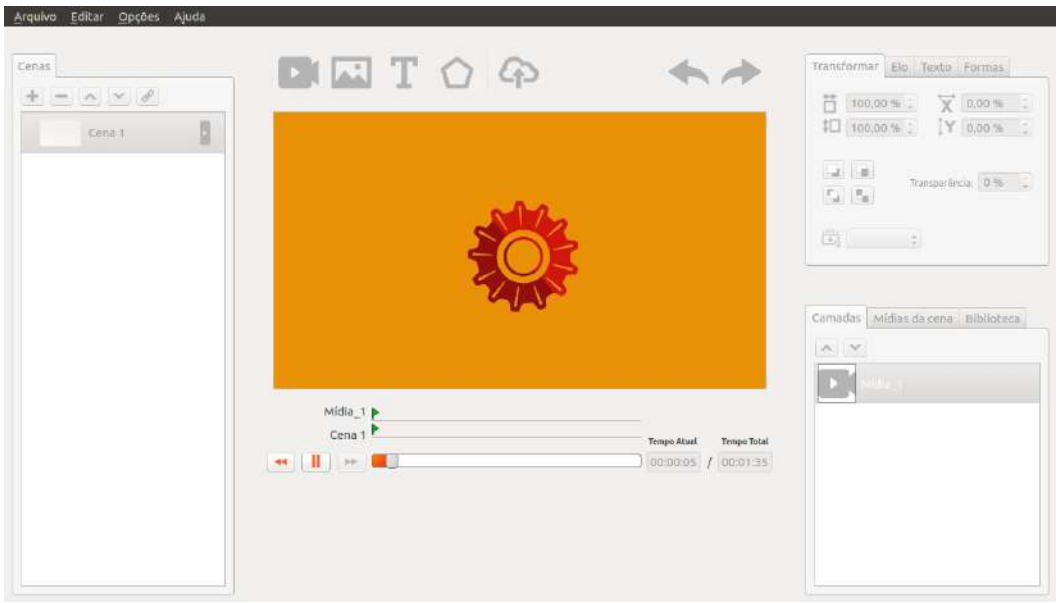


Figura 1.16. Interface da Cacuriá durante a execução da cena.

Como agora a cena possui um tempo total maior do que zero, é possível exibir a cena. Isso é feito clicando-se no ícone *Play* da linha temporal, como mostra a Figura 1.17.

Figura 1.17. Botão *Play* da linha temporal.

Sincronismo de imagens na *timeline*

Para visualizar como é feito sincronismo na ferramenta, pode-se executar as etapas descritas a seguir. Primeiramente, cria-se uma cena com o tempo total de 10 segundos. Isso é feito editando o campo "Tempo Total" localizado na Visão Temporal. Na barra de ferramentas há um ícone "Imagem", que ao ser clicado abre uma janela de navegação onde o usuário deve indicar o local da imagem que deseja inserir na área de trabalho da ferramenta.Ao escolher uma mídia, a mesma passa a ser exibida no centro da área de trabalho. Em seguida, altera-se o tempo atual da cena para 3 segundos e reposiciona-se a imagem. O reposicionamento pode ser feito clicando-se com o botão esquerdo sob a imagem e em sequência arrastando-a. No Cacuriá, sempre que uma mídia tem alguma de suas propriedades alterada, é adicionado um marcador temporal sob a linha temporal, como mostra a Figura 1.18. Na imagem também é possível visualizar as alças de redimensionamento da imagem, localizadas nas suas extremidades. Todas as mídias do Cacuriá possuem as alças de redimensionamento. Assim, além de alterar o tamanho das mídias através da Visão de Propriedades, o usuário também pode fazer essa alteração pro meio das alças de redimensionamento.

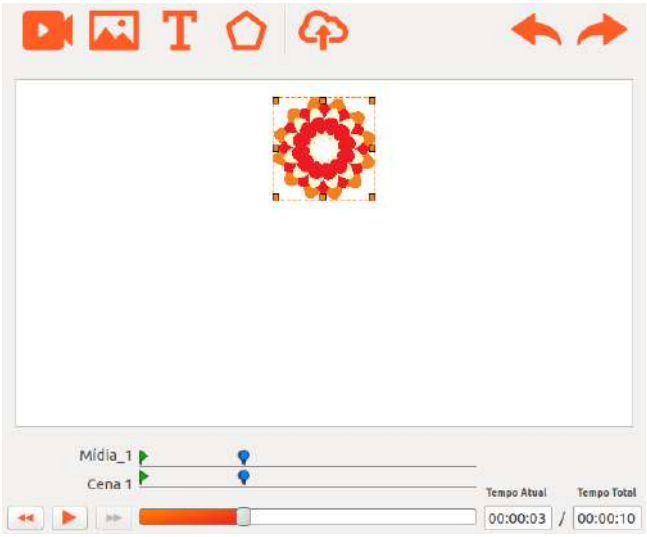


Figura 1.18. Cena com imagem reposicionada.

Continuando o exemplo, é definido o fim da exibição da imagem no instante 8 segundos. Isso pode ser feito arrastando o cursor temporal até os 8 segundos ou editando o "Tempo Atual". Em seguida, clica-se com o botão direito sob a imagem e selecione a opção "Definir Fim de Exibição". A Figura 1.19 mostra essa alteração.

Figura 1.19. Definido fim de exibição da mídia.

Ao realizar a definição, a imagem não será mais exibida na área de trabalho a partir do tempo 8 segundos. Outra consequência é adição do marcador na linha temporal. A área de trabalho resultante dessa operação pode ser vista na Figura 1.20.

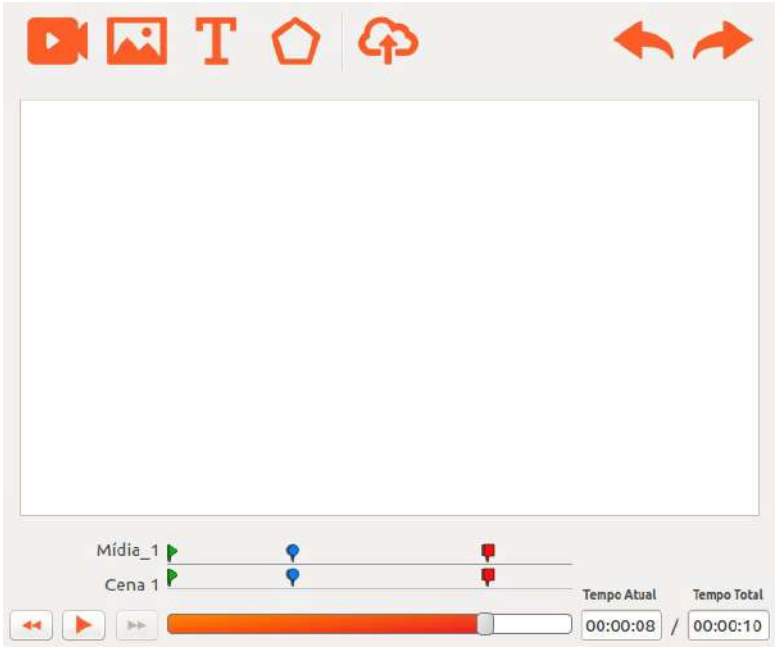


Figura 1.20. Área de trabalho após ser definido o fim de exibição.

No Cacuriá também é possível substituir a fonte das mídias do tipo imagem e vídeo. Isto é, pode-se trocar a mídia mantendo as propriedades que já haviam sido definidas na ferramenta. Para realizar essa ação é preciso apertar com botão direito do mouse sob a imagem e selecionar a opção “Trocar mídia”, como mostra a Figura 1.21. Como

consequência, a imagem é substituída por uma nova. Contudo, os marcadores temporais são mantidos. Ao executar a cena, será perceptível que as ações realizadas (reposicionamento e definição do fim de exibição) na mídia anterior estão atribuídas para nova imagem. A aba de Formas, mostrada na Figura permite que o usuário edite as propriedades de preenchimento e contorno da mídia em fica.

Figura 1.21. Funcionalidade "Trocar mídia".

Inserindo formas e texto na cena

Para inserir um texto é necessário selecionar o ícone "Texto" na barra de ferramentas. Ao ser clicado, será aberta uma caixa de texto e na Visão de Propriedades, a aba "Texto" ficará em foco. Essa configuração é mostrada na Figura 1.22. Na aba especifica de texto, é possível alterar a família, o tamanho e a cor da fonte. Também pode-se configurar o alinhamento do texto e aplicar as características negrito, itálico e sublinhado.

Figura 1.22. Inserção de texto.

A ferramenta Cacuriá oferece uma série de formas básicas pré-definidas. Para adicioná-las a área de trabalho é preciso selecionar o ícone "Forma" na barra de ferramentas.

Ao clicar no ícone é aberto um seletor de figuras geométricas onde o usuário deve escolher a forma desejada, o que é mostrado na Figura 1.23. Sempre que uma forma é selecionada na área de trabalho, a aba com as propriedades específicas de formas fica em evidência.

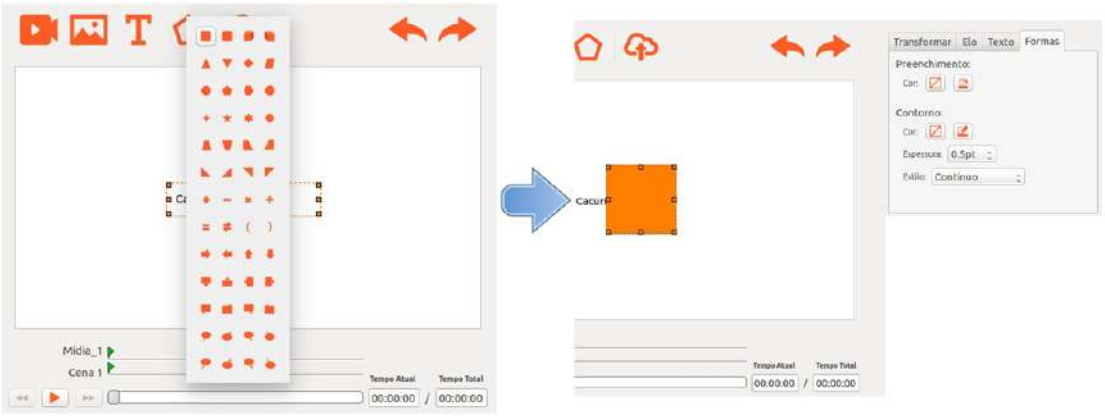


Figura 1.23. Formas disponíveis na ferramenta Cacuriá.

As formas na ferramenta Cacuriá também suportam a adição de texto. Para inserir texto em uma forma presente na área de trabalho, basta clicar duas vezes sob a mesma. O texto inserido pode ser editado na aba "Texto" na Visão de Propriedades. E assim como é inserido, para editar o texto contido na forma, clica-se duas vezes sob a mesma.

Sobreposição de mídias

Como citado anteriormente, as mídias podem ser empilhadas na área de trabalho, fazendo que sejam sobrepostas. Dessa forma, um recurso importante é a alteração de camadas. Para ilustrar uma forma de realizar essa ação, considera-se a disposição de duas mídias, um texto e uma forma, mostradas na Figura 1.24.

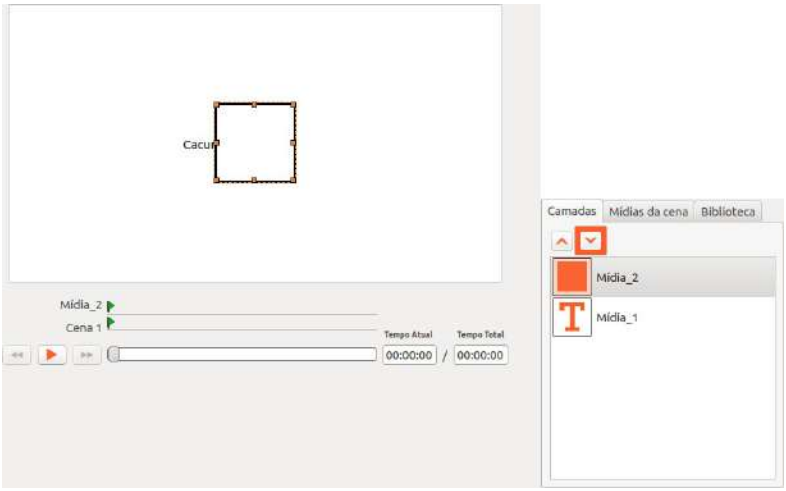


Figura 1.24. Sobreposição de mídias.

Na Figura 1.24, percebe-se que parte do texto não é visível porque a a forma o

sobrepõe. Essa sobreposição também é percebida na biblioteca. Assim, para exibir o texto totalmente é preciso descer a camada da forma ou subir a camada do texto. Após esse comando, o texto é exibido por completo, como mostra a Figura 1.25. Observe que a nova configuração das camadas da cena também alterou a representação na biblioteca.

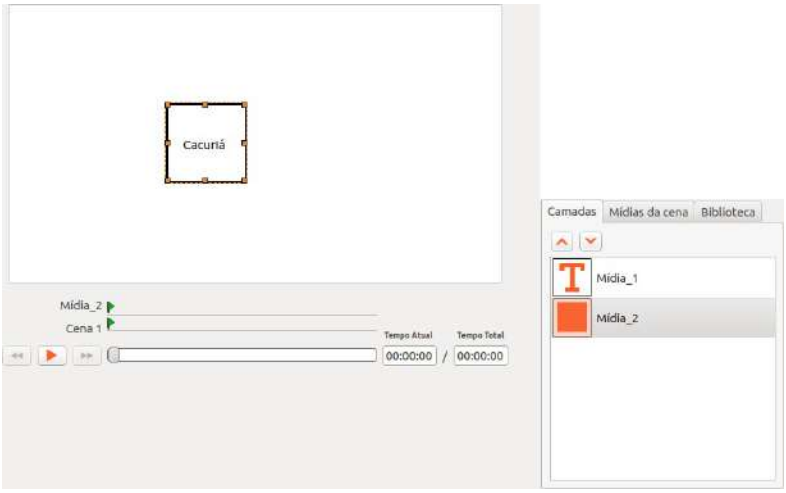


Figura 1.25. Área de trabalho após alteração da camada.

Utilizando layouts

A ferramenta Cacuriá também oferece layouts que contêm áreas, chamadas *slots*, que podem ser preenchidas pelo usuário. Os layouts reservam áreas na área de trabalho e quando o usuário seleciona uma mídia para preenche-la, a mídia inserida assume todas as propriedades do *slot*, isto é, tamanho, posicionamento e ordem de camada.

Em cada cena só pode ser aplicado um layout. Para utilizá-lo, o usuário de abrir o "Menu de Layout" e clicar no que deseja aplicar. O "Menu de Layout" está localizado na lista de cenas, como mostra a Figura 1.26.

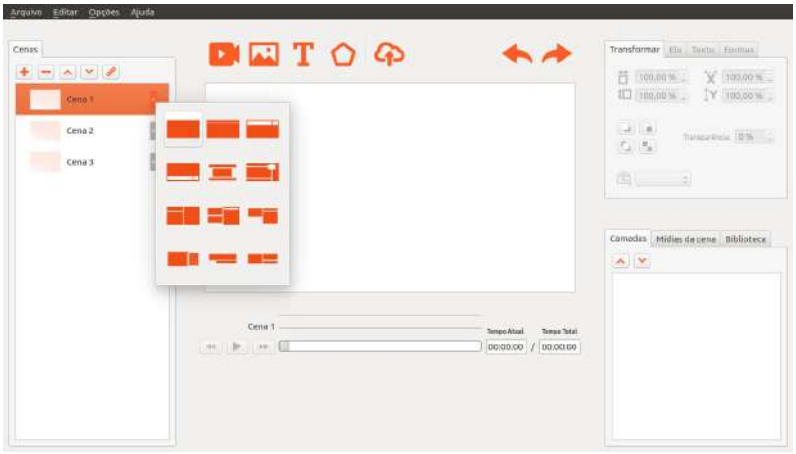


Figura 1.26. Menu de layouts.

Na Figura é mostrada a área de trabalho da cena após a inserção do primeiro layout

do menu. Observe que esse layout possui apenas um *slot* que ocupa toda a área de trabalho. No centro de qualquer *slot* encontram-se quatro botões que representam as mídias que podem preencher aquele *slot*.

Figura 1.27. Cena com layout aplicado.

Quando o usuário selecionar o botão de vídeo ou de imagem no *slot*, será aberta uma janela para que seja escolhida a mídia. Esse processo é similar ao adicionar uma dessas mídias pelos botões do menu. A Figura 1.28 mostra como fica a área de trabalho da cena após a inserção de uma mídia no *slot*. Perceba que o *slot* deixa de ser visível e a mídia passa a ocupar a área correspondente ao mesmo.

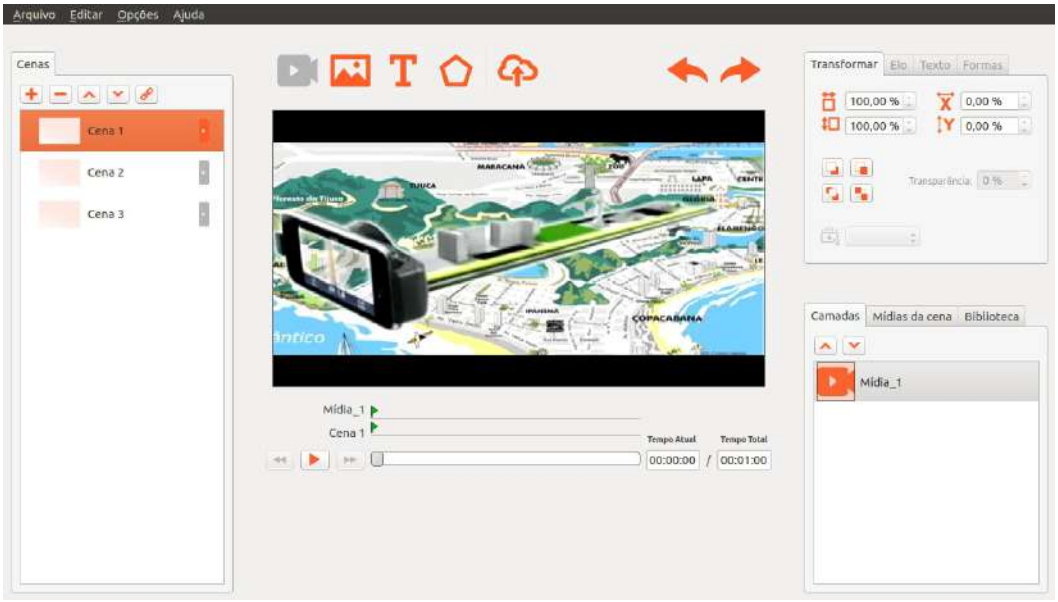


Figura 1.28. Cena com layout preenchido.

Sequências de cenas

Para ilustrar como uma sequência de cenas pode ser executada, é criado um projeto com três vídeos que serão exibidos em sequência. Uma forma de fazer isso é descrita a seguir.

Como o Cacuriá restringe um vídeo por cena, é necessário criar no projeto mais duas cenas. Lembrando que um novo projeto no Cacuriá já possui uma cena incorporada. A adição das cenas é feita clicando no botão “Adicionar cena” localizado na Visão de Cenas. A Figura 1.29 ilustra o processo de adição.

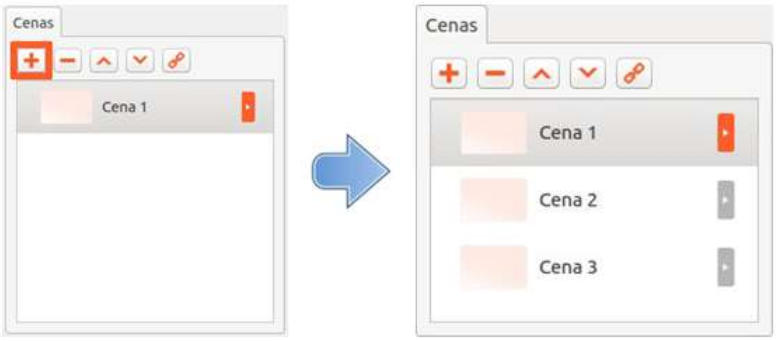


Figura 1.29. Adição de duas cenas ao projeto.

O próximo passo é inserir um vídeo em cada uma das cenas. Feito isso, os vídeos são adicionados às áreas de trabalhos das cenas, o que pode ser visto na Figura 1.30.

(a) Cena 1. (b) Cena 2. (c) Cena 3.

Figura 1.30. Vídeos nas cenas do projeto.

Para exibir os vídeos em sequência é necessário utilizar o recurso de elos automáticos. Os elos automáticos permitem que sejam executadas transições entre uma ou mais cenas. Uma forma de criar um elo automático é clicando na opção “Criar elo automático entre cenas”, localizado na Visão de Cenas.

A primeira ação para executar o vídeo em sequência, é configurar para que com o fim do primeiro vídeo seja exibido o segundo. Assim, é determinado que quando a “Cena 1” atingir seu fim, que no exemplo em questão acontece com 1 minuto de execução, deverá ser iniciada a execução da “Cena 2” a partir do seu início, ou seja, no instante 00:00. Esse definição é mostrada na Figura 1.31.

Figura 1.31. Configuração do elo automático entre a “Cena 1” e a “Cena 2”.

As marcações de elo automático são exibidas tanto na lista de cenas quanto na linha temporal, como mostra as Figuras 1.32 e 1.33, respectivamente.

Figura 1.32. Exibição do elo automático na Visão de Cenas.

Figura 1.33. Exibição do elo automático na Visão Temporal.

Outra forma de definir elos automáticos entre as cenas, é clicando com o botão direito sob a cena e escolhendo essa opção no menu. A configuração de elo entre a “Cena 2” e a “Cena 3” é feita dessa forma e ilustrada na Figura 1.34. Essa definição faz com que ao fim da exibição do segundo vídeo seja iniciada a exibição do terceiro. Com isso, tem-se a exibição dos três vídeos em sequência.

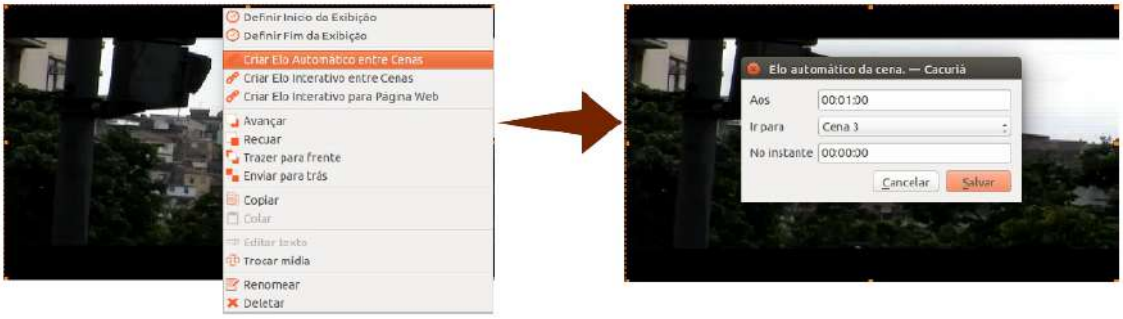


Figura 1.34. Configuração do elo automático entre a "Cena 2" e a "Cena 3".

Utilizando o atributo foco

É possível definir um atributo de foco para as mídias. Esse característica determina a ação que é executada na mídia quando esta estiver em foco durante quando a cena é executada. O usuário pode escolher entre as seguintes opções: "aumentar", "encolher", "subir", "cair", "recurar" e "avançar". O atributo "Foco" está localizado na Visão de Propriedades. A utilização desse recurso é mostrada na Figura 1.35.

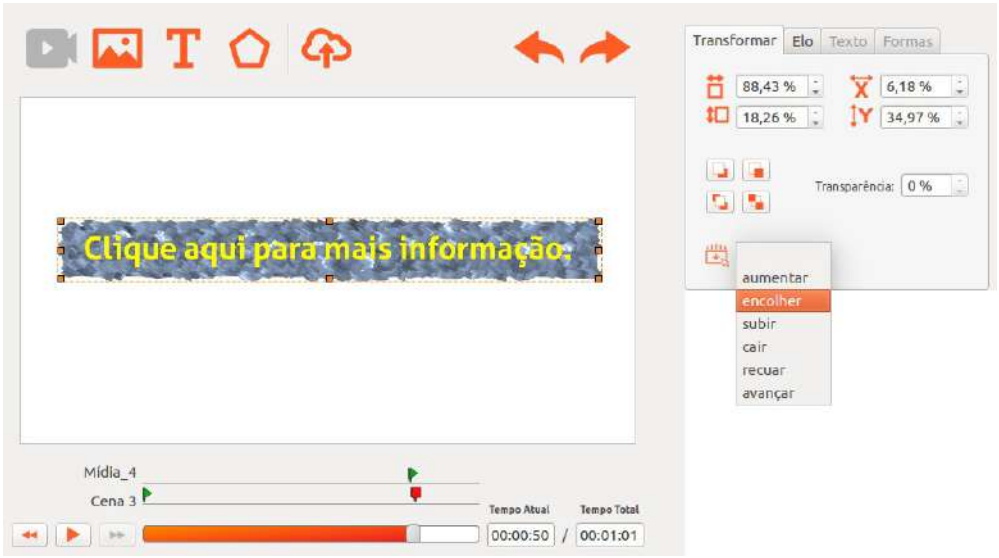


Figura 1.35. Definição de foco para uma imagem.

Diante dessa configuração, quando o espectador tiver assistindo a esse OA e posicionar o mouse sob a imagem, o mesmo perceberá a imagem encolher. Ao mover o mouse para outra área, fora da imagem, a mesma voltará a ser exibida normalmente.

Elo interativo para Página Web

Outro recurso de grande utilidade é a possibilidade de definir elos para páginas Web usando as mídias. Ao inserir um elo para Web, quando a mídia for clicada, a página será aberta no *browser* padrão do usuário.

A criação de um elo para página Web pode ser feito de suas formas: a primeira,

pela Visão de Propriedades e a segunda, pelo menu de opções que aberto quando se clica com o botão direito sob a mídia. A Figura 1.36 mostra a criação utilizando a segunda forma.

Figura 1.36. Criação de um elo para página Web.

Interatividade

Uma das principais vantagens da ferramenta Cacuriá é a possibilidade de criar OAs não-lineares. Ou seja, em certo momento do OA o usuário pode seguir caminhos diferentes. Para ilustrar a utilização desse recurso, a seguir será descrito o passo-a-passo da autoria de um OA não-linear.

O OA que será construído consiste em um vídeo interativo de um professor apresentando uma introdução à algoritmos de ordenação. Em um determinado momento, o professor questiona o aluno sobre qual algoritmo, *insertsort* ou *quicksort*, ele quer assistir a explicação. O aluno pode, então, clicar na figura que representa o algoritmo escolhido. Como consequência, o vídeo corrente é redirecionado para o vídeo com a explicação referente ao algoritmo escolhido pelo aluno.

No OA serão usadas três cenas: a primeira terá o vídeo introdutório, a segunda terá o vídeo com a explicação do algoritmo *insertsort* e a terceira terá o vídeo com a explicação do algoritmo *quicksort*. A Figura 1.37 mostra a interface após a inserção dos três vídeos nas suas respectivas cenas.

(a) Cena Introdução. (b) Cena Insertsort. (c) Cena Quicksort.

Figura 1.37. Representação de mídias na Biblioteca do Cacuriá.

Aos 40 segundos do vídeo de introdução, o professor questiona ao aluno sobre qual algoritmo ele quer ver a explicação. Então, nesse momento devem ser inseridas na

primeira cena as imagens que representam essas opções. A Figura 1.38 mostra como fica a área de trabalho depois das imagens serem adicionadas.

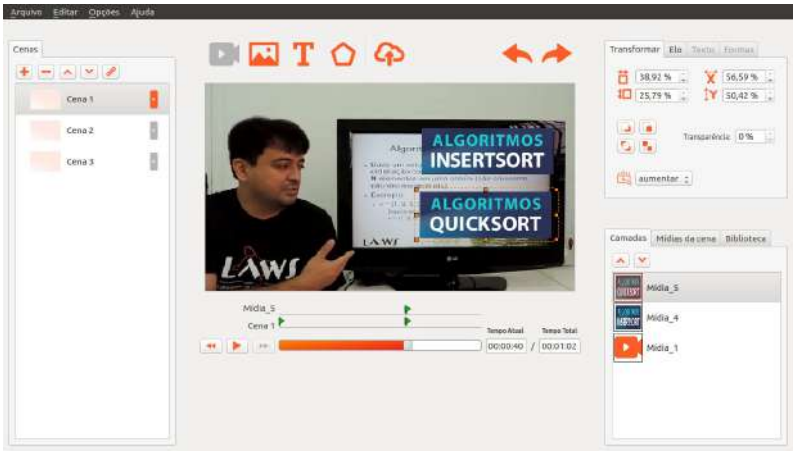


Figura 1.38. Primeira cena após a adição das imagens.

Agora, é necessário configurar as interações a partir das imagens. Para isso, a imagem do algoritmo *insertsort* deve ser selecionada, em seguida altera-se a propriedade "Ir para Cena" localizado na aba "Elo" da Visão de Propriedades, como mostra a Figura .

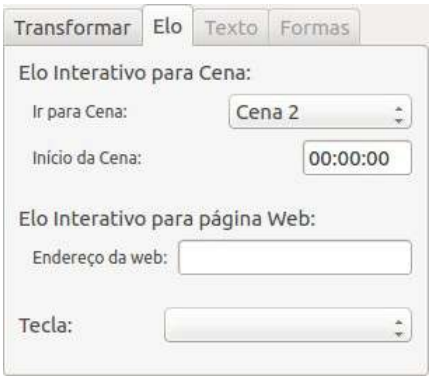


Figura 1.39. Definição de elo interativo pela Visão de Propriedades.

Outra forma de definir um elo interativo é por meio do menu de opções exibido quando se clica sob a mídia com o botão direito. É dessa maneira que é configurado o elo interativo na imagem que representa o algoritmo *quicksort*. Esse processo pode ser visto na Figura 1.40.

Figura 1.40. Definição de elo interativo.

Com isso, a construção do OA está finalizada. Ao assisti-lo, o usuário navegará entre os vídeos clicando sob as imagens que representam as opções para as explicações dos algoritmos.

Exportando o OA para HTML5

Para exportar um OA é utilizado o "Menu Arquivo". Nele, o usuário deve ir até a opção "Exportar" e em seguida escolher a opção HTML ou NCL, como mostra a Figura 1.41. Após o projeto ter sido exportado com sucesso para HTML, é possível executá-lo no *browser*. Basta ir até a pasta do projeto e clicar duas vezes sob o arquivo com o nome do seu projeto com a extensão .html.

Figura 1.41. Exportação do projeto.

Publicação do OA

A publicação do OA é feita clicando na opção "Publicar" na barra de ferramentas. Em seguida é exibida uma janela com a confirmação e o *browser* nativo abrirá na página do VIDEOAULA@RNP. Essa etapas são mostradas na Figura 1.42.

Figura 1.42. Publicação do projeto.

1.4. Conclusão

Este minicurso apresenta uma discussão sobre o conceito de objetos de aprendizagem multimídia e os detalhes que envolvem sua autoria. Espera-se que o participante termine o curso apto tanto a planejar quanto a desenvolver OAs com recursos avançados de interatividade.

Quanto ao planejamento de novos OAs, apresenta-se um roteiro que inicia pela matriz disciplinar que se deseja ensinar. A partir de tal documento, o conteúdo é decomposto em tópicos, cada um deles que se sugere produzir vídeos curtos de apoio de poucos minutos. Sugere-se, ainda, que se organize em uma tabela própria as relações entre os diversos tópicos de tal forma a construir *hiperlinks* entre eles. O planejamento é concluído com um *storyboard* que organiza a forma como os vídeos e o material adicional multimídia deve ser produzido.

O minicurso apresenta, também, um intenso treinamento na ferramenta de autoria multimídia Cacuriá, que permite sair do planejamento ao desenvolvimento de novos OAs mesmo sem conhecimento prévio de programação ou desenvolvimento de *software*. O minicurso compreende da instalação, à exemplificação dos diversos recursos da ferramenta, até a publicação do conteúdo gerado.

Enfim, acredita-se que, por meio deste minicurso, seus participantes possam ser capazes de criar novos OAs diretamente. A Figura 1.43 exemplifica 32 diferentes OAs criados por ex-alunos de versões anteriores deste minicurso, de diferentes tipos de conteúdo e oriundos de diversas organizações.

Figura 1.43. Mosaico de OAs criados com a ferramenta Cacuriá

Referências

Brecht, H. D. (2012). Learning from online video lectures. *Journal of Information Technology Education*, 11(1):227–250.

Busson, A. J. G., Damasceno, A. L. d. B., Azevedo, R. G. d. A., Neto, C. d. S. S., Lima, T. d. S., e Colcher, S. (2017). A hypervideo model for learning objects. In *Proceedings of the 28th ACM Conference on Hypertext and Social Media*, pages 245–253. ACM.

Busson, A. J. G., Damasceno, A. L. d. B., Lima, T. d. S., e Soares Neto, C. d. S. (2016). Scenesync: A hypermedia authoring language for temporal synchronism of learning objects. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web*, pages 175–182. ACM.

Damasceno, A. L., Galabo, R. J., e Soares Neto, C. S. (2014a). Cacuriá: authoring tool for multimedia learning objects. In *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*, pages 59–66. ACM.

Damasceno, A. L. d. B., Busson, A. J. G., e Neto, C. d. S. S. (2015). Voa : Vídeos como objetos de aprendizagem.

Damasceno, A. L. d. B., de Sousa Lima, T., Neto, C. d. S. S., e others (2014b). Cacuriá: Uma ferramenta de autoria multimídia para objetos de aprendizagem. In *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, volume 3, page 76.

Neto, C. d. S. S., de Brandão, A. L., e Grandson, A. J. (2015). Videocolaboração para conteúdo multimídia educacional: Visão de futuro do laws/ufma. *I RNP CT-Video Workshop / XXI Brazilian Symposium on Multimedia and the Web*.

Paternò, F. (2013). End user development: Survey of an emerging field for empowering people. *ISRN Software Engineering*, 2013.

Polsani, P. R. (2006). Use and abuse of reusable learning objects. *Journal of Digital information*, 3(4).

Portela, S. L. C. (2017). Objetos de aprendizagem para o ensino de algoritmos. Monografia (Graduação) Curso de Ciência da Computação), UFMA (Universidade da Federal do MAranhão), São Luís, Brazil.

DSc. Carlos de Salles Soares Neto Possui graduação em Ciência da Computação pela Universidade Federal do Maranhão (2000), mestrado e doutorado em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2003 e 2010). Atualmente é professor adjunto da Universidade Federal do Maranhão onde atua como coordenador do LAWS (Laboratório de Sistemas Web Avançados) e pesquisador associado do Laboratório TeleMídia da PUC-Rio. Tem experiência na área de Ciência da Computação, com ênfase em Hipermídia, atuando principalmente nos seguintes temas: aplicações multimídia, TV digital e modelos de documentos multimídia.

Thacyla de Sousa Lima é mestranda em Ciência da Computação sob orientação do Prof. Carlos de Salles Soares Neto na Universidade Federal do Maranhão (UFMA) e é bacharela (2016) em Ciência da Computação pela UFMA. É pesquisadora no laboratório TeleMídia – UFMA. Tem experiência na área de sistemas multimídia, trabalhando principalmente com modelos de documento e autoria hipermídia. Participou como pesquisadora no programa de GTs da RNP (2013-2015) onde desenvolveu Cacuriá, uma ferramenta de autoria para criar objetos de aprendizagem, que resultou em publicações em conferências e simpósios tais como CBIE 2014, WebMedia 2015 e 2016, e ACM Hypertext 2017.

MSc. André Damasceno é doutorando em Informática na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) sob orientação da Profa. Simone Diniz Junqueira Barbosa (PUC-Rio). Possui graduação (2013) e mestrado (2015) em Ciência da Computação pela Universidade Federal do Maranhão (UFMA). É pesquisador do laboratório TeleMídia/PUC-Rio e colaborador do Laboratory of Advanced Web Systems (LAWS) – UFMA. Seus interesses de pesquisa são sistemas multimídia e ciência de dados, trabalhando principalmente nos tópicos: modelo de documentos hipermídia, autoria e aplicações para Web e iDTV e análises visuais. Atualmente, está trabalhando no projeto oficial de desenvolvimento Ginga Middleware, que é o middleware do sistema nipo-brasileiro de TV Digital (ISDB-TB) e recomendação ITU-T H.761 para serviços IPTV. Participou como pesquisador no programa de GTs da RNP (2012-2015) onde desenvolveu Cacuriá, uma ferramenta de autoria para criar objetos de aprendizagem, que resultou em publicações em alguns congressos tais como CBIE 2014, SBGames 2014, UbuntuNet-Connect 2014 e 2015, WebMedia 2014, 2015 e 2016, TNC 2015, HCI 2017 e ACM Hypertext 2017.

MSc. Antonio Busson é doutorando em Informática na Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio) sob orientação do Prof. Sergio Colcher. Possui graduação (2013) e mestrado (2015) em Ciência da Computação pela Universidade Federal do Maranhão (UFMA). É pesquisador do laboratório TeleMídia/PUC-Rio e colaborador do Laboratory of Advanced Web Systems (LAWS) – UFMA. Seus interesses de pesquisa são sistemas multimídia, trabalhando principalmente nos seguintes tópicos: codificação, processamento e transmissão de conteúdo multimídia; Modelos de documentos hipermídia e aplicativos como Web, iDTV e Jogos. Atualmente, está trabalhando no projeto oficial de desenvolvimento Ginga Middleware, que é o middleware do sistema nipo-brasileiro de TV Digital (ISDB-TB) e recomendação ITU-T H.761 para serviços IPTV. Também participou do desenvolvimento do GT-VOA em parceria com a RNP, a Rede Nacional de Ensino e Pesquisa, que foi desenvolvido no do programa de grupos de trabalho da RNP, durante os ciclos de 2012–2013, 2013–2014, e 2015. O trabalho resultou em publicações em conferências e simpósios como CBIE 2014, SBGames 2016, WebMedia 2015 e 2016, e ACM Hypertext 2017.

Capítulo 2

Cross-platform Multimedia Application Development for Mobile, Web, Embedded and IoT with Qt/QML

Manoel C. M. Neto[1], Sandro S. Andrade[1] e Renato L. Novais[1]

[1]Grupo de Sistemas Distribuídos, Otimização, Redes e Tempo-Real – GSORT
Instituto Federal da Bahia (IFBA)
R. Emídio dos Santos, s/n
Barbalho, Salvador – Ba – Brasil, 40301-015

{manoelnetom,renato,sandroandrade}@ifba.edu.br

Abstract. *Qt is cross-platform development toolkit that enables the quick and cost-effective design, development, deployment, and maintainance of software for multiple platforms while delivering a seamless user experience across a range of devices. Over the past years, Qt has been successfully enabling advanced multimedia applications through a quite simple declarative language named QML (Qt Modeling Language). QML provides a set of high-level components for handling audio and video, animations, maps, cameras, and sensors, just to mention a few. Qt is widely used by several companies around the world, but its adoption in Brazil has been hindered by different factors. This short-course aims at presenting the main features Qt/QML provides regarding the development of cross-platform multimedia applications.*

Abstract. *[Resumo] O Qt é um toolkit para desenvolvimento multiplataforma de aplicações que viabiliza o desenvolvimento, implantação e manutenção de soluções de software de forma rápida e com um baixo custo. Ao longo do últimos anos, o Qt tem sido utilizado com sucesso no desenvolvimento de aplicações multimídia avançadas através de uma linguagem declarativa simples, denominada QML (Qt Modeling Language). A QML disponibiliza um conjunto de componentes de alto nível para manipulação de áudio e vídeo, animações, mapas, câmeras e sensores, somente para citar alguns. O Qt é amplamente usado no mundo inteiro mas seu uso no Brasil tem sido dificultado por uma série de fatores. Este minicurso tem como objetivo apresentar as principais funcionalidades do Qt/QML voltadas ao desenvolvimento de aplicações multimídia multiplataforma.*

2.1. Introdução

Ambientes multimídia interativos são aqueles nos quais a computação é usada para melhorar de forma imperceptível as atividades comuns do dia a dia [Neto 2015]. Uma das forças motrizes do interesse emergente nestes ambientes é tornar os computadores não apenas verdadeiramente amigáveis ao usuário, mas também essencialmente invisíveis para ele. Um ponto importante para alcançar este objetivo é a computação Ubíqua (UBiComp) [Krumm 2009].

A UbiComp, em seus vários desdobramentos e aplicações, é considerada por muitos como o novo paradigma da computação para o século XXI. Ela é a área da computação que estuda o acoplamento do mundo físico ao mundo da informação e fornece uma abundância de serviços e aplicações, permitindo que usuários, máquinas, dados e objetos do espaço físico interajam uns com os outros de forma transparente. Uma das dificuldades do desenvolvimento de aplicações para ambientes ubíquos é lidar com a diversidade de plataformas necessárias para o correto funcionamento desses sistemas. Um efeito colateral dessa dificuldade é o retrabalho necessário para adaptar a mesma aplicação para cada uma das possíveis plataformas. Neste contexto, o *framework* Qt [Russo and Vytiniotis 2009, The Qt Company 2016] surge como uma alternativa importante.

O Qt é um *toolkit* para desenvolvimento de interfaces gráficas de usuário (GUI) multiplataforma [Summerfield 2007], aplicações de comunicação via rede, suporte ao processamento de documentos XML, bancos de dados, multimídia, aplicações embarcadas, etc. Ele tem sido usado por diferentes aplicações (comerciais e *open-source*) desde 1995 em companhias e organizações como: Adobe, Boeing, IBM, Motorola, NASA, Skype. Uma contribuição importante do Qt para o desenvolvimento de aplicações focadas nas interfaces de usuários é a ***Qt Modeling Language*** (QML). A QML é uma linguagem cujos documentos descrevem uma árvore de elementos. Esses elementos podem ser combinados para construir desde componentes simples como botões até aplicações mais complexas.

Este artigo representa o texto base do minicurso apresentado no XXIII Simpósio Brasileiro de Sistemas Multimídia e Web – WebMedia. O curso é motivado no interesse de pessoas e organizações por temas como *Internet of Things* (IoT), sistemas embarcados e dispositivos móveis. Esta é uma tendência cujas aplicações já ganham espaço em companhias, cidades e na vida cotidiana de muitas pessoas. O minicurso apresenta as primeiras noções de como desenvolver uma aplicação multimídia multiplataforma através de exemplos práticos que usam funcionalidades do Qt/QML. Ele apresenta a construção de elementos encontrados em muitas aplicações comuns em ambientes multimídia interativos. Estes conceitos introdutórios são fundamentos importantes para diversas aplicações em diversas áreas.

2.2. Visão Geral

Um amplo conjunto de soluções para o desenvolvimento de software com Android, iOS, embarcados e outras plataformas está atualmente disponível no mercado. Desenvolver aplicações que executem não só no Android/iOS, mas também em um smartWatch ou smartTV é um dos desafios atuais da Computação, que geralmente implicam no desenvolvimento de duas ou mais soluções totalmente diferentes.

O Qt é um *toolkit* moderno para criação de aplicações gráficas e uma das poucas soluções atualmente disponíveis para desenvolvimento multiplataforma e *multi-device*. Ele pode ser utilizado não só para o Android e iOS, mas também para o Linux, Windows, macOS e uma série de plataformas embarcadas, sem a necessidade de duplicação de trabalho. O desenvolvimento do *toolkit* foi iniciado em 1995. Desde então, ele evoluiu e hoje provê uma base sólida para muitas aplicações [Blanchette and Summerfield 2006]. Este curso foca no Qt 5.9, versão mais atual do *framework*.

O suporte ao desenvolvimento multiplataforma do Qt é realizado através da utilização de uma arquitetura em camadas, onde aspectos particulares de uma determinada plataforma são tratados em um componente denominado *Qt Platform Abstraction* (QPA)[1]. Cada plataforma suportada pelo Qt possui uma diferente implementação do QPA, disponibilizando as funções básicas de renderização gráfica e outros serviços. Com isso, uma mesma API é mantida para todas as camadas acima da QPA, viabilizando a compilação do código-fonte e execução da aplicação em múltiplas plataformas. Todos estes aspectos são transparentes para o desenvolvedor usuário do Qt.

Outro aspecto importante do Qt são as extensões que o *toolkit* oferece à linguagem de programação C++. Tais extensões têm como objetivo minimizar uma série de problemas e deficiências característicos do desenvolvimento com C++, tais como *memory leaks*, ausência de reflexão computacional, objetos *composite* e comunicação baseada em eventos (via *signals* e *slots*). Tais extensões são tratadas pelo *meta-object compiler* (`moc`) – um pré-processador que converte código-fonte escrito com as extensões do Qt para código C++ puro. Assim, aplicações/bibliotecas podem ser compiladas por qualquer compilador C++ padrão como o `clang` ou `g++`. O *toolkit* possui diversos módulos que cobrem muitos requisitos e funcionalidades. Neste artigo, serão apresentados algumas dessas funcionalidades, especialmente aquelas voltadas para o desenvolvimento de aplicações *mobile*, web, embarcadas e para IoT.

2.2.1. Ambiente de Desenvolvimento

O Qt é disponibilizado como um *Software Development Kit* (SDKs) completo e sua instalação pode ser realizada em uma série de plataformas[2]. O SDK é de fácil instalação e já inclui uma IDE (*Integrated Development Environment*) denominada Qt Creator. O Qt Creator é um ambiente altamente produtivo para desenvolvimento Qt, recomendado para iniciantes e será o ambiente utilizado neste minicurso. A versão mais recente do SDK pode ser obtida do site `http://www.qt.io`. É importante ressaltar que o Qt também pode ser usado a partir de linhas de comando. Isso abre a possibilidade de usar outros editores de código no processo de desenvolvimento.

Para testar a instalação, abra o Qt Creator, crie um novo projeto do tipo *Qt Quick Controls 2* (*File → New File or Project → Qt Quick Controls 2 Application*) e nomeie-o `HelloWorld`. Depois disso, o Qt Creator irá criar vários arquivos automaticamente tais como o `HelloWorld.pro` e o `main.qml`. Arquivos `.pro` são usados para armazenar e gerenciar configurações relevantes do projeto. Os arquivos `.qml` armazenam o código-fonte de uma aplicação escrita em QML. Usando o QtCreator, copie e cole o código a

[1] `http://doc.qt.io/qt-5/qpa.html`
[2] `http://doc.qt.io/qt-5/supported-platforms.html`

seguir (Listagem 2.1) no arquivo main.qml e depois disso, selecione *Build* → *Run* no menu.

Listagem 2.1. Exemplo básico de QML

```
1  import QtQuick 2.6
2  Rectangle {
3      width: 360
4      height: 360
5      Text {
6          anchors.centerIn: parent
7          text: "Hello World"
8      }
9      MouseArea {
10         anchors.fill: parent
11         onClicked: Qt.quit();
12     }
13 }
```

2.3. Principais Módulos

Em função da grande quantidade de funcionalidades ofertadas, o Qt é dividido em módulos, possibilitando que o desenvolvedor indique quais partes do *toolkit* ele deseja utilizar. Um módulo é uma coleção de funcionalidades, disponibilizadas como APIs (*Application Programming Interfaces*) C++, QML ou ambos. Os módulos do Qt são divididos em dois grupos: *Qt Essentials* e *Qt Add-Ons*. Os módulos do grupo *Qt Essentials* contêm as funcionalidades mais comumente utilizadas, estão disponíveis em todas as plataformas suportadas e garantem compatibilidade binária entre todas as versões *major* do Qt (ex: toda a família de versões 5.x). Já os módulos do grupo *Qt Add-Ons* contêm funcionalidades relevantes apenas em casos mais específicos. As próximas seções apresentam alguns dos principais módulos do Qt usados em diferentes tipos de aplicações.

2.3.1. Qt Quick e Qt QML

O módulo `Qt QML` disponibiliza o interpretador (*engine*) QML e outros recursos de infraestrutura da linguagem. Já o módulo `Qt Quick` é responsável por disponibilizar todos os tipos básicos para criação de interfaces gráficas de usuário em QML. Estes módulos oferecem uma série de recursos: i) o interpretador QML; ii) um interpretador JavaScript; e iii) mecanismos para integração QML/C++. Eles permitem aos desenvolvedores de aplicativos tanto estender o QML com tipos personalizados quanto integrar o código QML com JavaScript e C++.

Assim como outras linguagens de marcação, um documento QML descreve uma árvore de objetos. Estes objetos incluem: um conjunto sofisticado de blocos de construção, gráficos (ex.: retângulo e imagem) e comportamentos (ex.: transição e animação) que podem ser combinados para criar componentes que variam em complexidade. Estes componentes podem ser botões simples e controles deslizantes, ou mesmo aplicativos móveis/embarcados muito mais complexos.

Nos documentos QML, cada objeto em uma árvore é especificado pelo tipo, seguido por um par de chaves. A Listagem 2.1 apresentou três objetos, um `Rectangle`

e seus dois filhos: `Text` e `MouseArea`. Entre as chaves, também podemos especificar atributos de um objeto. Cada objeto tem um atributo especial chamado `id`. Atribuir um `id` permite que um objeto seja referenciado por outros objetos.

O QML permite especificar o valor de um atributo de forma declarativa e isto é denominado *property binding*. Estes *bindings* são especificados como um par `atributo: valor`. Ao realizar um *binding*, o valor do atributo é atualizado automaticamente quando o valor for modificado. A Listagem 2.2 apresenta um exemplo de *binding*. O elemento `Rectangle` define a sua largura em função de `otherItem.width`, o que significa que ele irá possuir sempre o mesmo valor de largura quedo "otherItem".

Listagem 2.2. Exemplo de Binding

```
1 Rectangle {
2     function calculateMyHeight() {
3         return Math.max(otherItem.height, 300);
4     }
5     anchors.centerIn: parent
6     width: otherItem.width
7     height: calculateMyHeight()
8     color: (width > 10) ? "blue":"red"
9 }
```

A outra maneira de definir um *binding* é usar uma expressão JavaScript válida. Assim, os *bindings* podem acessar propriedades do objeto, fazer chamadas de função e até mesmo usar objetos JavaScript incorporados. No exemplo acima, um elemento `Rectangle` define sua própria altura ao se referir a uma função JavaScript chamada `calculateMyHeight`.

Outra característica importante do QML é o mecanismo para controlar mudanças de estado nas propriedades. Cada elemento possui um estado base "implícito". Cada mudança de estado é descrita listando as propriedades e valores dos elementos que diferem do estado base. No exemplo a seguir (Listagem 2.3), no estado base, myRect está posicionado no ponto (0, 0). No estado "moved", ele passa a estar posicionado no ponto (50, 50). Clicar dentro do `MouseArea` altera o estado para "moved", movendo assim o retângulo. O QML também suporta a animação de objetos. Para animar o exemplo acima, pode-se utilizar o elemento "transition".

Listagem 2.3. Exemplo de Binding e Animação

```
1  import QtQuick 2.6
2
3  Item {
4      id: myItem
5      width: 200; height: 200
6      Rectangle {
7          id: myRect
8          width: 100; height: 100
9          color: "red"
10     }
11     states: [
12         State {
```

```
13                name: "moved"
14                PropertyChanges {
15                    target: myRect
16                    x: 50; y: 50
17                }
18            }
19        ]
20        MouseArea {
21            anchors.fill: parent
22            onClicked: myItem.state = 'moved'
23        }
24        transitions: [
25            Transition {
26                from: "*"; to: "moved"
27                NumberAnimation {
28                    properties: "x,y"
29                    duration: 500
30                }
31            }
32        ]
33    }
```

2.3.2. Qt Multimedia

O módulo `Qt Multimedia` oferece uma série de funcionalidades relacionadas à manipulação de áudio, vídeos e outros artefatos multimídia. Ele fornece um rico conjunto de tipos QML e classes C++ para lidar com conteúdo multimídia além das APIs necessárias para suportar codificação e decodificação[3], acesso a recursos como câmera e microfone permitindo reproduzir/gravar som, vídeo ou tirar fotos.

Neste módulo, quatro elementos podem ser destacados: `MediaPlayer`, `Camera`, `VideoOutput` e `SoundEffect`. O elemento `MediaPlayer` permite controlar as funções relacionadas à reprodução de mídias (reproduzir, parar, pausar, etc). O elemento `VideoOutput` pode ser usado para reproduzir um vídeo. O elemento `Camera` é usado para capturar um fluxo de vídeo ao vivo. O elemento `SoundEffect` fornece uma maneira de reproduzir efeitos sonoros no QML.

Listagem 2.4. Exemplo de Funcionalidades de Multimídia

```
1  import QtQuick 2.5
2  import QtMultimedia 5.6
3
4  Item {
5      width: 1024
6      height: 600
7      MediaPlayer {
8          id: player
```

[3]É importante destacar que os recursos de decodificação e codificação são tratados através de `back-ends` específicos em cada plataforma (por exemplo, DirectShow usado no Windows ou QuickTime no OSX).

```
9           source: "trailer400p.ogg"
10      }
11      VideoOutput {
12          anchors.fill: parent
13          source: player
14      }
15      Component.onCompleted: {
16          player.play();
17      }
18  }
```

Na Listagem 2.19, um elemento `MediaPlayer`, denominado *player*, tem uma propriedade denominada *source* que aponta para o arquivo de vídeo trailer400p.ogg. A propriedade *source* de um elemento `VideoOutput` foi vinculada a *player*. Dessa forma, assim que o componente principal for completamente inicializado, a função `play()` do objeto *player* é chamada e a apresentação do vídeo é iniciada. O elemento `MediaPlayer` também possui outros atributos úteis. Por exemplo, os atributos de duração e posição podem ser usados para construir uma barra de progresso. Algumas outras operações básicas – como alterar o volume ao reproduzir a mídia – podem ser controladas através da propriedade de volume suportada por um elemento `MediaPlayer`.

Pode-se utilizar um elemento `Camera` para capturar imagens ou vídeos de uma câmera real e gerenciar as configurações de processamento aplicadas às imagens capturadas (Listagem 2.5). Por exemplo, para exibir um visualizador de imagens da câmera (*viewfinder*), pode-se combinar um `VideoOutput` com um `Camera` configurado como sua fonte. O elemento `Camera` contém um conjunto de sub-elementos que cobrem muitos recursos. Por exemplo, para capturar imagens, pode-se usar o elemento *Camera.imageCapture*. Quando o método de captura é chamado, uma foto é tirada. Isso significa que, primeiro o *Camera.imageCapture* emite um sinal *imagemCaptured* seguido de um sinal *imageSaved*. Se várias câmeras estiverem disponíveis, pode-se selecionar qual delas usar, vinculando o atributo *deviceId* a *QtMultimedia.availableCameras*. Em um dispositivo móvel, pode-se alternar convenientemente entre câmeras voltadas para a frente e para trás, definindo a propriedade da posição.

Listagem 2.5. Exemplo de uso da Câmera

```
1  VideoOutput {
2      anchors.fill: parent
3      source: camera
4  }
5
6  Camera { id: camera }
7
8  Button {
9      id: shotButton
10     text: "Take Photo"
11     onClicked: camera.imageCapture.capture();
12 }
```

Os elementos `SoundEffect` permitem reproduzir arquivos de áudio não com-

pactados (geralmente arquivos WAV) em um modo de latência geralmente menor e é adequado para sons de tipo *feedback* em resposta a ações do usuário (por exemplo, sons de teclado virtual, comentários para diálogos *popup* ou sons de jogos). Normalmente, o efeito sonoro deve ser reutilizado, o que permite que todas as análises e preparações sejam feitas antes do tempo e apenas sejam disparadas quando necessário. Isso é fácil de alcançar com o *property binding* do QML. O exemplo da Listagem 2.6 reproduz um arquivo WAV no clique do mouse.

Listagem 2.6. Exemplo de uso de Áudio

```
1  Text {
2      text: "Click Me!";
3      font.pointSize: 24;
4      width: 150; height: 50;
5
6      SoundEffect {
7          id: playSound
8          source: "soundeffect.wav"
9      }
10     MouseArea {
11         id: playArea
12         anchors.fill: parent
13         onPressed: playSound.play()
14     }
15 }
```

2.3.3. Qt Network

O Qt fornece classes para comunicação, integração com a web e para comunicação entre processos distribuídos através de uma rede de computadores. Essas classes estão agrupadas no módulo `Qt Network` que fornece uma camada de abstração para as operações de baixo nível e exibe apenas classes e funções de alto nível. O `Qt Network` também permite construir aplicações que usem protocolos de nível mais baixo, como TCP e UDP. Classes como `QTcpSocket` e `QUdpSocket` permitem que o desenvolvedor envie e receba mensagens usando estes protocolos. A resolução de nomes (DNS) – atividade comum em uma rede – é de responsabilidade da classe `QHostInfo`. Filtrar e redistribuir o tráfego de rede através de *proxies* pode ser tratado pela classe `QNetWorkProxy`.

O módulo `Qt Network` também fornece a API de *Bearer Management*. Esta API possui funções que podem iniciar ou interromper interfaces de rede e *roaming* entre pontos de acesso. No entanto, não ela não permite gerenciar as configurações de rede. Cada plataforma de cada aplicação deve assumir esta responsabilidade.

2.3.4. IoT e Qt

A quantidade de dispositivos e coisas que nos rodeiam estão aumentando rapidamente, tornando-se mais inteligentes e exigindo um software que funcione em uma maior variedade de hardwares que incluem dispositivos IoT com ou sem tela tai como: relógios inteligentes, televisores inteligentes, PCs, dentre outros. O Qt foi, desde sua concepção, projetado para criar interfaces de usuário e essa capacidade inata é necessária ao IoT.

Com o Qt, pode-se construir UIs simples ou complexas, clássicas ou modernas, padronizadas ou personalizadas. Porém, o mais importante no contexto da IoT é que tudo que é construído com o Qt é multiplataforma.

Outro benefício do Qt é o compartilhamento de software entre o *firmware* dos dispositivos embarcados e os aplicativos usados para controlá-los. A maioria das tecnologias usadas para RAD (*Rapid-Application Development*) obriga a usar, no ambiente de desenvolvimento, algo diferente do que está embarcado nos dispositivos. Isso significa, por exemplo, que podem haver duas instâncias do mesmo código, em duas linguagens, usando duas APIs, que precisam de dois conjuntos de testes e que podem gerar duas maneiras distintas de ocultar os erros. Com o Qt, o mesmo projeto, a mesma construção e os mesmos testes de um módulo serão compartilhados entre todos os locais em que o software será executado [Islam et al. 2015].

Para atender às demandas de IoT, o framework Qt oferece muitas APIs. O módulo `Qt Sensors`, por exemplo, possui dezenas de elementos e classes predefinidas que abstraem vários sensores existentes. Por exemplo, o elemento Accelerometer da QML é um *wrapper* da classe `QAccelerometer` e abstrai um acelerômetro real.

2.3.5. QtLite

Ao longo dos anos, o Qt foi usado em uma vasta gama de sistemas operacionais e dispositivos embarcados. Não demorou muito para que o Linux fosse tão importante para o Qt em embarcados quanto para *desktops*. Muitos outros sistemas operacionais embarcados seguiram essa tendência e o Qt suportou uma ampla gama de sistemas operacionais (Linux, Windows e vários sistemas operacionais de tempo-real). No entanto, para utilizar de forma eficiente o Qt nesses sistemas operacionais, e especialmente nos dispositivos embarcados, é necessário atender requisitos estritos de desempenho e consumo de memória. Muitas vezes é um desafio longo configurar o Qt para usar eficientemente os diferentes componentes de hardware, bibliotecas disponíveis ou retirar as partes do Qt que não são necessárias em um dispositivo embarcado.

A mais recente iniciativa Qt na área de IoT é o projeto QtLite. Ele tem como principal objetivo tornar o Qt uma estrutura muito mais direcionada, que facilite todo o ciclo de desenvolvimento e a vida útil dos produtos baseados em dispositivos embarcados. O QtLite disponibiliza um novo sistema de configuração que permite aos desenvolvedores definir as funcionalidades necessárias com mais detalhes, sem ter que incluir recursos desnecessários. O fluxo de trabalho de desenvolvimento também está sendo aprimorado para permitir que os desenvolvedores comecem com uma configuração mínima flexível que lhes permita adicionar recursos à medida que forem necessários.

2.4. Exemplo de Aplicativos Móveis QML

Nesta seção, serão apresentados exemplos de aplicativos móveis com um conjunto variado de componentes QML. Os aplicativos desenvolvidos contêm componentes básicos QML, conexão com banco de dados, utilização de serviços RESTful, uso de mídias e sensores. O código-fonte das aplicações está disponível em `http://github.com/GSORT/QML-Exemplos`.

2.4.1. Minhas Compras

Nesta seção, será apresentado um aplicativo que simula compras de produtos, focando em suas principais partes. O aplicativo faz uso de banco de dados (SQLite) e contém componentes básicos de um sistema em QML. A Figura 2.1 apresenta as telas do aplicativo.

Figura 2.1. Telas do aplicativo Minhas Compras

Para criar uma aplicação desse tipo, selecione *File* → *New File or Projects* → *Qt Quick Controls 2 Application*. Será criado uma aplicação com uma estrutura bastante comum em aplicativos atuais. Dois componentes comumente utilizados são o `SwipeView` e o `TabBar`, descritos na Listagem 2.7.

O `SwipeView` é composto de páginas (componente `Page`). O usuário pode navegar entre as páginas deslizando o dedo horizontalmente na tela do dispositivo. No código apresentado, existem duas páginas: `Compras` e `CadastrarCompras`. Estas duas páginas foram criadas como objetos definidos pelo usuário. Para fazer isso, basta salvar o arquivo com o nome do componente desejado e extensão .qml. A partir dai, é permitido referenciar o componente em outras partes do código. Tanto *Compras* quanto *CadastrarCompras* implementam o componente *Page*. O componente *TabBar* está associado à propriedade *footer* do *ApplicationWindow* (tela principal). Ele serve para criar opções de navegação para a aplicação. No nosso exemplo, tem dois *TabButton* (o código interno foi suprimido), os quais permitem visualizar um ícone e um texto.

Listagem 2.7. Exemplo de SwipeView e TabBar

```
1 SwipeView {
2     id: swipeView
```

```
3       anchors.fill: parent
4       width: applicationWindow.width
5       currentIndex: tabBar.currentIndex
6
7       Compras {
8           id: comprasPage
9       }
10
11      CadastrarCompra {
12          id: cadastrarCompraPage
13      }
14  }
15  footer: TabBar {
16      id: tabBar
17      currentIndex: swipeView.currentIndex
18
19      TabButton {...}
20      TabButton {...}
21  }
```

Na linha 5, há um *binding* entre o `SwipeView` e o `tabBar` (*currentIndex: tabBar.currentIndex*). Assim, quando o índice atual do `TabBar` for modificado, o índice do `SwipeView` também será. Da mesma forma, na linha 17, há o *binding* no sentido inverso (*currentIndex: swipeView.currentIndex*): quando o índice atual do `SwipeView` mudar o índice atual do `TabBar` também muda.

O Componente `Compras` (Listagem 2.8), que implementa a página "Minhas Compras", serve para listar os produtos comprados e o valor total gasto.

Listagem 2.8. Parte do Código Compras.qml

```
1  Page {
2      title: "Minhas Compras"
3
4      property double valorTotal
5      property alias listViewModel: listView.model
6
7      Label {
8          visible: !listView.model.count
9          text: "Nehuma compra registrada!"
10         anchors.centerIn: parent
11     }
12
13     ListView {
14         id: listView
15         model: ListModel { }
16         anchors.fill: parent
17
18         header: Rectangle {...}
19         delegate: Rectangle {...}
20     }
```

```
21  }
```

Neste página, há um `Label` para informar quando não há nenhum item comprado. Observe o *binding* que existe em sua propriedade *visible*. Ele só fica visível quando a lista de elementos está vazia. Esta lista de elementos, por sua vez, é implementado pelo componente `ListView`.

O `ListView` provê uma visão em lista para elementos de um modelo. O atributo *model* define o modelo como sendo do tipo `ListModel`, inicialmente vazio. Há ainda dois componentes: o *header* e o *delegate*. O *header* é o componente responsável por mostrar os totais das compras. Ele tem uma cor diferente e possui dois elementos de textos, um alinhado/ancorado à esquerda ("Valor Total") e o outro à direita (o total de fato). O *delegate* define um *template* para visualização de cada um dos itens do modelo.

A Listagem 2.9 apresenta um resumo do componente *delegate*, implementado como um `Rectangle`. Além de alguns atributos, ele possui um *MouseArea* que trata o evento sobre o retângulo. Foram especificados: i) um item `Column`, que serve para mostrar nome do produto (`Row` 1) e Valor Pago (`Row` 2). Aqui há um exemplo dos posicionadores do QML. O `Column` coloca os elementos um abaixo do outro. O `Row` por sua vez, um ao lado do outro. No caso dos dois `Rows` existentes, eles têm dois `Texts` para mostrar o *label* da informação e a informação propriamente dita (e.g., Produto: Laranja); ii) dois componentes `Text`. Eles são responsáveis por apresentar a data e a quantidade. Esses componentes estão ancorados à direita do *parent* (*delegate*); iii) por fim, há o `Divider` que é responsável por mostrar uma barra horizontal que divide os itens.

Listagem 2.9. Parte do Código do *delegate*

```
1  delegate: Rectangle {
2      id: delegate
3      width: parent.width; height: 65
4
5      property int _itemId: itemId
6      property int _qtde: qtde
7      property real _valor: valor
8
9      MouseArea {...}
10
11     Column {
12         anchors { fill: parent; margins: 16 }
13
14         Row {...}
15
16         Row {...}
17
18     }
19
20     Text {...}
21
22     Text {...}
23
```

```
24      Divider { }
25  }
```

Um ponto importante é como os elementos do *model* são preenchidos. Eles são preenchidos a partir dos dados que estão no banco de dados. O componente `DataBase` contém o código-fonte responsável por fazer as operações de acesso ao banco. Antes de apresentar o preenchimento do *model*, explicaremos o componente `DataBase`.

O componente `DataBase` é apresentado de forma resumida na Listagem 2.10. Ele foi definido como um *Item*, componente não-visual do QML. Inicialmente, foi definido um *id*: rootItem. Depois são definidos dois sinais. Os sinais são responsáveis por gerar eventos. O sinal *compraRegistrada(var compra)* é invocado quando uma compra é efetuada. Observe o parâmetro *var compra*. Ele é utilizado para armazenar a informação da compra realizada no momento. Desta forma, quem tiver interessado nesse evento, pode capturá-lo, realizando alguma operação. O sinal *comprasCarregado(var compras)* é utilizado após ter uma operação de seleção no banco, a qual retorna todas as compras realizadas até então. O parâmetro *var compras* é utilizado para armazenar a informação de todas as compras realizada no momento.

Listagem 2.10. Código para manipular o banco de dados

```
1  import QtQuick 2.0
2  import QtQuick.LocalStorage 2.0
3
4  Item {
5      id: rootItem
6
7      signal compraRegistrada(var compra)
8      signal comprasCarregado(var compras)
9
10     readonly property string mainTable: "create table if not
11     exists compras(id integer not null primary key autoincrement,
12     produto TEXT, valor float, qtde integer, data_compra integer)"
13
14     readonly property var database:
15     LocalStorage.openDatabaseSync("MinhasCompras",
16     "1.0", "Minhas Compras", 1000000)
17
18     Component.onCompleted: {
19         rootItem.database.transaction(function(tx) {
20         tx.executeSql(mainTable) })
21         carregarCompras()
22     }
23
24     function salvarCompra(produto, valor, qtde, dataCompra) {...}
25
26     function removerCompra(id) {...}
27
28     function carregarCompras() {...}
29 }
```

A seguir, tem-se duas definições de propriedades *readonly*. Elas são usadas para definir os comandos SQL que criam a tabela e o banco de dados "MinhasCompras". Essas duas variáveis são utilizadas dentro do *handler Component.onCompleted*. Como o próprio nome diz, esse evento é invocado logo após o componente ter sido construído. O código da linha 19 é responsável por criar o banco e a tabela da aplicação. Na linha 21, é realizada a chamada para a função *carregarCompras()*, responsável por realizar a operação de *select* no banco. A Listagem 2.10 apresenta ainda as funções *salvarCompra()* e *removerCompra()*, responsáveis por fazer o *insert* e o *delete* no banco de dados, respectivamente.

A Listagem 2.14 apresenta em detalhes o código da função *salvarCompra()*. Ela possui os parâmetros que são informados pelo usuário. Na função é possível observar o código responsável pelo *insert*, e logo após tem-se a chamada de *compraRegistrada*. Esse é um ponto importante. É ai que é invocado o sinal definido inicialmente. Observe que é definido um objeto chave:valor com os dados do elemento inserido. Desta forma, que tratar esse evento poderá ter acesso ao último registro de compra realizado. Na nossa aplicação, isso será utilizado para atualizar a lista de compras logo após a compra ter sido realizada.

Listagem 2.11. Função de Cadastrar uma Compra

```
1  function salvarCompra(produto, valor, qtde, dataCompra) {
2      var result = ({})
3      rootItem.database.transaction(function(tx) {
4          result = tx.executeSql('insert or replace into compras
5          VALUES(?, ?, ?, ?, ?)', [null, produto, valor, qtde,
6          dataCompra])
7          compraRegistrada({
8              "itemId": parseInt(result.insertId),
9              "produto": produto,
10             "valor": parseFloat(valor),
11             "qtde": parseInt(qtde),
12             "data_compra": dataCompra
13         })
14     })
15     return result.insertId
16 }
```

A Listagem 2.12 apresenta em detalhes o código da função *carregarCompras()*. Na função é possível observar o código responsável pelo *select* e, logo após, tem-se um *for* que constrói vários objetos definidos no mesmo formato chave:valor. Eles são inseridos em um vetor *result*. Por fim, é invocado o sinal *comprasCarregado*, passando a lista de compras. Desta forma, quem tratar este sinal poderá ter acesso a todas as compras realizadas. Na nossa aplicação, isso será utilizado para montar a lista de compras inicialmente.

Listagem 2.12. Função de Listar Compras

```
1  function carregarCompras() {
2      var result = []
3      rootItem.database.transaction(function(tx) {
```

```
4           var query = "SELECT * FROM compras"
5           var results = tx.executeSql(query, [])
6           for (var i = 0; i < results.rows.length; i++) {
7               var item = results.rows.item(i)
8               var objc = {
9                   "itemId": parseInt(item.id),
10                  "produto": item.produto,
11                  "valor": parseFloat(item.valor),
12                  "qtde": parseInt(item.qtde),
13                  "data_compra": item.data_compra
14              }
15              result.push(objc)
16              result = result
17          }
18      })
19      comprasCarregado(result)
20  }
```

Agora já é possível retornar para o preenchimento do *model* no componente *Compras*. No componente *main.qml* há um trecho de código responsável por instanciar o componente `Database` (Listagem 2.13). Além de instanciar o `Database`, permitindo que o banco de dados seja criado, é possível observar o tratamento dos dois sinais destacados anteriormente. Em QML, para capturar um sinal deve-se utilizar um *signal handler*, cujo nome é sempre *on* seguido do nome do sinal. Assim, *onCompraRegistrada* captura o sinal *compraRegistrada()* e *onComprasCarregado* captura o sinal *comprasCarregado()*. O primeiro efetua um *append* (adição) de um novo elemento no *model* definido em minhas compras. Por sua vez, o segundo preenche o *model* com todas as compras existentes.

Listagem 2.13. Código que preenche o *model* de compras

```
1  Database {
2      id: database
3      onCompraRegistrada: comprasPage.listViewModel.append(compra)
4      onComprasCarregado: {
5          for (var i = 0; i < compras.length; ++i)
6              comprasPage.listViewModel.append(compras[i])
7      }
8  }
```

A partir desse momento, explicaremos o componente *CadastrarCompra.qml*, responsável por fazer o cadastro de uma compra. A Listagem 2.14 apresenta um trecho de código desse componente. Além das propriedades *title* e *dataSelecionada*, ele tem os subcomponentes `DatePicker` e `Column`. Dentro de `Column` são definidos, um abaixo do outro, os componentes visuais `Label` e `TextField`, responsáveis pela representação visual da tela.

Listagem 2.14. Componente *CadastrarCompra.qml*

```
1  Page {
2      title: "Cadastrar Compra"
3
```

```
4       property string dataSelecionada: ""
5
6       DatePicker {
7           id: datePicker
8           onDateSelected: {
9               dataSelecionada =
10              "%1/%2/%3".arg(date.day).arg(date.month).arg(date.year)
11              dataCompra.focus = false
12          }
13      }
14
15      Column {
16          spacing: 10
17          anchors { fill: parent; margins: 16 }
18
19          Label { text: "Digite o nome do produto" }
20          TextField {...}
21
22          Label { text: "Digite o valor do produto" }
23          TextField {...}
24
25          Label { text: "Digite quantidade de itens comprado" }
26          TextField {...}
27
28          Label { text: "Digite a data da compra" }
29          TextField {
30              id: dataCompra
31              width: parent.width
32              readOnly: true
33              text: dataSelecionada
34              onFocusChanged: if (focus) datePicker.open()
35          }
36
37          Button {...}
38      }
39  }
```

O `DatePicker` é um componente responsável por apresentar um calendário[4]. Ele tem o tratamento de um sinal *dateSelected* que preenche o atributo *dataSelecionada*. Observe que *dataSelecionada* é utilizado para ajustar a propriedade *text* do `TextField` com *id* igual a "dataCompra".

A Listagem 2.15 apresenta o código referente ao componente *Button* de *CadastrarCompra.qml*. Nele há o tratamento de um sinal *clicked*, que é responsável por invocar a função *salvarCompra* de *Database.qml*.

[4]Ele foi implementando separadamente no arquivo `DatePicker.qml` (o mesmo não será explicado em detalhes aqui)

Listagem 2.15. Componente *Button CadastrarCompra.qml*

```
1  Button {
2      text: "Salvar"
3      width: parent.width
4      onClicked: {
5          var result = database.salvarCompra(produto.text,
6          valor.text, qtde.text, dataSelecionada)
7          if (result) {
8              showdialog("OK", "Compra registrada com sucesso!")
9              produto.text = valor.text = qtde.text
10             = dataSelecionada = ""
11         } else {
12             showdialog("Erro!", "N o foi poss vel salvar o item!")
13         }
14     }
15 }
```

Desta forma, concluímos a explicação das principais partes da aplicação Minhas Compras.

2.4.2. Serviços RESTful e Mapas

Nessa seção apresentaremos o aplicativo que lista as estações de metrô do Rio de Janeiro[5]. Os dados das estações estão disponíveis em formato JSON[6]. Nessa aplicação, utilizaremos serviços RESTful e mapas.

A Figura 2.2 apresenta as telas do aplicativo. Neste exemplo, também são utilizados os componentes básicos do exemplo da Seção 2.4.1: `SwipeView`, `Page`, `TabBar`, `Column`, *ListView*, etc. A tela da esquerda lista todas as estações. Ao clicar em uma das estações, é apresentado o mapa com a localização dessa estação (tela da direita).

A Listagem 2.16 apresenta o código que foi utilizado para construir a tela que lista todas as estações. Foi suprimido o código do componente `Column` (Linha 28), responsável por mostrar os dados da estação. Ele segue o mesmo padrão do exemplo anterior.

Listagem 2.16. Código que implementa a listagem de estações

```
1  Page {
2      id: page1
3      title: "Esta\c{c}\~oes de Metr\ o  - RJ"
4
5      BusyIndicator {
6          visible: firstPageView.count === 0
7          anchors.centerIn: parent
8      }
9      ListView {
```

[5]Utilizamos os dados abertos da prefeitura do Rio de Janeiro http://data.rio/group/transporte-e-mobilidade

[6]JSON disponível em http://dadosabertos.rio.rj.gov.br/metro/api/v1/rest/Estacoes.cfm?token=40A22AF7-A9C6-86D6-E211ABA64BF73830pretty=true

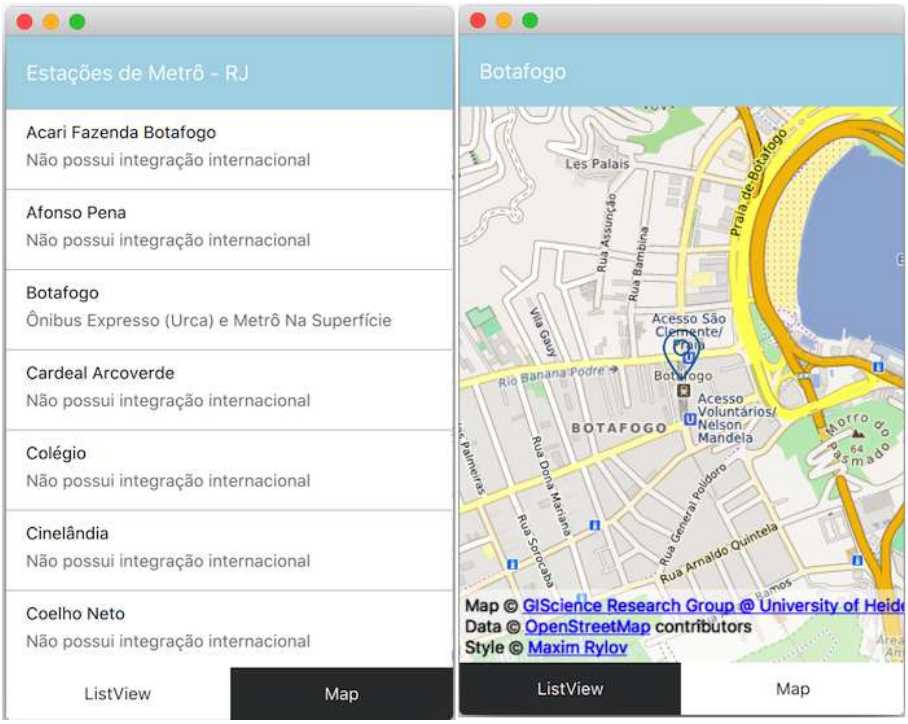


Figura 2.2. Telas da app Mapa Interativo

```
10          id: firstPageView
11          spacing: 1
12          anchors.fill: parent
13          delegate: Rectangle {
14              color: "#fff"
15              width: firstPageView.width; height: 60
16
17              MouseArea {
18                  anchors.fill: parent
19                  onClicked: {
20                      currentPlaceName = ESTACAO
21                      currentPlaceCoordinates =
22                      QtPositioning.coordinate(LATITUDE, LONGITUDE)
23                      swipeView.incrementCurrentIndex()
24                  }
25              }
26
27              Column {...}
28
29              Rectangle { width: parent.width; height: 1; color: "#ccc" }
30          }
31          model: ListModel {
32              id: firstPageModel
33          }
34
35          Component.onCompleted: {
```

```
36                 var baseUrl = "http://dadosabertos.rio.rj.gov.br/metro/api/v1
37                 /rest/Estacoes.cfm?token=40A22AF7-A9C6-86D6-E211ABA64BF73830&
38                 pretty=true&filter="
39                 requestHttp("GET", baseUrl, null, function(s,r) {
40                     for (var i = 0; i < r.length; ++i)
41                         firstPageModel.append(r[i])
42                 })
43             }
44         }
45 }
```

Nas linhas 5 a 8 define-se o componente *BusyIndicator*. Ele é utilizado para indicar que está havendo um processamento em *background*. No nosso caso, isso se refere à busca dos dados através do serviço RESTful. A partir da linha 10 é definido o componente `ListView`. Destacaremos as partes ainda não discutidas aqui nesse texto. O componente `MouseArea` (linhas 18-26) cria um evento de clique para cada elemento da lista. Quando o usuário clica em uma das estações é modificado o nome da estação atual e ajustadas as coordenadas atual para a estação selecionada. Os atributos ESTACAO, LATITUDE e LONGITUDE fazem parte do JSON recuperado. Por fim, é incrementado o índice do `SwipeView`, para abrir a página correspondente ao mapa.

Nas linhas 36-44 é implementado o *handler* `Component.onCompleted`. Ele é invocado após o componente ao qual ele faz parte ser carregado. Nesse evento, define-se a URL que contém os dados da estação e realiza-se uma requisição GET do HTTP. Por fim, há um laço que preenche o *model* (linhas 32-34) com cada um dos objetos do JSON.

A função que faz a requisição HTTP é apresentada na listagem 2.17.

Listagem 2.17. Função que faz a requisição HTTP

```
1  function parseJson(str) {
2      return str.toString().replace("<pre>", "").replace("</pre>", "")
3  }
4
5  function requestHttp(type, url, args, callback) {
6      var xhr = new XMLHttpRequest
7      xhr.open(type === "POST" ? "POST" : "GET", url, true)
8      xhr.setRequestHeader("Content-type", "Application/Json")
9      xhr.onreadystatechange = function() {
10         if (xhr.readyState === XMLHttpRequest.DONE) {
11             try {
12                 callback(xhr.status, JSON.parse(parseJson(xhr.responseText)
13             } catch(e) {
14                 console.log("Request error:")
15                 console.error(e)
16             }
17         }
18     }
19     xhr.send(args || ({}))
20 }
```

A Listagem 2.18 apresenta o código da página que mostra o mapa. Primeiramente, tem-se o componente `Plugin` (linhas 5-18), responsável por descrever os serviços baseado em localização. Esse componente serve para especificar qual o *plugin* GeoServices será utilizado em várias tarefas da API de localização. Os *plugins* são usados nos componentes `Map`, `RouteModel`, `GeocodeModel`, entre outros.

Listagem 2.18. Código que implementa o mapa

```
1  Page {
2      id: page2
3      title: currentPlaceName
4
5      Plugin {
6          id: mapPlugin
7          name: "osm"
8          PluginParameter { name: "osm.useragent";
9              value: "My First QML App" }
10         PluginParameter { name: "osm.mapping.host";
11             value: "http://osm.tile.server.address/" }
12         PluginParameter { name: "osm.mapping.copyright";
13             value: "Open Street Map" }
14         PluginParameter { name: "osm.routing.host";
15             value: "http://osrm.server.address/viaroute" }
16         PluginParameter { name: "osm.geocoding.host";
17             value: "http://geocoding.server.address" }
18     }
19
20     Map {
21         id: map
22         anchors.fill: parent
23         plugin: mapPlugin
24         center: currentPlaceCoordinates
25         zoomLevel: 15
26
27         MapQuickItem {
28             id: marker
29             coordinate: currentPlaceCoordinates
30             anchorPoint.x: image.width * 0.5
31             anchorPoint.y: image.height
32             sourceItem: Column {
33                 Image { id: image; source: "qrc:/marker.png";
34                 width: 45; height: width }
35             }
36         }
37     }
38 }
```

Em seguida, tem-se o componente `Map` (linhas 20-37), responsável por apresentar o mapa. Observe os seguintes atributos: *center*, que define a centralização do mapa. Este atributo tem um *binding* com *currentPlaceCoordinates*, definido no `MouseArea` expli-

cado anteriormente; *plugin*, que relaciona com o *plugin mapPlugin* definido; *zoomLevel* que define o nível de *zoom* do mapa.

Dentro do `Map` é definido um objeto `MapQuickItem`, o qual apresenta um objeto no mapa. Nesse caso, é apresentado um ícone (imagem definida na linha 33). O componente `MapQuickItem` possui o atributo *coordinate*, que define a posição da imagem (ícone) no mapa. Este atributo também possui um *binding* com *currentPlaceCoordinates*. A imagem é definida pela propriedade *sourceItem*.

2.4.3. Multimedia

Conforme mencionado anteriormente, o módulo `Qt Multimedia` disponibiliza uma série de recursos para manipulação de áudio e video. A Listagem 2.19 apresenta um exemplo de uso das câmeras frontal e traseira de um smartphone, com a exibição instantânea da imagem capturada.

Listagem 2.19. Exemplo de uso de câmera frontal e traseira

```
1  import QtQuick 2.4
2  import QtQuick.Controls 1.3
3  import QtQuick.Layouts 1.1
4  import QtMultimedia 5.4
5
6  Item {
7      id: root
8      anchors.fill: parent
9      property var cameras: QtMultimedia.availableCameras
10     property int currentCamera: 0
11
12     ColumnLayout {
13         anchors { fill: parent; margins: spacing }
14         VideoOutput {
15             Layout { fillWidth: true; fillHeight: true }
16             source: Camera { id: camera }
17             autoOrientation: true
18         }
19         RowLayout {
20             Label {
21                 Layout.fillWidth: true
22                 text: camera.displayName
23             }
24             Button {
25              visible: cameras.length > 1
26              text: "Change camera"
27              onClicked: {
28               currentCamera++
29               if (currentCamera >= cameras.length)
30                 currentCamera = 0
31         camera.deviceId =cameras[currentCamera].deviceId
32               }
33             }
```

```
34              }
35          }
36  }
```

Na linha 9, é criada uma propriedade denominada *cameras*, inicializada com um *array* de objetos que representam todas as câmeras disponíveis no dispositivo utilizado. A propriedade *currentCamera* (linha 10) armazena a câmera sendo atualmente utilizada. Em seguida, utiliza-se um objeto do tipo `VideoOutput` (linhas 14 a 18) para exibir a imagem sendo capturada pela câmera atual. Isto é realizado ao ajustar o atributo *source* para um objeto do tipo `Camera`.

Logo abaixo do objeto `VideoOutput`, são apresentados o nome da câmera sendo atualmente utilizada (através de um objeto `Label` – linha 22) e um botão para alternar a câmera atual (objeto `Button` – linha 24). Ao clicar neste botão, a aplicação alterna a câmera atualmente utilizada para a próxima câmera do array de câmeras disponíveis, retornando para a primeira quando não há mais câmeras disponíveis (linhas 27 a 33).

2.4.4. Sensores

O módulo `Qt Sensors` disponibiliza um amplo conjunto de funcionalidades para acesso a sensores tais como acelerômetro, sensor de proximidade, altímetro, sensor de luz, sensor de temperatura, giroscópio, sensor de humidade, dentre outros. Grande parte destes recursos funcionam de maneira uniforme em uma ampla faixa de dispositivos.

A Listagem 2.20 apresenta um exemplo de uso de acelerômetro e sensor de proximidade. O exemplo funciona em qualquer smartphone Android ou iOS que possua estes dois recursos. O aplicativo permite que um pequeno círculo verde tenha a sua posição (x, y) controlada através do acelerômetro do smartphone. Adicionamente, a ativação do sensor de proximidade modifica a cor do círculo de verde para preto.

Listagem 2.20. Exemplo de uso de acelerômetro e sensor de proximidade

```
1  import QtQuick 2.8
2  import QtQuick.Controls 2.1
3  import QtSensors 5.8
4  import QtQuick.Window 2.0
5
6  ApplicationWindow {
7      id: window
8      width: 640; height: 480
9
10     property var pixelDensity
11
12     Rectangle {
13         id: ball
14         x: window.width/2 - width/2; y: window.height/2 - height/2
15         width: 24*pixelDensity; height: 24*pixelDensity
16         color: "green"; radius: width/2
17         Behavior on x { NumberAnimation { duration: 100 } }
18         Behavior on y { NumberAnimation { duration: 100 } }
```

```
19      }
20      Accelerometer {
21          active: true; dataRate: 20
22          onReadingChanged: {
23              var newX = ball.x - reading.x * pixelDensity * 30
24              var newY = ball.y + reading.y * pixelDensity * 30
25              if (newX + ball.width > window.width)
26                  newX = window.width - ball.width
27              if (newY + ball.height > window.height)
28                  newY = window.height - ball.height
29              if (newX < 0) newX = 0
30              if (newY < 0) newY =  0
31              ball.x = newX
32              ball.y = newY
33          }
34      }
35      ProximitySensor {
36          active: true; dataRate: 20
37          onReadingChanged: ball.color = (reading.near) ? "black":"green"
38      }
39
40      Component.onCompleted: pixelDensity = Screen.logicalPixelDensity
41  }
```

A linha 3 realiza o import do módulo `Qt Sensors`, fazendo com que os objetos QML relacionados a sensores estejam disponíveis para uso na aplicação. O círculo verde a ser controlado é criado nas linhas 12 a 19 (no QML, um círculo é representado como um retângulo com o atributo *radius* ajustado para metade da sua largura). O acesso ao acelerômetro é realizado nas linhas 20 a 34, onde um *handler* para o signal readingChanged realiza o posicionamento do círculo de acordo com os dados coletados no acelerômetro. As linhas 35 a 38 realizam a leitura do sensor de proximidade.

2.5. Conclusão

Em um ambiente multimídia interativo, os dispositivos do usuário são muito heterogêneos. Embora alguns dispositivos possam ter gráficos, endereços IP, telas sensíveis ao toque (painéis de controle de casa, por exemplo), a maioria dos casos pode ter muito menos recursos. Os usuários precisam ter uma maneira de se comunicar com seus dispositivos, configurar os seus comportamentos, obter dados ou visualizá-los e alterar o seu status. Nestes ambientes, os requisitos de experiência do usuário (UX) transcendem um simples LED intermitente ou uma interface de um botão e é parte importante para o sucesso de um ambiente multimídia interativo.

Mais do que nunca, o mundo da IoT depende de software para criar dispositivos mais inteligentes e conectados. Os critérios para a sua escolha das tecnologias de software em um projeto ubíquo pode ou não coincidir com os que definimos neste artigo. No entanto, se seu projeto precisa ser conectado, desenvolvido rapidamente, multiplataforma com interfaces de usuário e recursos de compartilhamento, então você deve considerar o uso do Qt.

Referências

Blanchette, J. and Summerfield, M. (2006). *C++ GUI Programming with Qt 4*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Islam, S. M. R., Kwak, D., Kabir, M. H., Hossain, M., and Kwak, K. S. (2015). The internet of things for health care: A comprehensive survey. *IEEE Access*, 3:678–708.

Krumm, J. (2009). *Ubiquitous Computing Fundamentals*. Chapman & Hall/CRC, New York, NY, USA, 1st edition.

Neto, M. C. M. (2015). Desenvolvimento de aplicações ubíquas com arduino e raspberry pi. In *Proceedings of the 20st Brazilian Symposium on Multimedia and the Web*, pages 1–40. SBC.

Russo, C. V. and Vytiniotis, D. (2009). Qml: Explicit first-class polymorphism for ml. In *Proceedings of the 2009 ACM SIGPLAN Workshop on ML*, ML '09, pages 3–14, New York, NY, USA. ACM.

Summerfield, M. (2007). *Rapid GUI Programming with Python and Qt : the Definitive Guide to PyQt Programming*.

The Qt Company (2016). Qt : cross-platform application and ui framework.

Biografia Resumida dos Autores

Manoel C M Neto

Manoel Carvalho Marques Neto é professor do Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBa), Doutor em Ciência da Computação pela Universidade Federal da Bahia (UFBa - 2011), Mestre em Redes de Computadores (2004) e graduação em Análise de Sistemas (2002), ambos pela Universidade Salvador (UNIFACS) . É pesquisador-membro do Grupo de Pesquisa em Sistemas Distribuídos, Otimização, Redes e Tempo-Real (GSORT - IFBa). Suas atuais áreas de interesse incluem: IoT, Computação Ubíqua, software livre, multimídia, web, mobile, entre outras. CV: http://lattes.cnpq.br/7300048297400666

Renato L Novais

Renato Lima Novais é doutor em Ciência da Computação pela Universidade Federal da Bahia (2013), Mestre em Informática pela Pontifícia Universidade Católica do Rio de Janeiro (2007), e graduado em Ciência da Computação pela Universidade Federal da Bahia (2004). Durante o doutorado realizou estágio sanduíche no Fraunhofer Center for Experimental Software Engineering, Maryland, USA. É professor do Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA). Tem experiência nas áreas de Engenharia de Software, Big Data e Smart Cities, atuando principalmente nos seguintes temas: visualização de informação, compreensão e evolução de software, e engenharia de software experimental. CV: http://lattes.cnpq.br/5036544358013553

Sandro S Andrade

Sandro Santos Andrade é professor do Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBa), Doutor em Ciência da Computação e Mestre em Mecatrônica, ambos pela Universidade Federal da Bahia (UFBa). É pesquisador-colaborador do Laboratório de Sistemas Distribuídos (LaSiD - UFBa) e pesquisador-membro do Grupo de Pesquisa em Sistemas Distribuídos, Otimização, Redes e Tempo-Real (GSORT - IFBa). Suas atuais áreas de interesse incluem: arquitetura de software, software livre, engenharia de software experimental, engenharia de software baseada em busca, sistemas distribuídos, sistemas self-adaptive/self-organizing e design science. Trabalhou por muitos anos em projetos envolvendo computação gráfica aplicada à medicina, desenvolvimento de jogos, visão computacional e computação musical. CV: http://lattes.cnpq.br/0177714301545658

Capítulo 3

Game Development for Researchers: From Concept to User Experience Data Collection

Marcio Maestrelo Funes[1], Leandro Agostini do Amaral[1],
Rudinei Goularte[1], Renata Pontin Mattos Fortes[1]

[1]Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação (ICMC/USP)

{marciofunes, leandroagostini}@usp.br, {rudinei, renata}@icmc.usp.br

Abstract

This chapter aims to present game development for researchers who wish to analyze user experience by means of data collection in their research, but find difficulties in the development of their applications. The chapter is divided into two parts: the first one presents game design theory highlighting definitions and examples about data collection involving guidelines and usability, the second one presents a game development approach by means of the Construct 2 tool using multimedia resources (audio, videos and photos) and variables (quantitative and qualitative) aiming to store data generated during users interaction.

Resumo

Este capítulo visa apresentar o desenvolvimento de jogos para pesquisadores que desejem analisar a experiência do usuário por meio da coleta de dados em suas pesquisas mas encontram dificuldades no desenvolvimento de suas aplicações. O capítulo está dividido em duas partes: a primeira parte apresenta a teoria sobre a concepção de jogos destacando as definições e exemplos sobre a coleta de dados envolvendo diretrizes e usabilidade, a segunda parte apresenta uma abordagem para o desenvolvimentos de jogos por meio da ferramenta Construct 2 utilizando recursos multimídia (áudio, vídeos e fotos) e variáveis (quantitativas e qualitativas) visando o armazenamento dos dados gerados durante a interação do usuário.

3.1. Introdução

Desenvolvimento de jogos se refere a uma área de conhecimentos que abrange diversas linhas de estudos em Computação, Educação, *Design*, Simulação, Entretenimento, entre

outras. Por envolver diversas possibilidades de integração de conteúdos e modos de interação avançados, muitas pesquisas podem se beneficiar de seus mecanismos para atender demandas específicas dos procedimentos metodológicos científicos, em especial para coleta de dados.

Acredita-se que a pesquisa científica possa obter muitas vantagens a partir da utilização de jogos [Coovert et al. 2017], porém este cenário se configurava, por bastante tempo, como de difícil acesso, por apresentar muitos obstáculos que tornaram tímida a imersão dos jogos no campo de pesquisas científicas, incluindo custos de desenvolvimento exorbitantes e representações gráficas inexatas que limitavam a fidelidade de simulação. Os avanços nas *engines* de jogos (por exemplo, CryEngine[1], Unity[2] e Unreal Engine[3]) diminuíram muito os problemas associados a sua adoção. Assim, é hora de considerar seriamente a utilidade de *serious games* para a área de pesquisa [Coovert et al. 2017].

Diante dos inúmeros domínios de conhecimento que abordaram o tema de jogos e atendo-se ao segmento que detém o entretenimento como segundo plano, emerge a denominação *serious games*. De modo geral, a expressão "*serious games*" (em Português, jogos sérios) é adotada com o significado de que os jogos desse tipo possuem uma motivação educacional explícita, cuidadosamente pensada, e não se destinam a ser jogados exclusivamente para diversão ou entretenimento. Porém, essa interpretação não significa que jogos sérios não sejam, ou não devam ser, divertidos; ao contrário, a ideia é de que jogos sérios possuam elementos criativos e agradáveis como seus componentes.

Os *serious games* são projetados para ter um impacto no público-alvo, que está além do entretenimento puro [Gee 2003, Greitzer et al. 2007, Michael and Chen 2005]. Um dos domínios de aplicação mais importantes está no campo da Educação, dado seu potencial reconhecido, tendo em conta a necessidade contínua de aprimoramento das técnicas e atualização de recursos educacionais [Frederik et al. 2010, Van Eck 2006].

Outra iniciativa mais recente para uso dos jogos sérios é sua aplicação com o público idoso. Esse público representa uma comunidade de potenciais usuários que podem se beneficiar muito de jogos digitais [Souza and Trevisan 2014], por representarem uma ferramenta que contribui para a qualidade de vida durante o processo de envelhecimento, amenizando o declínio de alguns aspectos decorrentes deste processo, como aspectos motores, perceptivos, cognitivos e psicossociais [Chen et al. 2012].

Nesse contexto, segundo [Valladares-Rodríguez et al. 2016], uma das áreas mais promissoras para aplicação de jogos sérios voltados para o público idoso é a avaliação neuropsicológica. Os jogos podem revolucionar a avaliação cognitiva dos idosos em ambientes clínicos, permitindo que as avaliações sejam mais frequentes, mais acessíveis e mais agradáveis [Tong et al. 2016]. Em uma pesquisa sobre diferentes momentos e situações clínicas dos idosos, os estudos de [Gonçalves 2012] revelam que *serious games* têm se mostrado muito importantes para manter e desenvolver as competências cognitivas e sociais dos idosos. Na literatura especializada, estudos apontam que jogos sérios para idosos podem ser uma opção para melhorar a deterioração das funções cognitivas [Torres

[1]http://www.crytek.com/cryengine
[2]https://unity3d.com/pt
[3]https://www.unrealengine.com/

2011], a manutenção da autoimagem, das funções motoras [Wiemeyer and Kliem 2011] e o convívio social (afeto e solidão) [Jung et al. 2009].

Diante do potencial relevante das pesquisas envolvendo jogos, este capítulo agrega conceitos sobre a utilização de jogos em pesquisa; em especial, é apresentado o procedimento de pesquisa de coleta de dados adotando-se jogos, e exemplos práticos de desenvolvimento.

3.2. Termos e Definições

São muitos os termos e definições utilizados em pesquisa para desenvolvimento e aplicações de jogos. Assim, neste capítulo fazemos um recorte, considerando os conceitos da área de jogos mais relacionados com a atividade de coleta de dados em ambiente de pesquisa, seja na produção de jogos completos ou no uso de elementos de *gamificação*, explicados mais adiante, para contribuir no engajamento do usuário dos jogos.

O engajamento do jogador está associado diretamente aos elementos de construção de um jogo, como também ao enriquecimento da experiência do usuário. Diante da diversidade de habilidades dos usuários (tanto limitações, quanto capacidades intensificadas), é essencial o uso de diretrizes que contribuam para a inclusão do maior número de especificidades, contribuindo com sua usabilidade e acessibilidade.

3.2.1. Jogos & Coleta de Dados

A utilização de jogos no ambiente de pesquisa visa enriquecer o objetivo fim de pesquisas aplicadas, em especial no contexto de coleta de dados, na obtenção de evidências científicas que promovam comprovação a questionamentos sob investigação. Assim, uso de jogos por pesquisadores não está associado apenas ao planejamento e desenvolvimento de jogos na forma de produtos acabados, ou completos, mas também no uso de elementos de *gamificação*, de modo a oferecer suporte às tarefas requeridas durante as atividades desenvolvidas na pesquisa.

Entende-se por *gamificação*, o uso de mecânicas, ideias e estética de jogos para engajar pessoas, motivar ações, promover o aprendizado e solucionar problemas [Kapp 2012]. Portanto, ao realizar *gamificação*, faz-se uso de elementos e técnicas de *design* de jogos em situações fora do ambiente de jogos, com o intuito de obter maior participação e envolvimento do usuário [Pedro 2016].

No contexto de auxílio computacional a pesquisadores, o conceito por trás do termo "jogo" pode ser resumido, conforme definido por [Salen and Zimmerman 2004]: refere-se a um sistema, no qual jogadores se envolvem em um conflito artificial, definido por regras e que possui um resultado quantificável.

De modo geral, o desenvolvimento de jogos fundamenta-se em determinados elementos, que podem caracterizar e até mesmo gerar o engajamento do usuário, tais como: o sistema, os jogadores, abstração, desafio, regras, interatividade, *feedback*, resultados quantificáveis, reações emocionais e história [Kapp 2012]. Esses elementos, definidos a seguir, devem ser considerados no âmbito de quaisquer tipos de jogos.

- **Sistema:** conjunto de elementos interligados que ocorrem no contexto do jogo;

- **Jogadores:** pessoas interagindo com o jogo ou com outras pessoas;
- **Abstração:** refere-se a uma imagem mental da realidade para definir o contexto do jogo;
- **Desafio:** o jogo incorpora atividades e objetivos que instigam os jogadores a atingirem metas e resultados em diferentes níveis de dificuldade;
- **Regras:** compreendem a estrutura de fluidez do jogo, definindo o ambiente para uma jogabilidade adequada;
- **Interatividade:** é o processo de relação do jogador com o conteúdo, com o jogo, ou com outros jogadores;
- ***Feedback:*** retorno do sistema às ações realizadas pelos jogadores;
- **Resultados quantificáveis:** o jogo deve reproduzir de maneira clara, ao usuário, o conceito de ganho ou perda, a partir da interação com o jogador;
- **Reações emocionais:** o jogo deve proporcionar sensações de prazer ou frustração, dependendo do resultado das atividades pelo jogador;
- **História:** o uso da narrativa de uma história contextualiza o jogador ao cenário, introduzindo um significado para o jogo.

Em especial, no contexto de coleta de dados, jogos propiciam que formas inovadoras de aplicação de seus elementos possam ser utilizadas nos procedimentos para coletar dados de modo isento e criterioso, conforme rigor prescrito nas investigações científicas. Na perspectiva de métodos de pesquisa em geral, tem-se que a coleta de dados é considerada uma das fases mais importantes em uma pesquisa [Pádua 2000], [Marconi and Lakatos 2001], [Marconi and Lakatos 2002], [da Costa and da Costa 2009], seja ela qualitativa ou quantitativa. Entre os principais instrumentos utilizados durante a coleta de dados, tem-se:

1. Observação – quando os fatos são percebidos diretamente pelo pesquisador, sem qualquer intermediação. Deve ser planejada, registrada, submetida a verificação e controles de validade e precisão.
Entre as desvantagens, tem-se: a presença do pesquisador pode provocar alterações no comportamento dos observados.

2. Entrevista – requer formulação de perguntas ao investigado, tratando-se de uma interação social. Possibilita a obtenção de dados referentes a um espectro de informações. Dentre as vantagens: proporciona obtenção de esclarecimentos e captura das expressões dos entrevistados (corporal, gestos, voz...).
Entre as desvantagens, tem-se: pode sofrer influência das opiniões pessoais do entrevistador, além de envolver custos para treinamento e aplicação das entrevistas.

3. Questionário – conjunto de questões apresentadas por escrito aos investigados, permite que um grande número de respondentes possa participar / colaborar.

Entre as desvantagens, tem-se: exclusão dos que não sabem ler e escrever, além de requerer um tratamento diferenciado para as pessoas com deficiência visual, por exemplo; impede o conhecimento das circunstâncias do respondente, quando respondeu ao questionário.

4. Documentos – são fontes em "papel" (arquivos, registros estatísticos, diários, biografias, jornais, revistas, relatórios, cartas memórias, boletins...) que armazenam dados diversos, os quais possibilitam aquisição de conhecimentos ali representados.
Entre as desvantagens, tem-se: requer minucioso rigor na obtenção e interpretação do material juntado.

Vale destacar que esses instrumentos podem ser utilizados de modo complementar, conforme o tipo da pesquisa. Em qualquer tipo de levantamento (coleta de dados) com o público-alvo de uma pesquisa, exige-se o máximo de cuidado no momento de coletar informações. Dentre os cuidados, no contexto do tema apresentado neste capítulo, os elementos de jogos devem ser adequadamente projetados, conforme discutimos nas seções a seguir.

3.2.2. Experiência do usuário & Jogos

Dentre as definições para a expressão "*User experience (UX)*" (em Português Experiência do Usuário - EU), muitos pesquisadores convergem, considerando que UX abrange todos os aspectos da interação do usuário final com determinado sistema, serviço ou produto [Hassenzahl and Tractinsky 2006], [Goodman et al. 2013].

Para [Goodman et al. 2013], um aspecto relevante a ser considerado no desenvolvimento de jogos, relacionado a experiência do usuário, é que eles podem ser tediosos, quando muito fáceis, e frustrantes, quando muito difíceis. A maioria dos jogos *single-player* (jogador sozinho) permite ao jogador apenas um ajuste grosseiro de nível de dificuldade, geralmente possibilitando a escolha entre os modos de jogo: fácil, médio, difícil ou muito difícil [Hunicke 2005]. Tal abordagem mostra-se estática frente à interação do jogador com o jogo e pode apresentar uma divergência entre o nível de habilidade do jogador e a dificuldade geral do jogo.

Poucos desenvolvedores de jogos comerciais implementaram sistemas de ajuste dinâmico para seus jogos, dentre estes, pode-se citar God Hand® (Clover Studio e Capcom, 2006), no qual um medidor regula, ao longo do jogo, a inteligência e força dos adversários. Este medidor aumenta sempre que o jogador consegue, com sucesso, se esquivar de ataques inimigos ou golpear seus oponentes, diminuindo a medida que o jogador é atingido. Uma grande recompensa é oferecida para os jogadores que vencem oponentes que, de acordo com o medidor, são muito difíceis para seu nível de habilidade. Entretanto, nos jogos comerciais, no campo dos *serious games*, não foi identificada nenhuma iniciativa neste sentido.

Não tão distante da adaptabilidade dos jogos aos diferentes níveis de habilidades, a experiência do usuário pode estar relacionada também com questões de interface. Tais questões são relativamente mais complexas para pessoas (potenciais jogadores) que apresentem algum declínio nas suas funções cognitivas ou motoras, incentivando pesquisas na concepção de jogos planejados para uso universal, para todos.

É importante ponderar sobre a perspectiva de se desenvolver um software tradicional, que não seja um jogo, considerando-se a experiência de usuário. O desenvolvimento com UX para software tradicional visa a remoção de problemas ao usuário. Por outro lado, no desenvolvimento de um jogo, a experiência do usuário se refere a oferecer "problemas" / desafios ao usuário. Em ambos os casos, deve-se examinar o raciocínio exigido dos usuários e sua capacidade cognitiva.

Em geral, o trabalho de um desenvolvedor que se preocupa com a experiência do usuário, de software tradicional, é direcionado para criar uma ótima experiência para o usuário ao usar o produto para um propósito. Assim, esse desenvolvedor faz o máximo possível para que a interface seja transparente (o usuário precise pensar o mínimo possível), sugestiva (o usuário saiba quais possibilidades de interação ele possui), flexível (que se adapte à medida que o usuário desenvolve habilidades), rica em informações e *feedback* (o usuário saiba mais sobre algo ou saiba situações que lhe traga problemas). Trata-se de conduzir os usuários a construir um modelo mental que os ajude a fazer o seu trabalho. Exemplos clássicos: o do volante de um carro ou a metáfora da área de trabalho inteira. O volante está escondendo do usuário o fato de que existe um mecanismo bastante complexo entre a roda e os pneus. Em vez disso, está tentando levar o usuário a pensar "girar a roda, virar o carro". Esta é uma ilusão intencional. O trabalho dos desenvolvedores de jogos, geralmente, é voltado a criar uma ótima experiência ao jogador ao jogar um jogo. Portanto, eles fazem o máximo possível para criar o jogo:

- **Desafiador** – muitas vezes, é requerido que o jogo faça com que o usuário pense, com atenção, use a memória e sua capacidade de raciocínio.
- **Explorável** – geralmente, é requerido que o usuário pense que há sempre mais possibilidades a serem descobertas.
- **Escalável** – para que os jogadores aprendam a jogar melhor, enquanto jogam.
- **Surpreendente** – para que os jogadores sejam cativados pelo espetáculo que o jogo apresente. Com esta relação de empolgação, tem-se inclusive um convite ao erro pois, muitas vezes, é requerido que os jogadores aprendam através de seus erros.

Finalmente, enquanto a experiência do usuário em software tradicional visa claramente ocultar toda a complexidade do produto, em jogos UX visa claramente ensinar a lidar com a complexidade. Ainda assim, acessibilidade em jogos é requisito de qualidade importante, e na próxima seção encontra-se descrito.

3.2.3. Diretrizes de acessibilidade para jogos

Com a finalidade de produzir uma referência objetiva aos desenvolvedores de jogos, e no sentido de incluir o maior número de pessoas possível, considerando as diversas necessidades apresentadas por pessoas com deficiências, um esforço colaborativo entre produtoras, especialistas e pesquisadores foi realizado, surgindo as diretrizes de acessibilidade para jogos [Ellis et al. 2012].

As diretrizes não devem ser utilizadas apenas durante o processo de desenvolvimento do jogo, mas também na avaliação contínua da aplicação, por especialistas e

usuários finais. Diante da variedade de abordagens existentes, no tocante à avaliação de acessibilidade para jogos digitais, o uso de mais de um método de avaliação é importante para se obter maior qualidade nos resultados [Fortes et al. 2017].

As diretrizes foram categorizadas em três tipos: básico, intermediário e avançado. Nessas categorias são contempladas as diferenças quanto ao número de pessoas beneficiadas, o impacto provido para os utilizadores e o custo de implementação.

As diretrizes são também agrupadas em subcategorias, relacionadas com as diferentes habilidades: motora, cognitiva, visual, auditiva e de fala. A seguir, esses **Conjuntos de Diretrizes de Acessibilidade para Jogos (CDAJ**) são apresentados.

[CDAJ.1] para USO GERAL

- Básico
  - Fornecer detalhes sobre os **recursos de acessibilidade** no pacote e/ou jogo
  - Oferecer uma grande variedade de níveis de dificuldade
  - Certificar que todas as configurações são salvas/lembradas
- Intermediário
  - Permitir que o nível de dificuldade possa ser alterado durante o jogo, seja através de configurações ou dificuldade adaptativa
  - Incluir algumas **pessoas com deficiência** entre os participantes que testarão o jogo
  - Oferecer um meio de contornar elementos de jogabilidade que não fazem parte da mecânica central, através das definições ou uma opção de ignorar durante o jogo
  - Incluir modos auxiliares, tais como mira automática e direção assistida
  - Fornecer possibilidade manual de salvar
  - Fornecer a possibilidade de salvar automático
  - Permitir uma preferência para ser definida a jogabilidade *multiplayer* online, com ou sem outros que estão usando recursos de acessibilidade, que poderiam dar uma vantagem competitiva
- Avançado
  - Incluir **todas as categorias relevantes de deficiência** (motora, cognitiva etc.) entre os participantes dos testes de jogo, em números representativos com base na idade e dados demográficos do público-alvo
  - Permitir que o jogo possa ser aperfeiçoado, expondo tantas variáveis quanto possível
  - Permitir salvar configurações de diferentes perfis, em qualquer jogo ou plataforma

[CDAJ.2] para usuários com DEFICIÊNCIA MOTORA

- Básico
  - Que os controles possam ser remapeados / reconfigurados
  - Certificar que todas as áreas da interface do usuário possam ser acessadas usando o mesmo método de entrada com mesma jogabilidade
  - Incluir uma opção para ajustar a sensibilidade dos controles
  - Garantir controles tão simples quanto possível, ou fornecer uma alternativa mais simples
  - Certificar que os elementos interativos / controles virtuais são grandes e bem espaçados, especialmente em telas pequenas ou sensíveis ao toque
- Intermediário
  - Suportar mais do que um dispositivo de entrada
  - Faça elementos interativos que exigem precisão (ex. cursor/toque controlados pelas opções do menu) estacionária
  - Certificar que várias ações simultâneas (ex. clicar/arrastar ou deslizar) não sejam necessárias, e incluir apenas como um método de entrada alternativa / complementar
  - Certificar que todas as ações chave possam ser realizadas por controles digitais (pad / chaves / ação de pressionar), com mais de uma entrada complexa (ex. discurso, gesto) não obrigatória, e incluir apenas como métodos de entrada suplementares/alternativas
  - Incluir uma opção para ajustar a velocidade do jogo
  - Evitar entradas repetidas (eventos em tempo rápido)
  - Se produzir um jogo para PC, apoiar o modo de janela para compatibilidade com teclados virtuais sobrepostos
  - Evitar/fornecer alternativas de exigência de botões a serem pressionados
  - Permitir que as interfaces possam ser reorganizadas
  - Permitir que as interfaces possam ser redimensionadas
- Avançado
  - Permitir que o jogo possa ser colocado em modos retrato e paisagem
  - Não faça que o sincronismo preciso seja essencial para a jogabilidade - oferecer alternativas, ações que possam ser realizadas durante a pausa, ou um mecanismo de salto
  - Incluir um período (pós atraso de aceitação) de 0,5 segundos entre as entradas
  - Fornecer esquemas de controle muito simples que são compatíveis com dispositivos de Tecnologia Assistiva, tais como interruptor ou *eye tracking*

[CDAJ.3] para usuários com DEFICIÊNCIA COGNITIVA

- Básico
  - Permitir que o jogo seja iniciado sem a necessidade de navegar através de múltiplos níveis de menus
  - Usar um tamanho de fonte padrão de fácil leitura
  - Usar uma linguagem simples e clara
  - Usar a formatação de texto simples e clara
  - Incluir tutoriais
  - Permitir aos jogadores que possam progredir através de interface de texto ao seu próprio ritmo
  - Evitar piscar imagens e padrões repetitivos
- Intermediário
  - Incluir ajuda contextual no jogo / orientações / dicas
  - Indicar / permitir lembrete dos objetivos atuais durante o jogo
  - Indicar / permitir lembrete de controles durante o jogo
  - Incluir um meio de praticar sem falhas, tais como um nível de prática ou *sandbox*
  - Empregar uma estrutura narrativa simples, clara
  - Se estiver usando uma narrativa global longa, fornecer resumos de progresso
  - Garantir que não haja informações essenciais (especialmente instruções) que sejam transmitidas apenas via texto, reforçar com recursos visuais e/ou voz
  - Dar uma clara indicação de que elementos interativos são interativos
  - Fornecer uma opção para desligar/ocultar movimento de fundo
  - Possibilitar chat via voz e texto para jogos *multiplayer*
  - Fornecer imagens do jogo (*thumbnails*) com os jogos salvos
  - Fornecer controles de volume separados ou de mudo para efeitos, fala e música de fundo
  - Certificar que escolhas de som/música para cada objeto-chave / eventos são distintos um do outro
  - Incluir uma opção para ajustar a velocidade do jogo
  - Fornecer uma escolha de cor de texto, escolha de baixo/alto contraste, no mínimo
- Avançado
  - Fornecer dublagens pré-gravadas para todo o texto, incluindo menus e instaladores

  - Evitar qualquer movimento ou eventos súbitos
  - Permitir que todas as narrativas e instruções possam ser repetidas
  - Usar chat baseado em símbolo (*smileys* etc.)
  - Fornecer uma opção para desligar/ocultar todos os elementos não interativos

[CDAJ.4] para usuários com DEFICIÊNCIA VISUAL

- Básico
  - Garantir que não haja nenhuma informação essencial transmitida por cores apenas
  - Se o jogo utiliza o campo de visão (*engine* 3D apenas), definir um padrão apropriado para o ambiente de visualização esperado
  - Usar um tamanho de fonte padrão de fácil leitura
  - Usar a formatação de texto simples e clara
  - Fornecer alto contraste entre o texto e o fundo
  - Certificar que elementos interativos/controles virtuais são grandes e bem espaçados, especialmente em telas pequenas ou sensíveis ao toque
- Intermediário
  - Se o jogo utiliza o campo de visão (*engine* 3D apenas), permitir um meio para que ele possa ser ajustado
  - Evitar (ou fornecer opção para desativar) qualquer diferença entre o movimento do controlador e o movimento da câmera, como arma/balanço do caminhar ou a suavização do mouse
  - Usar o som *surround*
  - Fornecer uma opção para desligar/esconder a animação de fundo
  - Garantir o apoio do leitor de tela para dispositivos móveis
  - Fornecer uma opção para ajustar o contraste
  - Certificar escolher som/música para objetos-chave / eventos são distintos um do outro
  - Fornecer uma escolha de cursor / conjunto de cores / design
  - Dar uma clara indicação de que elementos interativos são interativos
  - Assegurar que o manual / website são fornecidos em um formato amigável ao leitor de tela
  - Fornecer controles de volume separados ou mudos para efeitos, de fala e de música de fundo
  - Evitar colocar informações temporárias essenciais fora da linha do olho do jogador

  - Permitir interfaces redimensionáveis

- Avançado
  - Permitir que o tamanho da fonte seja ajustado
  - Fornecer um mapa de áudio *pingable*, estilo sonar
  - Fornecer dublagens pré-gravadas para todo o texto, incluindo menus e instaladores
  - Fornecer um GPS sonoro
  - Permitir orientação fácil / movimento ao longo dos pontos cardeais
  - Certifique-se de que todas as ações chave possam ser realizadas por controles digitais (pad / chaves / ação de pressionar), com mais de uma entrada complexa (ex. discurso, gesto) não obrigatória, e incluir apenas como métodos de entrada suplementares/alternativas
  - Garantir o apoio do leitor de tela, incluindo menus e instaladores
  - Use design de som / música distinta para todos os objetos e eventos
  - Simular a gravação binaural

[CDAJ.5] para usuários com DEFICIÊNCIA AUDITIVA

- Básico
  - Fornecer controles de volume separados ou mudos para efeitos, de expressão e fundo de música
  - Garantir que não haja informação essencial transmitida apenas por áudio, reforçar com texto / visual
  - Se quaisquer legendas forem fornecidas, utilize um tamanho de fonte padrão de fácil leitura, formatação de texto simples e clara e forneça alto contraste entre o texto e o fundo

- Intermediário
  - Manter o ruído de fundo mínimo durante o discurso
  - Fornecer uma alternativa em texto para toda a fala (legendas)
  - Permitir alternativas em texto a serem exibidas antes de qualquer som reproduzido
  - Fornecer uma descrição em texto da narrativa / atmosfera significativa, com sons de fundo
  - Fornecer uma indicação visual de quem está falando
  - Possibilitar chat via voz e texto para jogos *multiplayer*
  - Fornecer meios visuais de comunicação no *multiplayer*

  - Permitir uma preferência para ser definida a jogabilidade *multiplayer* online, com ou sem outros que estão usando recursos de acessibilidade que poderiam dar uma vantagem competitiva
  - Garantir que todas as informações suplementares importantes (por exemplo: a direção que você está sendo filmado) veiculada por áudio são reproduzidas em texto / visuais
  - Proporcionar uma alternância entre estéreo / mono

- Avançado

  - Certificar que as legendas são cortadas e apresentadas em uma frequência apropriada para a faixa etária alvo
  - Fornecer cadastro

[CDA.6] para usuários com DEFICIÊNCIA de FALA

- Básico

  - Certificar que a entrada de fala não é necessária, e é incluída apenas como um método de entrada alternativa / complementar

- Intermediário

  - Permitir uma preferência para ser definida a jogabilidade *multiplayer* online, com ou sem outros que estão usando recursos de acessibilidade que poderiam dar uma vantagem competitiva
  - Possibilitar *chat* via voz e texto para jogos *multiplayer*
  - Fornecer meios visuais de comunicação no *multiplayer*
  - Reconhecimento de fala, com base em palavras individuais, a partir de um pequeno vocabulário (ex. 'sim', 'não', 'abrir') em vez de frases longas ou palavras multi-silábicas

- Avançado

  - Para o reconhecimento de fala tomar por base chegar a um limite de volume (por exemplo, 50%) em vez de palavras

Tais diretrizes norteiam a composição de jogos adaptados às inúmeras dificuldades inerentes às pessoas com deficiência e mesmo aos idosos, que muitas vezes apresentam declínios em suas funções cognitivas e motoras.

3.3. Coletando dados por meio de um jogo: Um exemplo

O estudo de caso aqui descrito exemplifica uma coleta de dados realizada por meio de jogos. Nesse exemplo, a coleta de dados desempenha papel de vital importância na pesquisa e foi realizada como parte de uma investigação sobre a conversão de baterias de testes cognitivos para o meio digital. O aplicativo de teste foi desenvolvido na forma de um jogo. O teste cognitivo digital foi submetido a um experimento com usuários, com base nas heurísticas de usabilidade [de Lima Salgado et al. 2017]. Heurísticas são processos que visam encontrar problemas que recorrentemente ocorrem, e em geral, são reconhecidas as possíveis soluções.

Em virtude de o usuário final do teste cognitivo digital ser o público idoso, foram consideradas questões de acessibilidade desde seu planejamento. Um exemplo foi a escolha da paleta de cores, a qual foi feita respeitando sua compreensão por indivíduos que apresentem daltonismo, mais especificamente nas 3 possíveis situações[4]: Tritanopia, Protanopia e Deuteranopia.

A ferramenta Color Oracle[5] foi utilizada para averiguar a paleta nas três situações e então foram eliminadas cores que em um ou mais casos se aproximavam muito umas das outras, se tornando quase indistinguíveis. Mesmo assim ainda é necessária cautela ao confeccionar os *assets* do jogo, isto é, *feedbacks* e objetivos não podem contar apenas com a cor para diferenciar objetos. Sons e formas também devem ser utilizados.

Na bateria de testes cognitivos, foram tratados oito aspectos relacionados com as exigências intelectuais dos usuários, a saber: (**Teste 01**) - Percepção Visual, Memória Incidental e Memória Imediata, (**Teste 02**) - Praxia, (**Teste 03**) - Abstração, (**Teste 04**) - Teste de Extensão de Dígitos (Direto), (**Teste 05**) - Teste de Extensão de Dígitos (Inverso), (**Teste 06**) - Atenção, (**Teste 07**) - Teste de Fluência Verbal e (**Teste 08**) - Reconhecimento Facial. A partir da definição desses aspectos, buscou-se uma correlação entre eles, a fim de se identificar / comprovar o nível de alcance que a portabilidade para o meio digital permitiria, sem que houvesse uma alteração significativa no contexto.

As coletas de dados no Estudo de Caso foram realizadas, tendo um conjunto de variáveis catalogadas, referentes aos aspectos relacionados com as cargas cognitivas requeridas dos usuários. A seguir, são listadas variáveis catalogadas, que exemplificam como foi planejado o experimento com usuários usando o jogo (jogando).

1. **(Teste 01) - Percepção Visual, Memória Incidental e Memória Imediata**

   (a) Quantidade de Objetos: número de objetos que aparecem na tela simultaneamente. Esta variável pode ter impacto direto no desempenho relativo de um mesmo jogador. Por exemplo, uma iteração com menos objetos na tela pode ter melhores resultados.

   (b) Quantidade de Formas: número de diferentes formas que aparecem na tela. Esta variável pode ter impacto direto no desempenho relativo de um mesmo

[4]Os tipos de Daltonismo analisados foram Trinatopia - insensibilidade às ondas curtas (os azuis); Protanopia - insensibilidade às ondas longas (os vermelhos); Deuteranopia - insensibilidade às ondas médias (os verdes)

[5]hyyp://www.colororacle.org/

jogador. Por exemplo, um teste onde apareçam triângulos e quadrados pode ter resultados inferiores a um em que apareçam apenas quadrados.

(c) Elementos de Confusão: número de objetos que serão inseridos para criar confusão, relevante apenas na versão anterior do teste, em que o jogador precisava marcar quais objetos eram diferentes da matriz memorizada. Esta variável pode ter impacto direto no desempenho relativo do mesmo jogador. Espera-se que em um cenário com mais elementos de confusão o jogador cometa mais erros.

(d) *Score*: pontuação do jogador, de 0 a 100. Este dado armazena, apenas, quantos dos espaços foram preenchidos corretamente, ignorando espaços que foram preenchidos sem necessidade. Para contornar a possibilidade do jogador preencher toda a matriz o número de objetos que podem ser colocados foi limitado para o mesmo número da variável Quantidade de Objetos. Este dado é importante para se comparar a matriz inserida pelo jogador e a matriz original, avaliando se o jogador de fato memorizou a matriz original.

(e) Tempo Total: soma do tempo desde que o jogador iniciou o teste até concluí-lo. Este dado pode ser um indicativo de dificuldade para interagir com a tela do tablet, dificuldade de compreensão do teste bem como limitações motoras ou distrações.

(f) Cliques:

i. Forma Clicada: dado que armazena qual foi a forma tocada pelo usuário, relevante apenas em casos com mais de uma forma disponível. Este dado pode indicar dificuldades para interagir com a tela do tablet, dificuldade de compreensão do teste bem como limitações motoras.

ii. Cor: dado que armazena a cor do objeto tocado, relevante apenas em casos em que haverá mais de uma cor de objeto. Este dado pode indicar dificuldade para distinguir as cores ou dificuldade de compreensão do teste.

iii. Tempo: tempo entre o clique atual e o anterior. Este dado pode indicar dificuldades para interagir com a tela do tablet, dificuldade de compreensão do teste, bem como limitações motoras.

iv. Tipo de clique: dado que marca se o jogador tocou um botão, um objeto de interesse ou nenhum dos dois anteriores, configurando então um *missclick*. Este dado pode indicar dificuldades para interagir com a tela do tablet, objetos muito pequenos e limitações motoras.

v. Acerto: dado que marca se o toque feito foi um acerto ou um erro, dentro das definições de acerto e erro do teste. Este dado pode indicar dificuldades para interagir com a tela do tablet, dificuldade de compreensão do teste bem como limitações motoras.

2. **(Teste 02) - Praxia**

(a) *Score*: pontuação de 0 a 100, marcando quantos objetos foram encaixados da forma correta. Este dado armazena a avaliação das conexões dois a dois, com

pequena margem de erro para cada conexão, dividindo a pontuação máxima pelo número de conexões totais que serão verificadas. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, dificuldade de compreensão do teste bem como limitações motoras.

(b) Tempo Total: soma do tempo desde que o jogador iniciou o teste até concluí-lo. Este dado pode ser um indicativo de dificuldade para interagir com a tela do tablet, dificuldade de compreensão do teste bem como limitações motoras ou distrações.

(c) Cliques:

i. Forma: a forma tocada pelo jogador. Este dado pode indicar que o jogador tem maior dificuldade de interagir com uma certa peça do Tangram, seja pelo seu tamanho ou formato.

ii. Cor: a cor da forma tocada pelo jogador, importante pois existem mais de uma cópia de algumas formas, mas de cores e/ou rotações diferentes. Este dado pode indicar que o jogador tem dificuldade em distinguir certas cores.

iii. Tempo: tempo entre o clique atual e o anterior. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, dificuldade de compreensão do teste bem como limitações motoras.

iv. Tipo de clique: este dado marca se o jogador tocou um botão, um objeto de interesse ou nenhum dos dois anteriores, configurando então um *missclick*. Este dado pode indicar dificuldades para interagir com a tela do tablet, objetos muito pequenos e limitações motoras.

3. **(Teste 03) – Abstração**

(a) *Score*: pontuação de 0 a 100, marcando quantas iterações do teste de abstração foram respondidas corretamente. Este dado pode indicar dificuldades para interagir com a tela do tablet, dificuldade de compreensão do teste bem como limitações motoras.

(b) Tempo Total: soma do tempo desde que o jogador iniciou o teste até concluí-lo. Este dado pode ser um indicativo de dificuldade para interagir com a tela do tablet, dificuldade de compreensão do teste, bem como limitações motoras ou distrações.

(c) Cliques:

i. Tempo: tempo entre o clique atual e o anterior. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, dificuldade de compreensão do teste bem como limitações motoras.

ii. Tipo de clique: este dado marca se o jogador tocou um botão, um objeto de interesse ou nenhum dos dois anteriores, configurando então um *missclick*. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, objetos muito pequenos e limitações motoras.

iii. Acerto: este dado marca se o toque feito foi um acerto ou um erro, dentro das definições de acerto e erro do teste. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, dificuldade de compreensão do teste bem como limitações motoras.

4. **(Teste 06) – Atenção**

   (a) Tamanho da Sequência: varíavel que armazena quão longa é a sequência apresentada na tela. Esta variável pode ter impacto direto no desempenho relativo de um mesmo jogador. Por exemplo, sequências mais longa podem causar maior confusão ao reposicionar os diversos números pela tela.

   (b) Total de Erros: dado que armazena quantas vezes o jogador tocou em um número que não era o próximo na sequência. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, dificuldade de compreensão do teste bem como limitações motoras.

   (c) Tempo Total: soma do tempo desde que o jogador iniciou o teste até concluí-lo. Este dado pode ser um indicativo de dificuldade para interagir com a tela do *tablet*, dificuldade de compreensão do teste bem como limitações motoras ou distrações.

   (d) Cliques:

      i. Número Pressionado: dado que armazena qual foi o dígito pressionado. Este dado pode indicar dificuldade de compreensão do teste e confusão com os números mostrados na tela.

      ii. Tempo: tempo entre o clique atual e o anterior. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, dificuldade de compreensão do teste bem como limitações motoras.

      iii. Acerto: este dado marca se o toque feito foi um acerto ou um erro, dentro das definições de acerto e erro do teste. Este dado pode indicar dificuldades para interagir com a tela do *tablet*, dificuldade de compreensão do teste, distração bem como limitações motoras.

3.4. Desenvolvimento de jogos para coleta de dados utilizando *Construct 2*

Como pode ser observado no estudo de caso da Seção3.3, apresentado como exemplo, é grande a quantidade de aspectos que podem ser estudados nos trabalhos científicos experimentais, bem como as suas variáveis catalogadas, que podem variar bastante dependendo do teste e do objetivo que se deseja alcançar ao realizar procedimentos de coleta de dados.

Muitas vezes, a falta de familiaridade com ferramentas, técnicas e linguagens utilizadas no desenvolvimento de jogos, bem como o desconhecimento dos elementos relacionados à *gamificação*, podem desencorajar profissionais e pesquisadores envolvidos com investigações experimentais a utilizarem formas alternativas para coleta de dados, em seus estudos de casos por exemplo. Abordagens envolvendo estudo de caso e jogos foram publicadas na literatura, recentemente [Barendregt et al. 2017] e [Sung and Berland 2017].

Segundo [Samodelkin et al. 2016], é cada vez mais importante explorar novas formas de interação, em particular, para coleta de dados de experiências de usuários. Nesse sentido, [Wood et al. 2017] afirmam que o desenvolvimento de jogos e o uso de *gamificação* permitem atingir uma parcela significativa de usuários, principalmente usuários mais jovens, uma vez que o contato direto com jogos pode ocorrer com maior frequência em seu dia a dia. A adoção de elementos de *gamificação* em estudos de casos, não necessariamente relacionados com a área de jogos, pode fornecer novas perspectivas de coleta de dados, novas experiências aos usuários e novos métodos para realização de pesquisas acadêmicas.

Nesse contexto, surge a necessidade de uma ferramenta que satisfaça aos seguintes requisitos: (a) ofereça diversas possibilidades de apoio ao desenvolvimento de jogos, à manipulação de elementos de *gamificação* de forma intuitiva, (b) que seja de fácil uso mesmo para pesquisadores sem experiências prévias com a área de jogos e, principalmente, (c) que possibilite o *design* de estudos de casos, bem como a coleta e armazenamento de variáveis para futuras análises.

A ferramenta *Construct 2* foi estudada, pois auxilia no desenvolvimento de jogos, e atende aos três requisitos especificados. De propriedade da empresa *Scirra*[6], seu lançamento ocorreu no ano de 2011 e um exemplo de sua interface pode ser visualizada na Figura 3.1.

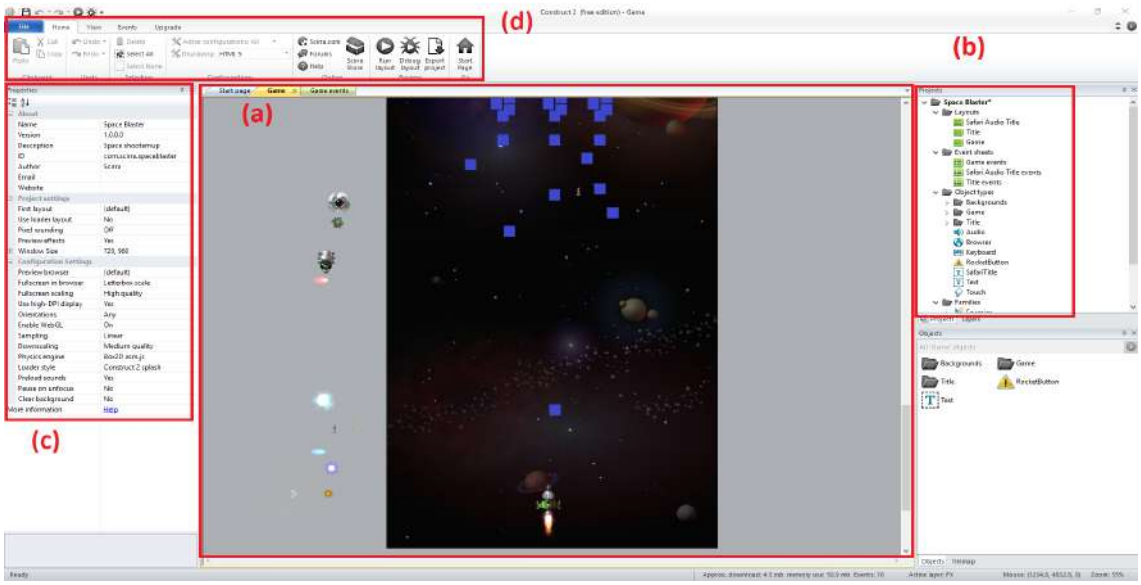


Figura 3.1. Visão geral da ferramenta *Construct 2*: (a) cenário, (b) lista de objetos (imagens, sons, controles), (c) painel de configuração de comportamentos e (d) menu de opções

Na Figura 3.1, a região em destaque rotulada por **(a)** mostra os elementos gráficos, chamados de *sprites*, que irão compor o jogo. Podem ser adicionados ao *layout* e manipulados, formando o cenário no qual o jogo irá ser executado. A região no destaque **(b)** mostra que é possível adicionar facilmente outros elementos como som, vídeo, controles

[6]https://www.scirra.com/

e etc. Na região de destaque **(c)**, encontram-se as diversas opções de configuração que permitem ajustar o comportamento que cada elemento presente no jogo irá receber. Por fim, no destaque **(d)** está o menu principal que permite compilar e testar o jogo bem como realizar a exportação do projeto criado.

Uma das principais características da ferramenta *Construct 2* se refere a não impor ao desenvolvedor o conhecimento prévio de alguma técnica ou linguagem de programação, tornando-se assim uma ferramenta ideal para profissionais e pesquisadores sem experiência prévia no desenvolvimento de jogos. Sua programação ocorre por meio de eventos que informam ao sistema como será a relação e o comportamento que todos os elementos presentes no jogo devem/podem realizar, além da interação com variáveis e controles. Com essa ferramenta, também é possível exportar o jogo para múltiplas plataformas, como: *Web*, *mobile* e *desktop*, além de comunicar com banco de dados para armazenamento de variáveis. Na Figura 3.2 encontra-se um exemplo da programação por eventos, na ferramenta *Construct 2*.

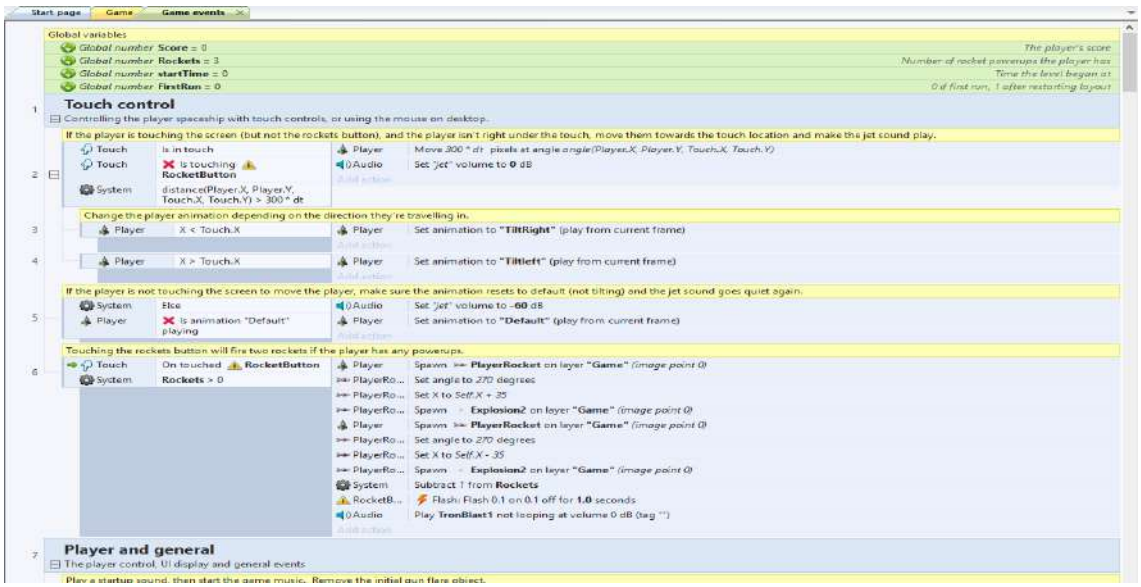


Figura 3.2. Interface que lista os eventos necessários para execução de um jogo em *Construct 2*

Devido ao foco deste capítulo estar em fornecer possibilidades de concepção de jogos orientados à coleta de dados, não serão abordados elementos básicos como o *download* e instalação. Assim, será apresentado um conjunto de exemplos nas Seções a seguir, os quais, a partir de uma abordagem para auxiliar pesquisadores no desenvolvimento de seus próprios experimentos, possibilita a adoção dos elementos de jogos. A abordagem está representada na Figura 3.3 e os elementos que a compõem são os seguintes:

1. ***Design* do experimento:** se refere à concepção dos elementos do jogo a ser construído, deverá ser realizada durante o desenvolvimento do jogo na ferramenta *Construct 2*, na fase inicial. Para auxiliar o pesquisador nesse *design* do seu experimento, deve-se considerar os conceitos teóricos sobre jogos, a experiência do usuário e diretrizes de acessibilidade para jogos, como apresentados na Seção 3.2. Na Seção

3.3 foi apresentado um exemplo de coleta de dados, por meio de heurísticas de usabilidade.

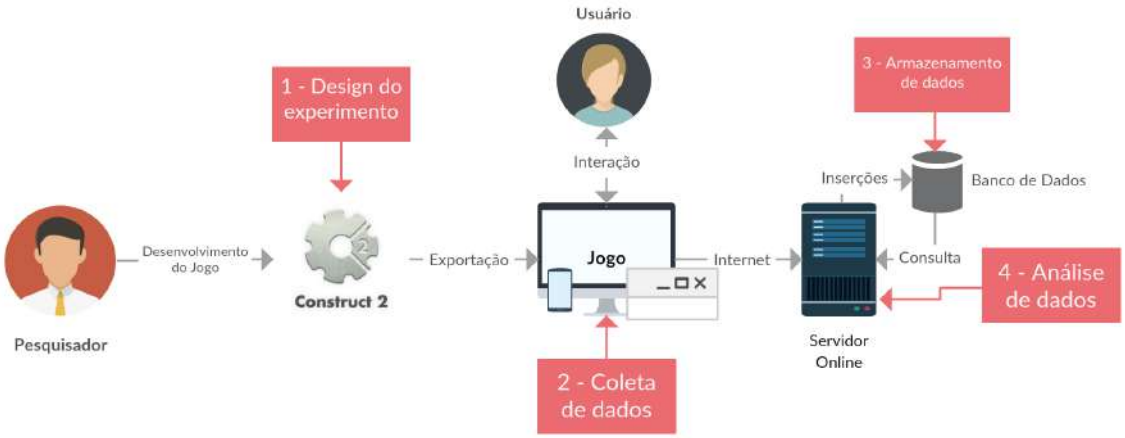


Figura 3.3. Abordagem para auxiliar pesquisadores no desenvolvimento de seus experimentos, envolvendo jogos e coleta de dados

2. **Coleta de dados:** ocorre durante a interação do usuário com o jogo, o qual pode ser projetado para multiplataformas (*Web, mobile* e *desktop*). A Seção 3.4.1 apresenta o uso de recursos multimídia para coleta de dados e a Seção 3.4.2 descreve a coleta de dados referente a variáveis quantitativas e qualitativas.

3. **Armazenamento de dados coletados:** para que o pesquisador possa realizar análises sobre os dados coletados, é importante que haja armazenamento dos dados em banco de dados hospedado em um servidor *online*. A Seção 3.4.3 descreve um jogo que utiliza diretrizes relativas a habilidade motora, para exemplificar o processo de armazenamento.

4. **Análise de dados:** após os dados coletados terem sido armazenados em um banco de dados, consultas aos dados podem ser efetuadas pelo pesquisador utilizando o servidor *online*. Como este tópico requer estudos mais aprofundados, recomenda-se as orientações de [MacKenzie 2012] para realização das análises estatísticas dos dados coletados.

3.4.1. Utilizando recursos multimídia na coleta de dados

3.4.1.1. Coleta do áudio do usuário para reconhecimento de voz

Na Seção 3.2.3 foram descritas diretrizes de acessibilidade que devem ser utilizadas também para melhorar a experiência do usuário. Dentre elas, utilizando como exemplo a habilidade de "Fala", uma diretriz, de categoria intermediaria, recomenda que: "o reconhecimento de fala deve ocorrer com base em palavras individuais, a partir de um pequeno vocabulário (ex. 'sim', 'não', 'abrir'), em vez de frases longas ou palavras multisilábicas". Com o intuito de coletar dados relacionados com a habilidade de "Fala", os seguintes eventos podem ser programados na ferramenta *Construct 2* como mostrados na Figura 3.4.

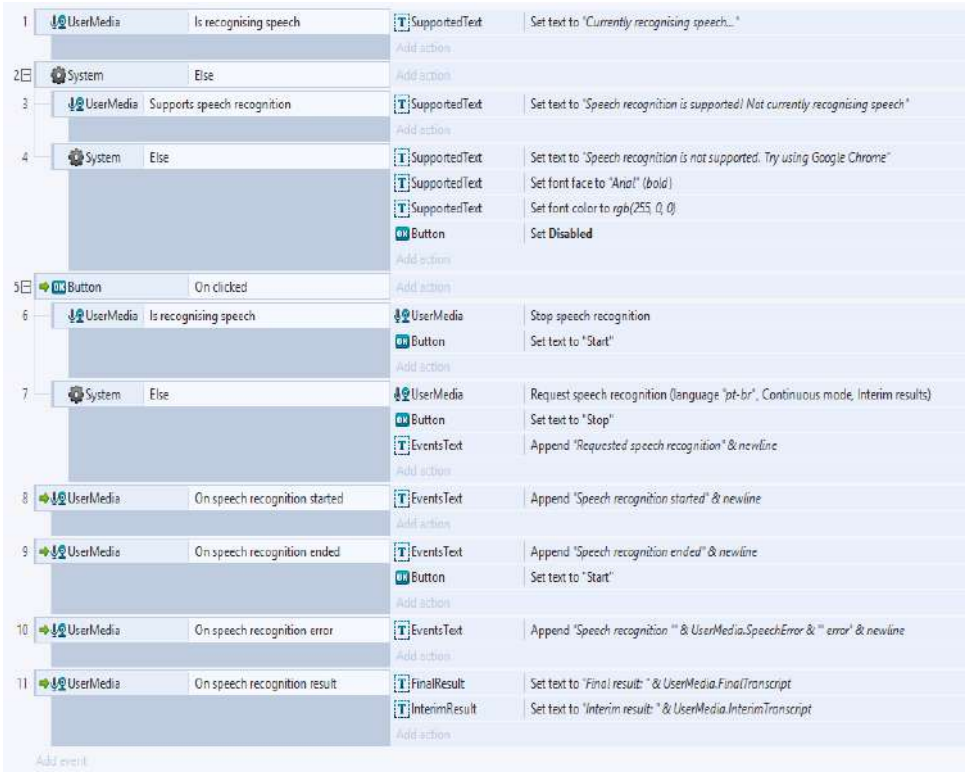


Figura 3.4. Programação em eventos sobre reconhecimento de fala

Após o projeto do jogo receber o objeto chamado *"UserMedia"*, os eventos listados na Figura 3.4 podem ser criados e manipulados. Na linha 1, o evento *"UserMedia is recognising speech"* define que o microfone deverá ser acessado sempre que o jogo for executado no navegador e fornece um *feedback* ao usuário com o texto *"Currently recognising speech..."*, o conteúdo do texto pode ser alterado ou substituído por uma imagem. Entre as linhas 2 e 4 são definidos os eventos, nos casos de não existir microfone ou o navegador não oferecer tal suporte.

Entre as linhas 5 e 7 está definido o evento principal de reconhecimento de fala, no qual existe a condição de clicar em um botão para iniciar ou pausar o reconhecimento. Na linha 7, a configuração *"Request speech recognition (language 'pt-br', Continuos mode, Interim results"* define o idioma utilizado no reconhecimento, bem como a configuração *"Continuos mode"* cumpre a diretriz, citada anteriormente, de reconhecer palavras individuais (para o reconhecimento de frases inteiras a configuração *"Single phrase mode"* deve ser utilizada). Na Figura 3.5 é apresentada a tela com o resultado do reconhecimento de fala.

3.4.1.2. Coleta do vídeo da tela de interação do usuário

Capturar a tela na qual ocorre a interação com um jogo pode ser um recurso valioso quando se trata de analisar a experiência do usuário. Segundo [MacKenzie 2012], a captura de tela é uma abordagem criativa para coletar dados, principalmente quando existe interação com algum *software*.

Figura 3.5. Resultado do reconhecimento de fala de palavras individuais

Um problema que pode inviabilizar a captura de tela, como recurso de coleta de dados, está nos *softwares* deste segmento, disponíveis no mercado. Em alguns casos, *softwares* especializados possuem licenças de uso, que podem inviabilizar pesquisas que não dispõem de verba ou exigem um nível de conhecimento específico para sua correta utilização.

A ferramenta *Construct 2* permite realizar a captura da tela, na qual o jogo está sendo executado, e possibilita, como saída, gerar um vídeo que pode ser gravado em formatos variados. A Figura 3.6 mostra a programação utilizada para tal objetivo.

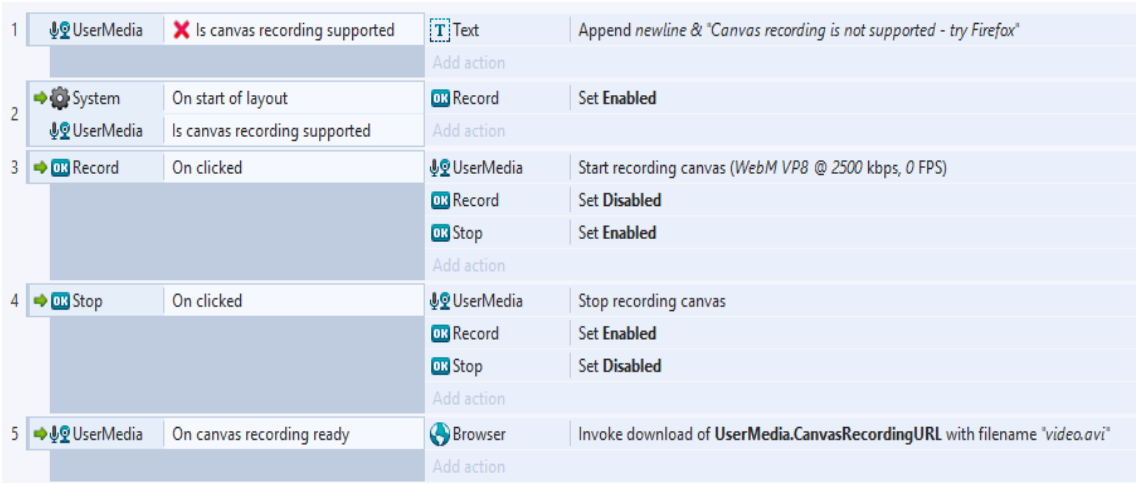


Figura 3.6. Programação de captura de tela na ferramenta *Construct 2*

Na linha 1 da Figura 3.6, é definido o teste de compatibilidade com o navegador; o *feedback* gerado no exemplo é dado de forma textual na interface, mas pode ser alterado.

A linha 2 informa ao sistema que, quando o jogo for iniciado (*"On start of layout"*) e o navegador oferecer suporte para captura de tela, o botão de gravação ficará ativo. Na linha 3, é configurado o evento que inicia a gravação, que está condicionado ao clique de um botão. Na linha 4, é prevista a ação de interromper a gravação por meio de um botão e, na linha 5, é previsto o *download* imediato do resultado da captura em formato de vídeo que pode ter seu formato (avi, mp4 e etc.) alterado.

Outro recurso que a ferramenta *Construct 2* possui é o que permite realizar a gravação do usuário jogando, utilizando uma ação do usuário como condição. Na Figura 3.7, por exemplo, a gravação somente ocorre quando o usuário obtiver 20 pontos e será interrompida quando o usuário atingir 30 pontos.

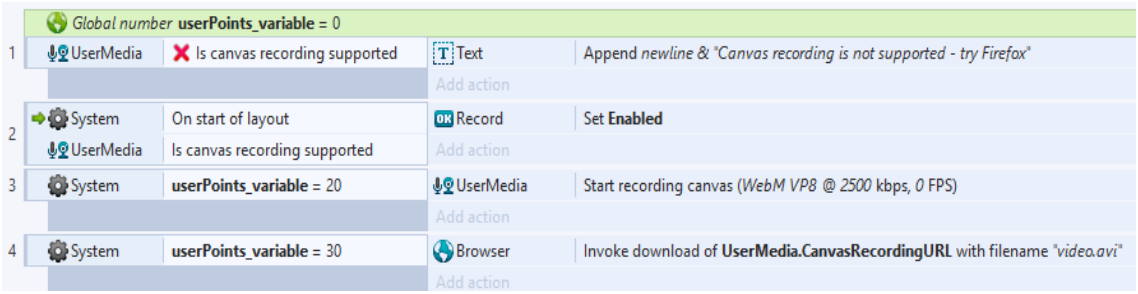


Figura 3.7. Exemplo de programação de captura de tela condicionada a ações do usuários

Na linha 3 da Figura 3.7, está a configuração para que o sistema apenas inicie a gravação quando o valor da variável *"userPoints_variable"* for igual a 20 e a linha 4 indica que a finalização da gravação ocorra quando a mesma variável atingir o valor 30. Programações como esta permitem maior flexibilidade para que estudos de caso possam ser realizados de modo mais preciso e alinhados aos dados que se espera obter.

3.4.1.3. Coleta de fotos durante a interação do usuário

Durante a interação de um usuário com o jogo, a ferramenta *Construct 2* oferece também a possibilidade de captura de fotos antes, durante e depois da coleta de dados. A gravação de vídeo do usuário, na versão disponível atualmente (r244), ainda não oferece tal suporte. A Figura 3.8 mostra um exemplo da programação necessária para o acesso da *webcam*.

Para a captura de fotos de um usuário, as seguintes linhas definidas na ferramenta *Construct 2*, apresentadas na Figura 3.8, mostram as principais definições a serem realizadas:

(i) Linha 1: verifica se existe uma *webcam* conectada e devidamente instalada quando o jogo é carregado.

(ii) Linhas 2 e 3: permitem a listagem e seleção das *webcams* detectadas.

(iii) Linha 4: permite, por meio de um botão, que a *webcam* selecionada anteriormente seja iniciada; também exibe a resolução detectada da *webcam*. Com esta programação já é possível visualizar no jogo a filmagem da *webcam* em tempo real.

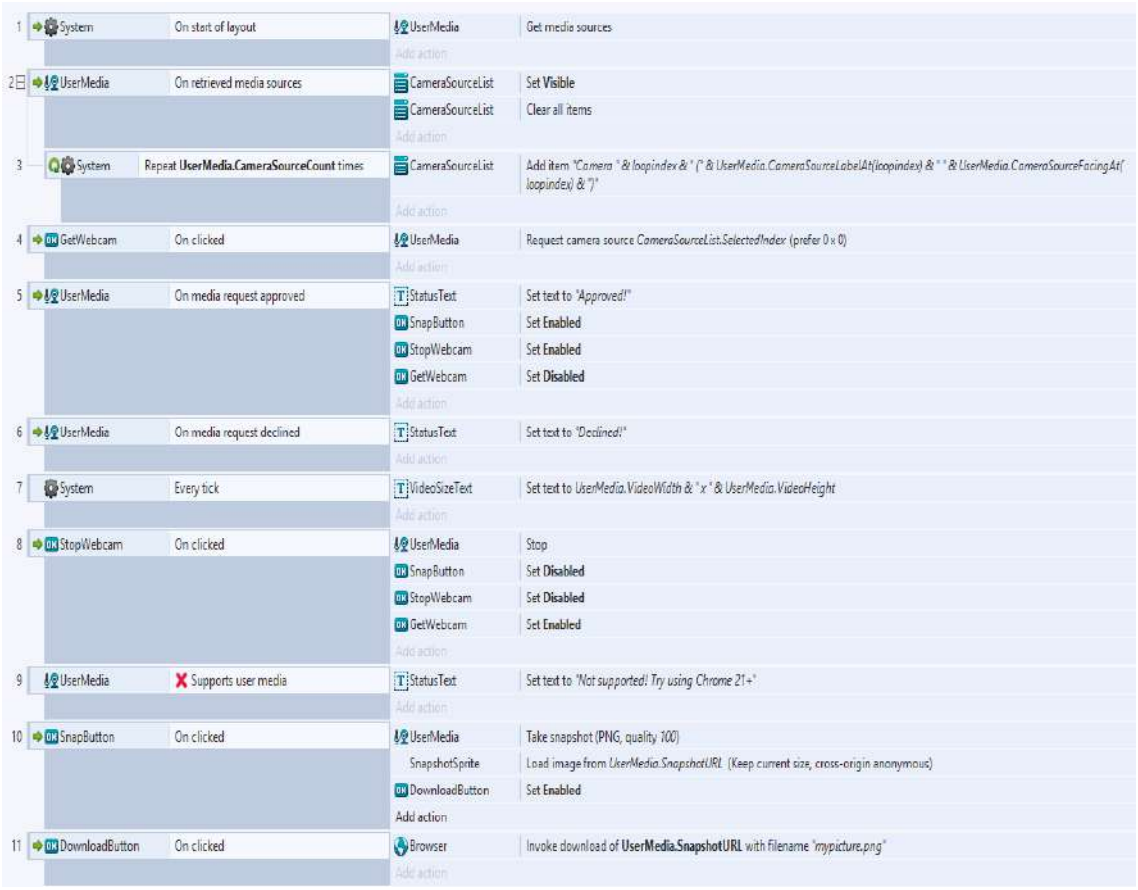


Figura 3.8. Programação utilizada na captura de fotos do usuário

(iv) Linha 5: caso a *webcam* seja reconhecida, ativa-se os botões que possibilitam tirar uma foto, e interromper o uso da *webcam*.

(v) Linha 6: exibe o *status* caso a *webcam* selecionada não esteja funcionando.

(vi) Linha 7: pede ao sistema que exiba o conteúdo da captura a todo momento.

(vii) Linha 8: prevê uma ação, no caso em que o botão que interrompe a gravação for acionado.

(viii) Linha 9: prevê uma ação, no caso em que o navegador não tenha suporte multimídia.

(ix) Linha 10: permite a captura da foto do usuário, por meio de um botão; também permite configurar a qualidade da imagem e armazena em memória.

(x) Linha 11: permite fazer o *download* da foto capturada anteriormente.

É possível também condicionar a captura da imagem do usuário, associada a uma ação. Essa possibilidade propicia a captura de um momento ou a sequência de momentos em que o usuário realiza determinação ação ou que alguma condição seja alcançada. Na

Figura 3.9, tem-se um exemplo de que uma foto do usuário é tirada, quando a variável *"userPoinst_variable"* é igual ao valor 10.

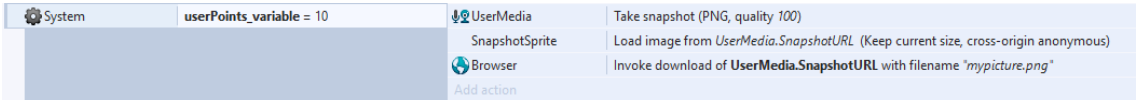


Figura 3.9. Exemplo de programação de captura de foto condicionada a ações do usuário.

3.4.2. Coletando dados de variáveis quantitativas e qualitativas

Segundo [MacKenzie 2012], uma atividade muito importante no planejamento de um experimento com usuários consiste na decisão sobre quais variáveis experimentais a pesquisa deverá considerar. Segundo o autor, pensar em quais variáveis experimentais podem ser obtidas e no *design* do experimento auxilia na transição de questões genéricas e muito amplas (e.g., *Será que o usuário vai gostar do meu jogo?)* para questões de pesquisa focadas e possíveis de serem testadas (e.g., *Uma tarefa pode ser realizada com mais rapidez ao comparar duas interfaces?)*

Nesta seção é apresentada a abordagem (seguindo o esquema apresentado na Figura 3.3) de desenvolvimento de jogos para a coleta de dados de experimento com usuário, por meio da ferramenta *Construct 2*, utilizando variáveis quantitativas e qualitativas. A seguir, tem-se a Seção 3.4.2.1 que apresenta um estudo de caso no qual são utilizados jogos para coleta de dados do usuário. Por meio deste estudo, é possível identificar exemplos de variáveis comumente utilizadas em pesquisas acadêmicas. A Seção 3.4.2.2 discute modos de utilizar a ferramenta *Construct 2* para replicar algumas etapas do estudo de caso apresentado, fornecendo ao desenvolvedor exemplos para realizar seus próprios experimentos e coletas de dados.

3.4.2.1. Planejando coleta de dados: Exemplo do jogo *Flappy Bird*

Utilizaremos como exemplo para o planejamento da coleta de dados, o jogo *Flappy Bird*. Lançado no ano de 2013, este jogo foi desenvolvido para dispositivos móveis, tornou-se muito popular principalmente pela característica de ser muito difícil, na opinião dos jogadores, que se sentiam desafiados a conquistar maiores pontuações.

Devido a sua popularidade, pesquisas utilizaram-no como base para compreender melhor a experiência do usuário. Uma dessas pesquisas, de [Isaksen et al. 2015], foi conduzida com os seguintes objetivos:

- Auxiliar *designers* a explorar os elementos presentes no espaço do jogo, como (*layouts*, cenários e *sprites*)
- Compreender melhor a relação entre *design* e experiência do usuário

A partir do foco definido nos objetivos elencados, o próximo passo da pesquisa consistiu em delimitar a análise estabelecendo uma questão de pesquisa, no contexto dos objetivos, que deve ser discutida e respondida.

Questão de pesquisa:
– Como os parâmetros, sem alteração das regras, podem afetar a dificuldade que um jogo possui?

Uma das formas encontradas para responder esta questão de pesquisa foi realizar um estudo de caso contendo as duas seguintes fases:

Fase 1 - Aplicação de Questionário - foco na coleta de variáveis independentes: sexo, idade, experiência como jogador e exposição ao jogo *Flappy Bird* e suas versões genéricas.

Fase 2 - Teste de Habilidade - foco na coleta de variáveis dependentes: precisão, tempo de resposta e ações por segundo.

3.4.2.2. Realizando coleta de dados: Exemplo do jogo *Flappy Bird*

Uma vez estabelecidas a questão de pesquisa e quais variáveis experimentais serão utilizadas na processo de coleta de dados, o próximo passo consiste no desenvolvimento do jogo. Utilizando o estudo de caso apresentado anteriormente, esta seção apresenta a abordagem para replicar a coleta de dados, utilizando a ferramenta *Construct 2*. Para replicar a coleta de dados, será utilizado um modelo genérico do jogo *Flappy Bird* que acompanha a versão gratuita da ferramenta, como mostrado na Figura 3.10.

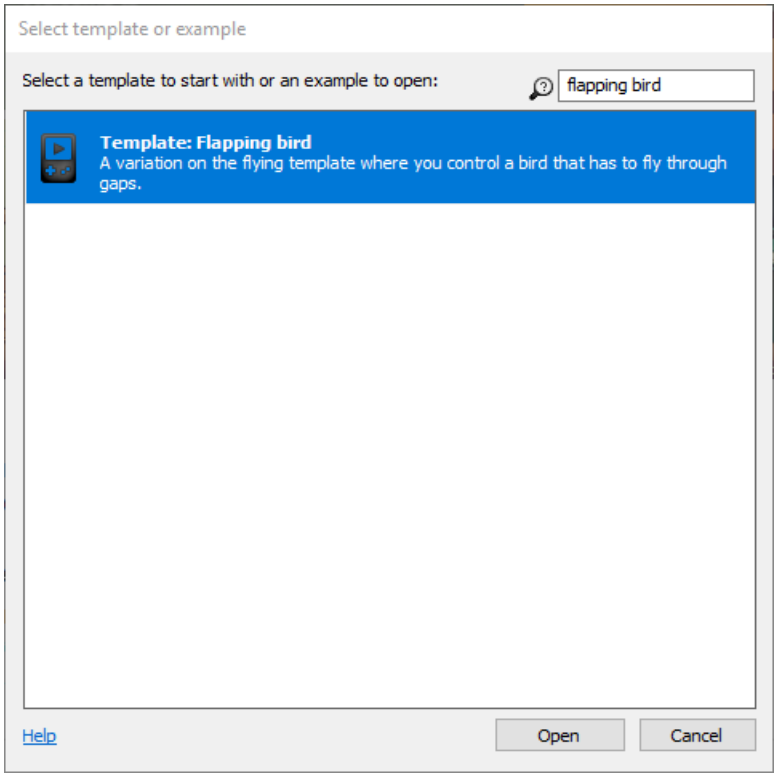


Figura 3.10. Seleção de *template* na ferramenta *Construct 2*

A partir de selecionado o *template* em *Construct 2*, a tela de desenvolvimento do referido jogo é mostrada (Figura 3.11), contendo os elementos necessários para o desenvolvimento do modo de coleta de dados nas variáveis.

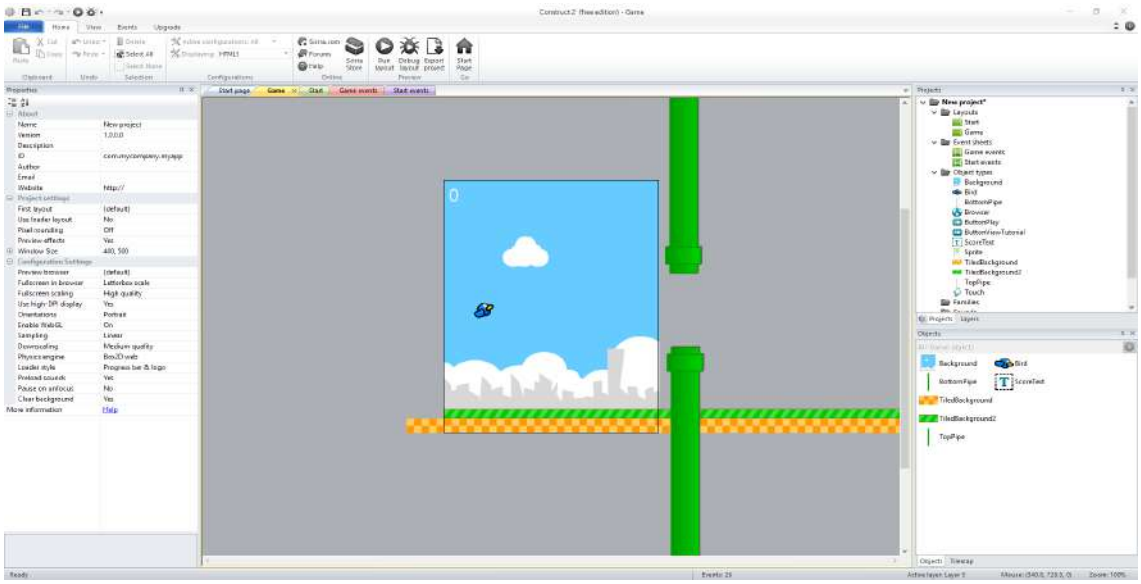


Figura 3.11. *Template Flapping bird* aberto no *Construct 2*

Fase 1 - Questionário

A primeira fase de coleta de dados, do exemplo citado anteriormente, utilizou as seguintes variáveis independentes: **sexo**, **idade**, **experiência como jogador** e **exposição ao jogo**. É muito comum que em estudos de caso, um questionário contendo perguntas sobre o perfil e opiniões dos usuários seja aplicado em algum momento.

Para coletar as mesmas variáveis na ferramenta *Construct 2*, deverá ser adicionado um novo *layout* ao *template*, no qual serão inseridos os campos em que o jogador deverá responder o questionário. Os campos adicionados podem ser observados na Figura 3.12.

Na três telas de interação pelo usuário (Figura 3.12), tem-se: item (a) apresenta ao usuário a tela inicial do jogo, item (b) são apresentadas as opções para coletar as informações relativas à fase de questionário. Nota-se que é possível inserir tanto alternativas preestabelecidas como na opção "Sexo"(masculino e feminino) e campos de texto como na opção "Idade"(anos). Para exemplificar as possibilidades de utilização nas opções "Experiência como jogador"e "Exposição ao *game*", foram oferecidas ao usuário opções listadas, utilizando a escala *Likert*, e também um campo de texto. Independente de qual estratégia o pesquisador irá adotar, para obter dados sobre o perfil do usuário, a ferramenta permite inserção de uma ou mais opções de coleta de dados.

No item (c) é apresentada a tela na qual o jogo será executado. Estratégias podem ser adotadas tanto na ordem sequencial como na repetição de telas de interação com o usuário. A ferramenta *Construct 2* permite criar condições que, ao serem satisfeitas, diferentes formulários ou telas podem ser apresentados ao usuário. Um exemplo dessa possibilidade é o fator tempo: após determinado tempo de jogo, automaticamente outro

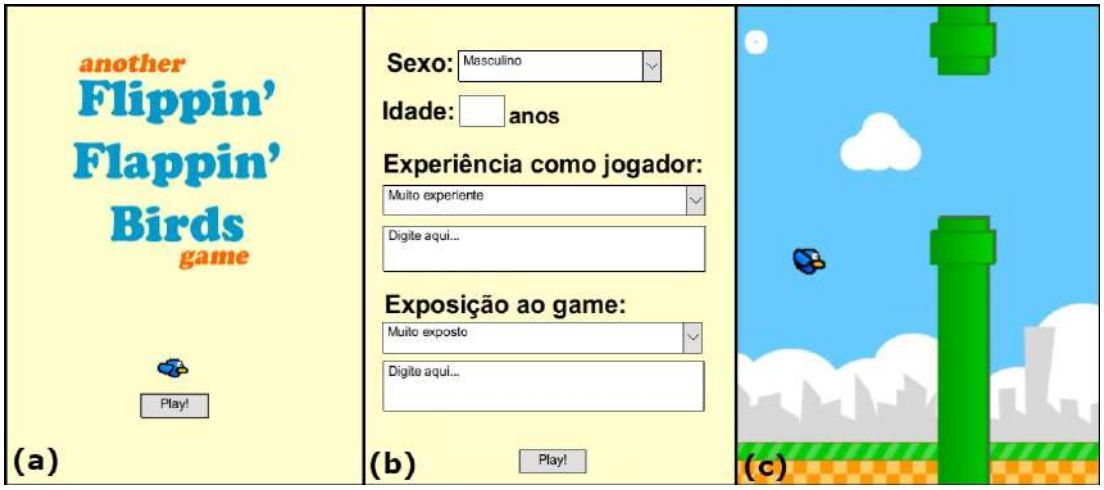


Figura 3.12. Sequências de interfaces do jogo: (a) tela inicial, (b) tela onde será coletado as informações do questionário e (c) tela do jogo

formulário de coleta de dados pode ser apresentado ao usuário. Outro exemplo está em cumprir certos objetivos ou tarefas durante a execução do jogo, dependendo dos resultados dos usuários, como coletar certa quantidade de moedas ou alcançar certa quantidade de pontos, formulários diferentes podem surgir aumentando assim a precisão na coleta de dados sobre a experiência do usuário.

Outra vantagem ao utilizar formulários para coleta de dados, durante a execução de jogos, se refere à possibilidade de utilizar as próprias informações inseridas pelo usuário para alterar a experiência de interação. Se um usuário se diz experiente em certo assunto, uma possibilidade (dependendo do que se deseja coletar) se encontra no nivelamento de dificuldade que pode ser ajustado para cada usuário e, assim, ao final dos experimentos, os dados coletados podem refletir, de forma mais fiel, a real experiência do usuário.

Fase 2 - Habilidade

A segunda fase de coleta de dados, do exemplo citado anteriormente, utilizou as variáveis dependentes: **precisão**, **tempo de resposta** e **ações por segundo**. Para essas variáveis, dados quantitativos devem ser devidamente coletados e apresentados de forma clara ao pesquisador para a devida análise posterior.

Coleta da variável "Precisão": para coletar os dados sobre precisão, aos usuários foi instruído que apenas clicassem em um botão sempre que uma linha, que se movia constantemente, estivesse horizontalmente alinhada a um alvo na tela. O teste foi repetido 20 vezes com três velocidades diferentes. Para verificar a possibilidade de a ferramenta replicar a mesma coleta de dados, o *template* foi modificado (Figura 3.13).

Na Figura 3.13, o item (a) mostra a posição do alvo vermelho adicionado juntamente a uma linha preta que realiza movimentos horizontais, no item (b) é apresentada a configuração utilizada para atribuir o comportamento (*Behavior*) de movimento da linha. Esse item é importante pois permite a fácil mudança da velocidade da linha, apenas indicando um valor no item *Period*. No item (c) é mostrada a posição do botão que deverá ser pressionado (no trabalho de [Isaksen et al. 2015] não é especificado qual o nome atribuído

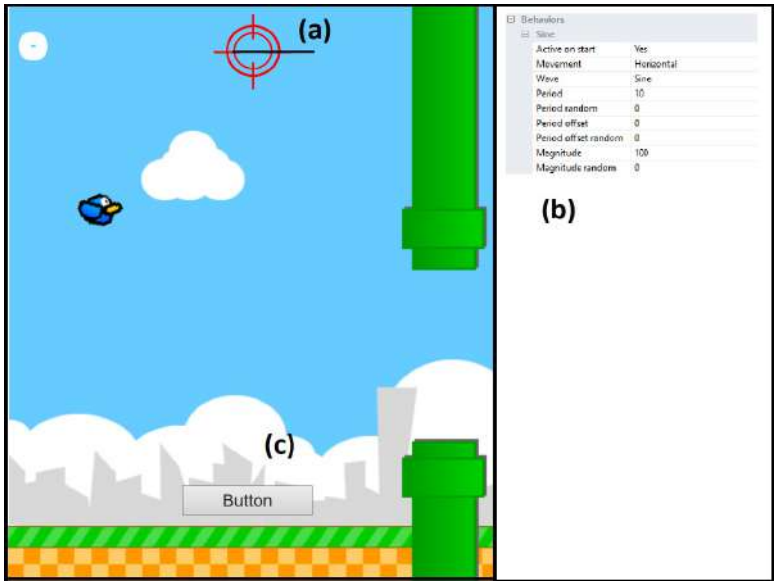


Figura 3.13. Elementos gráficos necessários para replicar coleta da variável precisão

ao botão). Esses elementos gráficos foram adicionados na interface do jogo, utilizando a ferramenta *Construct 2*, e podem ser reposicionados e arrastados facilmente com o uso do *mouse*, permitindo foco na coleta de dados e não na programação desse elementos por meio de alguma linguagem ou método específico. Para verificar se a regra de precisão foi respeitada, um programa foi adicionado ao exemplo (Figura 3.14).

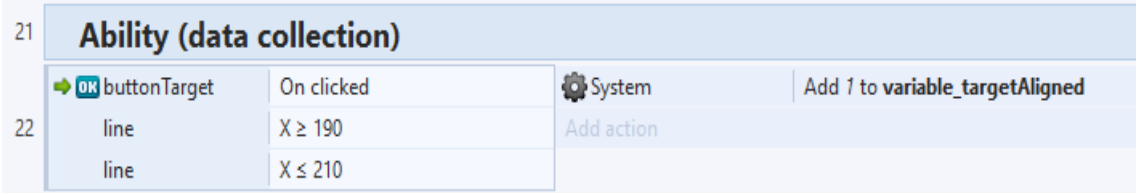


Figura 3.14. Programa “desenvolvido” para coleta da variável precisão

Uma condição foi definida na linha 22 da Figura 3.15, de forma que uma variável chamada *"variable_targetAligned"* apenas receba a adição de um valor, quando o botão *"buttonTarget"* for clicado e a linha *"line"* estiver entre os valores X >= 190 e X <= 210, valores estes que representam o alinhamento do alvo e da linha na interface do jogo. Esse intervalo de valores no eixo X pode ser configurado para maior flexibilidade do alinhamento da linha com o alvo. Assim, ao final da interação, é possível saber, com precisão, quantas vezes o usuário conseguiu realizar a condição de pressionar o botão no alinhamento da linha com o alvo.

Essa condição também pode ser reajustada durante a interação do usuário. Por exemplo, na linha 23, é possível aumentar a velocidade da linha *"line"* criando a seguinte condição "quando a variável *'variable_targetAkigned'* for maior que 5, o período de movimento será 8", como mostra a Figura 3.15.

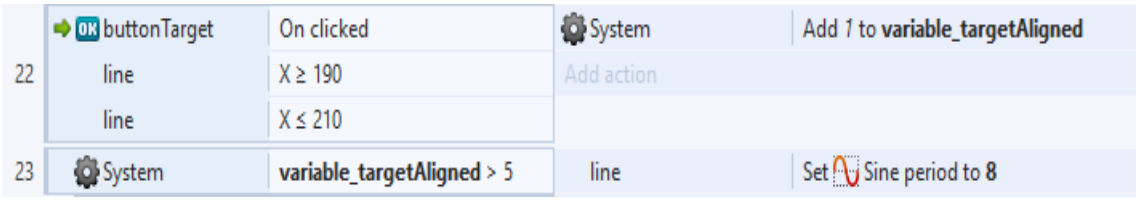


Figura 3.15. Programação realizada para coleta da variável precisão e condições de velocidade para o objeto linha

Coleta da variável "Tempo de reação": para medir o tempo de reação, os usuários foram instruídos a pressionar o botão tão rápido quanto visualizassem a linha horizontal ao lado direito da janela do jogo. A média de atraso, que permitiu calcular o tempo de reação, foi obtida utilizando o momento em que a linha se posicionava a direita da janela e o momento que o usuário pressionava o botão. Para coletar esta variável, uma programação foi adicionada, como mostra a Figura 3.16.

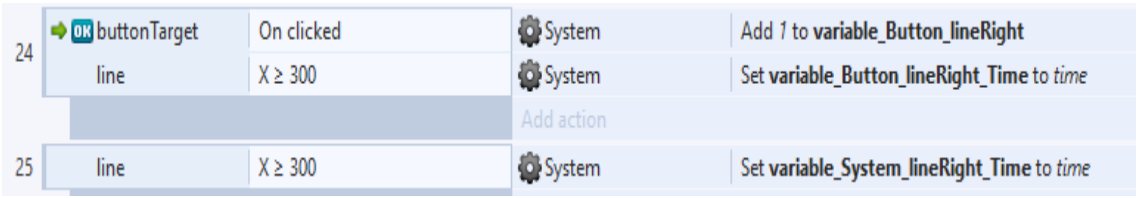


Figura 3.16. Programação realizada para verificar o *"click"* e a posição da linha horizontal

Na Figura 3.16, a linha 24 contém a programação que apenas adiciona um valor à variável *"variable_Button_lineRight"* e captura o tempo (segundos) na variável *"variable_Button_lineRight_Time"* quando a seguinte condição é satisfeita: "Quando o botão *"buttonTarget"* é clicado e a posição da linha *"line"* for maior ou igual a 300"(o valor do eixo X >= 30 indica se a linha está à direita da janela do jogo). Na programação da linha 25, o momento (tempo em segundos) em que a linha se encontra à direita da janela (x >= 300), é adicionado na variável *"variable_System_lineRight_Time"*, subtraindo as duas variáveis de tempo é possível calcular a média de atraso como objetivado no *design* da coleta do "Tempo de reação".

Coleta da variável "Ações por segundo": para coleta dos dados referentes a uma certa variável, os usuários foram instruídos a manter o botão pressionado por 10 segundos; no trabalho de [Isaksen et al. 2015] não foi especificado quando os usuários deveriam realizar esta tarefa. Para replicar esta coleta, foi adicionado ao exemplo, na linha 26, a seguinte programação: "se o botão *buttonTarget* for pressionado por 10 segundos, adicione um valor à variável *variable_touchButton*"(Figura 3.17).

A ferramenta *Construct 2* possui outras formas de ações relacionadas ao toque do tipo *"touch screen"*, permitindo ao pesquisador, que não possui familiarização com desenvolvimento de jogo, diferentes alternativas para coletar dados por meio da realização de tarefas.

Por meio da análise do trabalho de [Isaksen et al. 2015] e o uso da ferramenta

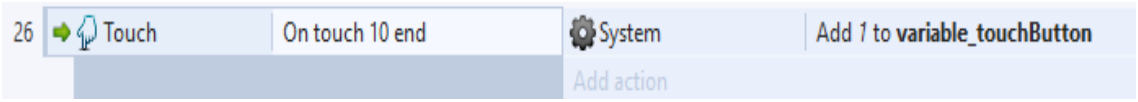


Figura 3.17. Programação realizada para verificar o tempo de toque no botão

de desenvolvimento *Construct 2*, foi possível replicar algumas etapas do estudo de caso, exemplificando uma abordagem para coleta de dados no contexto de jogos. Desta forma, pesquisadores que não possuem conhecimentos prévios, sobre desenvolvimento de jogos, dispõem de uma abordagem para replicar ou desenvolver experimentos para coleta de dados da experiência do usuário.

3.4.3. Armazenamento de dados coletados

Uma das características mais importantes na coleta de dados relativos à experiência do usuário está em desenvolver estratégias de armazenamento de dados gerados durante a interação, para análises posteriores. Os dados devem, portanto, ser armazenados de forma a permitir fácil manipulação e consulta posterior.

Nesta seção é descrito um modo de armazenamento dos dados coletados em uma base de dados *online*, durante a execução de um jogo desenvolvido por meio da ferramenta *Construct 2*. A possibilidade de armazenar dados em uma base de dados *online* visa assegurar que os dados estarão seguros e poderão ser acessados e analisados de qualquer lugar.

3.4.3.1. Exemplo de jogo com habilidade motora

Utilizando como exemplo a habilidade "Motora", na Seção 3.2.3 no [CDAJ.2] foi apresentada a seguinte diretriz, de categoria intermediaria, que recomenda: “faça elementos interativos que exigem precisão (ex. cursor/toque controlados pelas opções do menu) estacionária”. Utilizando essa diretriz como base, a Figura 3.18 mostra uma interface de jogo, na qual os usuários deverão arrastar, com auxilio do *mouse*, as maçãs até a cesta, permitindo coletar duas variáveis: a quantidade de maçãs e o tempo exato em que a maçã foi colocada na cesta.

A interface da Figura 3.18 foi desenvolvida utilizando a ferramenta *Construct 2*, para Web, e poderá ser acessada localmente ou hospedada em um servidor. Para armazenar as variáveis “quantidade de maçãs na cesta” e “tempo em que a maçã e colocada” foi necessário um servidor *online*, seguindo o esquema LAMP[7], um código na linguagem PHP para realizar o acesso e inserção no banco de dados.

Na Figura 3.19 é apresentado o código-fonte para criar a tabela chamada *“apples”* que irá receber os dados do jogo. Na linha 3, o *score* irá receber a quantidade de maçãs colocada na cesta e na linha 4, o *time* irá receber o tempo exato em que cada maçã foi colocada na cesta.

[7]Linux, Apache, MySQL e PHP

Figura 3.18. Exemplo de jogo para coleta de dados relacionados a habilidade Motora

```
1  CREATE TABLE 'apples' (
2    'id' int(11) NOT NULL,
3    'score' int(11) NOT NULL,
4    'time' time NOT NULL
5  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

Figura 3.19. Código SQL utilizado para o desenvolvimento da tabela que irá receber os dados do jogo

Para que as variáveis fossem armazenadas no banco de dados, um código na linguagem PHP foi hospedado no servidor *online*. A programação do jogo faz uma requisição utilizando o método *POST* ao código-fonte PHP (Figura 3.20). Na linha 2 desse código, foi adicionado um *header* para permitir que o recurso *online* fosse acessado de forma cruzada, isso é necessário devido a forma em que o *Construct 2* permite configurar o acesso a recursos externos; entre as linhas 4 e 7 são criadas variáveis de conexão com o banco de dados, e na linha 7, a variável *$dblink* realiza a conexão com o banco utilizando as variáveis anteriores.

Por fim, as linhas 11 e 12 contêm as variáveis *score* e *time* que irão receber os valores enviados pelo jogo por meio do método *POST*. A linha 13 executa a *query* (consulta) para inserção de dados no banco e, a linha 15 encerra a conexão. O código-fonte exemplifica apenas uma forma simples de inserção de dados no banco, outros recursos em SQL ou a utilização de estruturas existentes podem ser utilizadas para obtenção de melhores resultados no armazenamento de dados.

A Figura 3.21 mostra uma programação para envio de dados para um banco de dados externo. Para realizar esse envio, a ferramenta *Construct 2* utiliza o AJAX[8] e,

[8]Asynchronous Javascript and XML

```php
<?php
 header('Access-Control-Allow-Origin: *');

 $db = "database";//Your database name
 $dbu = "username";//Your database username
 $dbp = "password";//Your database users' password
 $host = "localhost";//Your host

 $dblink = mysqli_connect($host,$dbu,$dbp,$db);

 $score = $_GET['score'];
 $time = $_GET['time'];
 $sql = mysqli_query($dblink, "INSERT INTO '$db'.'apples' ('id','score','time') VALUES ('','$score','$time');");

mysqli_close($dblink);
```

Figura 3.20. Código PHP utilizado para intermediação do jogo com o banco de dados

portanto, ele deve ser adicionado durante o desenvolvimento do jogo na ferramenta.

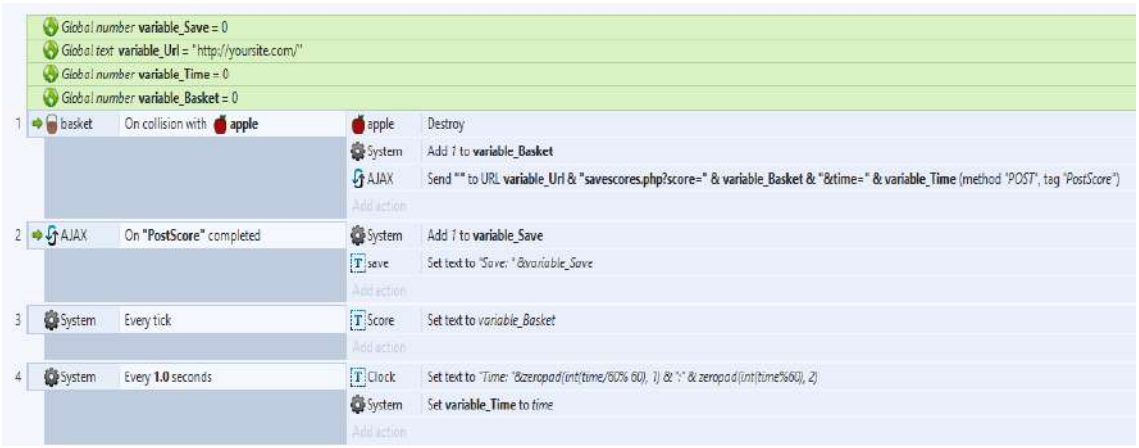


Figura 3.21. Programação utilizada para armazenamentos de dados em um banco de dados externo

Na linha 1 da Figura 3.21, uma condição foi criada para que, quando ocorre uma colisão entre as *sprites "basket"* e *"apple"*, a *sprite "apple"*, desapareça (*apple Destroy*) da interface do jogo para que o usuário entenda que a maçã foi colocada na cesta; também deverá ser adicionado o valor 1 à variável *variable_Basket*. A parte mais importante da programação ocorre ainda na linha 1, com uma chamada em AJAX que é feita da seguinte forma: "*variable_Url & "savescores.php?score="& variable_Basket & "&time="& variable_Time*, na qual:

- *variable_Url*: uma variável global, que contém o endereço do servidor *online*, o qual inclui o código PHP e o banco de dados. Um detalhe importante é que, ao final do endereço atribuído a esta variável, deverá ser incluído "/"para que a concatenação utilizando o simbolo "&"possa ser realizada.

- *"savescores.php?score="*: esse trecho é a continuação do endereço que será concatenado com a variável do endereço do servidor. Nele deverá ser colocado o nome do arquivo que contém o código PHP hospedado no servidor; no exemplo, o arquivo possui o nome "savescores.php", como a requisição utilizará o método *POST* a continuação da requisição passa o valor que a variável *score* irá receber no banco de dados.
- *variable_Basket*: esta variável possui o valor de qual maçã foi colocada na cesta; como é desejado saber também o tempo em que a maçã é colocada na cesta; toda vez que uma maçã é colocada, uma nova entrada no banco é realizada. Dessa forma, será possível saber quanto tempo se passou entre o usuário colocar a maçã 1 na cesta e a maçã 2, por exemplo.
- *"&time="& variable_Time*: o segundo valor que irá ser enviado ao banco é o da variável *time* que, neste exemplo, está concatenado com a variável *variable_Time*. Esta variável recebe o tempo exato que uma maçã é colocada na cesta.
- *(method "POST", tag "PostScore")*: configura qual é o método de requisição que a chamada AJAX deverá efetuar e atribui uma *"tag"* para esta chamada; esse recurso permite verificar se a chamada foi realizada com sucesso.

A linha 2 da Figura 3.21 diz ao sistema do jogo que a todo momento (*"Every Tick"*) o valor da variável *variable_Basket* deverá ser atualizado na interface. Uma caixa de texto com o nome *Score* foi criada e fica sobre a cesta na interface do jogo.

A linha 3 informa ao sistema que a cada 1.0 segundo, a caixa de texto chamada *Clock*, deverá receber o tempo e ser exibido na interface. A ferramenta trabalha, por padrão, com milésimos de segundos sendo necessário realizar a conversão para minutos e segundos, para isso foi utilizada a expressão: *zeropad(int(time/60% 60), 1) & ":"& zeropad(int(time%60), 2)*. Ainda nesta linha, o valor de tempo é adicionado à variável *variable_Time* a cada 1,0 segundo para que o valor exibido na interface e o valor que irá ser adicionado ao banco sempre sejam os mesmos.

Na linha 4 da Figura 3.21, é verificado se a chamada AJAX nomeada de *"PostScore"* foi executada com sucesso. Foi possível atribuir uma ação (também é possível atribuir ações caso a chamada não seja executada com sucesso). A ação atribuída no exemplo adiciona o valor 1 à variável *variable_Save* e exibe este valor na interface por meio de uma caixa de texto chamada *save*, essa variável foi criada para informar que os dados estão sendo salvos com sucesso no banco de dados.

Na Figura 3.22, é apresentado o resultado da consulta dos dados armazenados. Assim, tem-se que a primeira maçã foi colocada na cesta aos 3 segundos e a terceira maçã foi colocada aos 30 segundos. Dessa forma, é possível verificar a velocidade que cada usuário, utilizando suas habilidades motoras, está interagindo com o jogo. Com outras possibilidades de experimentos seria viável controlar também a velocidade em que novas maçãs aparecem na tela, verificar a ordem em que os usuários colocam maçãs na cesta, mudar a posição da cesta e etc.

Por meio do exemplo de jogo aqui descrito, utilizando conceitos sobre habilidades motoras e o uso da ferramenta de desenvolvimento *Construct 2*, foi possível observar um

Figura 3.22. Consulta na tabela *apples* retornando os valores armazenados

modo de armazenar os dados gerados durante a interação do usuário em uma base de dados online. Existem outros modos para armazenar dados utilizando a ferramenta, porém, dentre os existentes atualmente, este modo se mostrou eficiente para análises posteriores.

3.5. Conclusões

Este capítulo abordou o tema de desenvolvimento de jogos para pesquisadores de áreas relacionadas com experimentos que consideram experiência do usuário. Em especial, é discutida a adoção de jogos nas pesquisas científicas, para auxiliar coleta de dados e estudos de caso. Este capítulo atende a um amplo conjunto de pesquisadores, mesmo os que encontram dificuldades ou não possuam familiaridade com o desenvolvimento de elementos de jogos. O uso de jogos em pesquisas se apresenta como um tema de grande relevância para profissionais e pesquisadores envolvidos com experiência do usuário.

Já a experiência dos autores mostrou que o desenvolvimento de jogos pode ser tanto um desafio quanto uma inovação para pesquisas nos diversos campos da Ciência. Pesquisadores dispõem do conhecimento para desenvolver experimentos contendo coleta de dados provenientes da interação do usuário. No entanto, muitas vezes, a falta de familiaridade com ferramentas, técnicas e linguagens utilizadas no desenvolvimento de jogos podem desencoraja-los a adotar estratégias inovadoras envolvendo elementos de *gamificação*.

Finalmente, neste capítulo, é apresentada uma abordagem de desenvolvimento de jogos por meio da ferramenta *Construct 2* que permite diversas possibilidades de apoio ao desenvolvimento de jogos e, principalmente, possibilita o *design* de estudos de casos, bem como a coleta de dados e o armazenamento de variáveis, para futuras análises. Para melhor aproveitamento da abordagem, foi apresentada uma visão geral sobre a área de jogos visando coleta de dados, contendo referências sobre a experiência do usuário, considerando-se os tópicos de usabilidade e acessibilidade, bem como exemplos de estudos relacionados.

3.6. Agradecimentos

Agradecemos ao suporte dado pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e a Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), processo n. 2016/01009-0.

Referências

Barendregt, W., Torgersson, O., Eriksson, E., and Börjesson, P. (2017). Intermediate-level knowledge in child-computer interaction: A call for action. In *Proceedings of the 2017 Conference on Interaction Design and Children*, IDC '17, pages 7–16, New York, NY, USA. ACM.

Chen, S.-T., Huang, Y.-G. L., and Chiang, I.-T. (2012). Using somatosensory video games to promote quality of life for the elderly with disabilities. In *Digital Game and Intelligent Toy Enhanced Learning (DIGITEL), 2012 IEEE Fourth International Conference on*, pages 258–262. IEEE.

Coovert, M. D., Winner, J., Bennett Jr, W., and Howard, D. J. (2017). Serious games are a serious tool for team research. *INTERNATIONAL JOURNAL OF SERIOUS GAMES*, 4(1):41–55.

da Costa, M. A. F. and da Costa, M. d. F. B. (2009). *Metodologia da pesquisa: conceitos e técnicas*. Interciência, Rio de Janeiro, 2 edition.

de Lima Salgado, A., do Amaral, L. A., de Mattos Fortes, R. P., Chagas, M. H. N., and Joyce, G. (2017). Addressing mobile usability and elderly users: Validating contextualized heuristics. In *International Conference of Design, User Experience, and Usability*, pages 379–394. Springer.

Ellis, B., Ford-Williams, G., Graham, L., Grammenos, D., and Hamilton, I. (2012). *Game Accessibility Guidelines* - http://www.gameaccessibilityguidelines.com.

Fortes, R. P. M., de Lima Salgado, A., de Souza Santos, F., do Amaral, L. A., and da Silva, E. A. N. (2017). Game accessibility evaluation methods: A literature survey. In *International Conference on Universal Access in Human-Computer Interaction*, pages 182–192. Springer.

Frederik, D. G., Peter, M., and Jan, V. L. (2010). Uncharted waters?: exploring experts' opinions on the opportunities and limitations of serious games for foreign language learning. In *Proceedings of the 3rd International Conference on Fun and Games*, pages 107–115. ACM.

Gee, J. P. (2003). What video games have to teach us about learning and literacy. *Computers in Entertainment (CIE)*, 1(1):20–20.

Gonçalves, C. (2012). Programa de estimulação cognitiva em idosos institucionalizados. *O Portal dos Psicólogos*, 18:1–18.

Goodman, E., Kuniavsky, M., and Moed, A. (2013). Observing the user experience: A practitioner's guide to user research. *IEEE Transactions on Professional Communication*, 56(3):260–261.

Greitzer, F. L., Kuchar, O. A., and Huston, K. (2007). Cognitive science implications for enhancing training effectiveness in a serious gaming context. *Journal on Educational Resources in Computing (JERIC)*, 7(3):2.

Hassenzahl, M. and Tractinsky, N. (2006). User experience - a research agenda. *Behaviour & Information Technology*, 25(2):91–97.

Hunicke, R. (2005). The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433. ACM.

Isaksen, A., Gopstein, D., and Nealen, A. (2015). Exploring game space using survival analysis. In *FDG*.

Jung, Y., Li, K. J., Janissa, N. S., Gladys, W. L. C., and Lee, K. M. (2009). Games for a better life: effects of playing wii games on the well-being of seniors in a long-term care facility. In *Proceedings of the Sixth Australasian Conference on Interactive Entertainment*, page 5. ACM.

Kapp, K. M. (2012). *The gamification of learning and instruction: game-based methods and strategies for training and education*. John Wiley & Sons.

MacKenzie, I. (2012). *Human-Computer Interaction: An Empirical Research Perspective*. Elsevier Science.

Marconi, M. A. and Lakatos, E. M. (2001). *Metodologia do Trabalho Científico*. Atlas, 6 edition.

Marconi, M. A. and Lakatos, E. M. (2002). *Técnicas de Pesquisa*. Atlas, 5 edition.

Michael, D. R. and Chen, S. L. (2005). *Serious games: Games that educate, train, and inform*. Muska & Lipman/Premier-Trade.

Pedro, L. Z. (2016). *Uso de gamificação em ambientes virtuais de aprendizagem para reduzir o problema da externalização de comportamentos indesejáveis*. PhD thesis, Universidade de São Paulo.

Pádua, E. M. M. (2000). *Metodologia da Pesquisa: Abordagem Teórico-Prática*. Papirus, 6 edition.

Salen, K. and Zimmerman, E. (2004). *Rules of play: Game design fundamentals*. MIT press.

Samodelkin, A., Alavesa, P., and Voroshilov, A. (2016). A platform for pervasive games for research. In *Proceedings of the 15th International Conference on Mobile and Ubiquitous Multimedia*, MUM '16, pages 335–337, New York, NY, USA. ACM.

Souza, G. R. and Trevisan, D. G. (2014). Estudo investigativo sobre idosos, jogos e sua motivações. *Cadernos de Informática*, 8(3):35–40.

Sung, I. and Berland, M. (2017). Forest friends demo: A game-exhibit to promote computer science concepts in informal spaces. In *Proceedings of the 2017 Conference on Interaction Design and Children*, IDC '17, pages 701–704, New York, NY, USA. ACM.

Tong, T., Chignell, M., Tierney, M. C., and Lee, J. (2016). A serious game for clinical assessment of cognitive status: validation study. *JMIR serious games*, 4(1).

Torres, A. C. S. (2011). Cognitive effects of video games on old people. *International Journal on Disability and Human Development*, 10(1):55–58.

Valladares-Rodríguez, S., Pérez-Rodríguez, R., Anido-Rifón, L., and Fernández-Iglesias, M. (2016). Trends on the application of serious games to neuropsychological evaluation: A scoping review. *Journal of biomedical informatics*, 64:296–319.

Van Eck, R. (2006). Digital game-based learning: It's not just the digital natives who are restless. *EDUCAUSE review*, 41(2):16.

Wiemeyer, J. and Kliem, A. (2011). Serious games in prevention and rehabilitation—a new panacea for elderly people? *European Review of Aging and Physical Activity*, 9(1):41.

Wood, M., Wood, G., and Balaam, M. (2017). Sex talk: Designing for sexual health with adolescents. In *Proceedings of the 2017 Conference on Interaction Design and Children*, IDC '17, pages 137–147, New York, NY, USA. ACM.

Biografia resumida dos autores

Marcio Maestrelo Funes: é mestrando no programa de Pós-Graduação em Ciências da Computação e Matemática Computacional no Instituto de Ciências Matemáticas e de Computação (ICMC), na Universidade de São Paulo (USP). Tem experiência na área de jogos tendo atuado como docente e desenvolvedor. Atualmente realiza pesquisas em avaliação de experiência do usuário com interfaces baseadas em gestos, acessibilidade e Web. Atua principalmente nos seguintes temas: Interface Humano-Computador, Computação Ubíqua, Interfaces Naturais, realidade aumentada e virtual, design e acessibilidade.

Leandro Agostini do Amaral: é doutorando no programa de Pós-Graduação em Ciências da Computação e Matemática Computacional no Instituto de Ciências Matemáticas e de Computação (ICMC), na Universidade de São Paulo (USP) e Pesquisador Responsável em um projeto PIPE/FAPESP que utiliza jogos para treinamento cognitivo voltado ao público acima de 50 anos. Desde 2010 pesquisa assuntos relacionados à Interação Humano-Computador, priorizando questões de acessibilidade.

Rudinei Goularte: é professor do Departamento de Ciências de Computação na Universidade de São Paulo, campus de São Carlos. Possui graduação em Ciências da Computação pela Universidade Federal de Mato Grosso do Sul (1995). Possui mestrado (1998), doutorado (2003) e livre-docência (2011) pela Universidade de São Paulo [São Carlos], todos em Ciência da Computação. Atualmente é professor associado do ICMC/USP em regime de dedicação integral à docência e à pesquisa e orientador pleno de mestrado e doutorado. Atua como consultor ad hoc da Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq). Desenvolve pesquisa em Multimídia nas linhas: codificação de vídeo digital, vídeo 3D, recuperação baseada em conteúdo, personalização de conteúdo e tv interativa.

Renata Pontin M. Fortes: é professora do Departamento de Ciências de Computação na Universidade de São Paulo, campus de São Carlos. Possui graduação em Bacharelado em Ciência da Computação pela Universidade de São Paulo (1982), mestrado em Ciência da Computação e Matemática Computacional pela Universidade de São Paulo (1991) e doutorado em Física pela Universidade de São Paulo (1996). Atualmente é professora associada da Universidade de São Paulo, consultor do Ministério da Educação e consultor ad hoc da Fundação de Amparo à Pesquisa do Estado de São Paulo. Tem experiência na área de Ciência da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: engenharia de web, projetos de software livre, acessibilidade na web e processo de software.

Capítulo 4

Model-driven Engineering in the Development of Ubiquitous Applications: Technologies, Tools and Languages

Marcos Alves Vieira[1] e Sergio T. Carvalho[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia Goiano (IF Goiano)
Iporá – GO – Brasil

[2]Instituto de Informática – Universidade Federal de Goiás (UFG)
Goiânia – GO – Brasil

marcos.vieira@ifgoiano.edu.br, sergio@inf.ufg.br

Abstract

Model-driven Engineering (MDE) is an approach that considers models as the main artifacts in software development. Models are generally built using domain specific languages, such as UML and XML. These languages, in turn, are defined by their own metamodel. In this context, this proposal aims to present a short course, with theoretical and practical approach, covering the basics of MDE as well as key frameworks and languages available for its support, aiming at the construction of ubiquitous applications. By the end of the short course, each participant will have the necessary background to be able to build a graphical modeling tool to design models in accordance with a particular metamodel. These models can then be used to document and maintain systems from different domains.

Resumo

A Engenharia Dirigida por Modelos (Model-driven Engineering - MDE) é uma abordagem que considera os modelos como os principais artefatos no desenvolvimento de um software. Modelos são geralmente construídos usando linguagens específicas de dominio, como a UML e a XML. Essas linguagens, por sua vez, são definidas por metamodelos próprios. Nesse contexto, essa proposta tem como objetivo apresentar um minicurso, com enfoque teórico e prático, abordando os fundamentos de MDE, assim como os principais frameworks e linguagens disponíveis para o seu suporte, com foco na construção de aplicações ubíquas. Ao final do minicurso, cada participante terá os conhecimentos necessários para construir uma ferramenta de modelagem gráfica que possibilite construir modelos em conformidade com um metamodelo em particular. Esses modelos podem então ser usados para documentar e manter sistemas de diferentes domínios.

4.1. Introdução

Esse capítulo tem como objetivo capacitar os seus leitores na construção de uma ferramenta de modelagem gráfica para concepção de modelos em conformidade com um determinado metamodelo. Seu público-alvo são estudantes de graduação e pós-graduação, além de profissionais da área de desenvolvimento de sistemas.

Os conceitos são apresentados diretamente por meio das ferramentas e de uma boa parcela de codificação e testes. Os leitores terão, portanto, a oportunidade de ter contato, não só com as definições e os aspectos conceituais da área, como também com o desenvolvimento passo a passo considerando um cenário-exemplo de computação ubíqua.

O capítulo está dividido em três partes:

- A primeira parte (Seção 4.2) apresenta alguns dos fundamentos teóricos necessários para a construção de um metamodelo para modelagem de espaços inteligentes:
  - Subseção 4.2.1: Computação Ubíqua
  - Subseção 4.2.2: Computação Sensível ao Contexto
  - Subseção 4.2.3: Espaços Inteligentes Fixos
  - Subseção 4.2.4: Espaços Inteligentes Pessoais
  - Subseção 4.2.5: Objetos Inteligentes
  - Subseção 4.2.6: Engenharia Dirigida por Modelos (*Model-driven Engineering* - MDE)
- A segunda parte (Seção 4.3) consiste nos fundamentos tecnológicos envolvidos:
  - Subseção 4.3.1: Eclipse Modeling Framework (EMF)
  - Subseção 4.3.2: Eclipse Graphical Modeling Framework (GMF)
  - Subseção 4.3.3: Epsilon, uma família de linguagens e ferramentas relacionadas à MDE
- Por fim, a terceira parte, apresentada na Seção 4.4, traz um tutorial alicerçado nos conceitos teóricos e tecnológicos previamente apresentados, com foco no desenvolvimento de uma ferramenta de modelagem gráfica que possibilite a construção de modelos com base em um metamodelo próprio, além de trazer um exemplo de ferramenta de modelagem construída com os conhecimentos apresentados nesse minicurso.

As considerações finais do capítulo são apresentadas na Seção 4.5.

4.2. Fundamentação Teórica

Nesta seção são apresentados os fundamentos teóricos e tecnológicos necessários para o desenvolvimento da ferramenta de modelagem gráfica.

4.2.1. Computação Ubíqua

A utilização de computadores pode ser dividida em três fases ou eras [Weiser and Brown 1997]. A primeira diz respeito à época em que o computador era um recurso escasso e atendia a diversos usuários, caracterizada como era dos *mainframes*. A segunda era corresponde aos computadores pessoais (PCs), onde cada pessoa tem acesso exclusivo a um computador. A terceira, por sua vez, representa a computação ubíqua, conforme previsto por Mark Weiser [Weiser 1991].

Weiser vislumbrou a possibilidade de tornar a utilização da computação invisível ao usuário, fundindo-a com elementos do dia-a-dia, ou seja, fazendo com que o usuário não precisasse perceber a tecnologia para aproveitar seus benefícios. Segundo este conceito, a computação estaria permeada nos objetos do ambiente físico do usuário, não requerendo dispositivos computacionais tradicionais para a interação, tais como teclado e mouse. Na computação ubíqua, o foco do usuário sai do dispositivo computacional que ele manipula e passa para a tarefa ou para a ação a ser realizada.

É de suma importância que um ambiente ubíquo ofereça suporte à mobilidade, isto é, manter disponíveis os recursos computacionais do usuário enquanto este se move de um lugar para outro [Lupiana et al. 2009]. Contudo, essa mobilidade requer uma infraestrutura de rede bem definida e uma série de outros recursos, tais como meios de obter a localização exata do usuário e também de possibilitar a descoberta e o uso dos serviços disponíveis.

Pesquisadores por todo o globo têm se sentido atraídos pela computação ubíqua. O Comitê de Pesquisa em Computação do Reino Unido (*UK Computing Research Committee* - UKCRC), por exemplo, identificou na área alguns dos grandes desafios da computação para as próximas décadas [Kavanagh and Hall 2008].

Sistemas de computação ubíqua são encontrados em diversos domínios. Um sistema ubíquo para assistência domiciliar à saúde (também conhecida como *homecare*), por exemplo, pode ser utilizado para identificar as atividades físicas cotidianas de um indivíduo ou auxiliar no tratamento de pessoas, indicando os horários para se tomar as medicações (*e.g.*, [Carvalho et al. 2011, Sztajnberg et al. 2009]), entre outras funções relacionadas ao cuidado com a saúde. Sistemas ubíquos para aprendizagem podem possibilitar a formação de grupos de alunos para trabalhar em conjunto na solução de um determinado problema proposto pelo professor, cada um utilizando seu próprio dispositivo móvel pessoal (*e.g.*, [Yau et al. 2003]), ou facilitar a participação em salas de aula virtuais por meio de reconhecimento de voz e gestos (*e.g.*, [Shi et al. 2003]).

A computação ubíqua necessita se apoiar em outros conceitos para que possa ser implementada em sua plenitude, como a computação sensível ao contexto, espaços inteligentes e objetos inteligentes. Estes conceitos são apresentados nas subseções seguintes.

4.2.2. Computação Sensível ao Contexto

Uma aplicação ubíqua deve ser minimamente intrusiva, o que exige um certo conhecimento de seu contexto de execução, isto é, deve ser possível para a aplicação obter informações sobre o estado dos usuários e do ambiente de execução, possibilitando que ela modifique seu comportamento com base nessas informações [Erthal 2014].

Neste trabalho o conceito de aplicação ubíqua utiliza uma adaptação daquele utilizado por [Roriz Junior 2013].

Uma aplicação ubíqua é um conjunto de instâncias de uma mesma aplicação em um cenário de computação ubíqua.

O termo "sensibilidade ao contexto" foi mencionado pela primeira vez por Schilit e Theimer [Schilit and Theimer 1994], como um software que "se adapta de acordo com a sua localização de uso, o conjunto de pessoas e os objetos próximos, assim como as mudanças que esses objetos sofrem no decorrer do tempo". Chagas *et al.* [Chagas et al. 2010] afirmam que a sensibilidade ao contexto permite "utilizar informações relevantes sobre entidades do ambiente para facilitar a interação entre usuários e aplicações".

As mudanças ocorridas no modelo de um software em execução podem ser desencadeadas por modificações no ambiente físico onde este software está em execução. Como exemplo, pode-se citar a elevação da temperatura de uma sala, que faz com que o software responsável por monitorar aquele ambiente ative o aparelho de ar-condicionado. O sistema em execução, nesse caso, reage a um contexto capturado (aumento de temperatura) e modifica seu comportamento (ativação do ar-condicionado).

A definição de contexto, no escopo deste trabalho, está relacionada àquela proposta por Abowd *et al.* [Abowd et al. 1999] e Dey [Dey 2001]:

Qualquer informação que possa ser usada para caracterizar a situação de uma pessoa, lugar ou objeto relevante para a interação entre um usuário e uma aplicação, incluindo estes dois últimos.

Nesse sentido, um sistema sensível ao contexto é capaz de obter informações do seu ambiente de execução, avaliar esta informação e mudar seu comportamento de acordo com a situação [Cetina et al. 2009].

4.2.3. Espaços Inteligentes

A convergência de tecnologias móveis e da Internet, propiciada principalmente pela popularização dos dispositivos e pela facilidade de acesso por meio de conexões móveis de terceira e quarta gerações (3G e 4G), está tornando o mundo mais conectado e com acesso à Internet mais disponível. Isso, combinado com Web das Coisas (*Web of Things* - WoT) [Guinard et al. 2009] e Internet das Coisas (*Internet of Things* - IoT) [Atzori et al. 2010], traz como potencial a capacidade de interligar, monitorar e controlar remotamente diversos dispositivos cotidianos (TVs, fechaduras, interruptores de lâmpadas, alarmes residenciais, etc.), fortalecendo o conceito de computação ubíqua.

A diminuição do escopo do ambiente de computação ubíqua mitiga os problemas de integração, em particular, devido à possibilidade de prever alguns comportamentos do usuário no local. Os espaços inteligentes (*smart spaces*) utilizam essa premissa para instrumentar e projetar a infraestrutura do ambiente como meio de estabelecer serviços para habilitar a computação ubíqua em um determinado ambiente.

A computação sensível ao contexto é propiciada pelos espaços inteligentes, pois estes permitem a aquisição de conhecimento a respeito do ambiente e a adaptação dos seus participantes no sentido de se tirar melhor proveito deste ambiente [Cook and Das 2007].

Para tal, os sensores monitoram e coletam informações do ambiente físico e determinadas ações são realizadas baseadas em decisões tomadas por um mecanismo de raciocínio.

Os mecanismos de raciocínio filtram e gerenciam as grandes quantidades de informações que trafegam nos espaços inteligentes diariamente. O papel de um mecanismo de raciocínio se divide em duas frentes: *(i)* modelar as informações coletadas em conhecimento abstrato útil; e *(ii)* raciocinar sobre esse conhecimento para apoiar de maneira eficaz as atividades diárias dos usuários [Cook and Das 2007].

Os espaços inteligentes estendem a computação para os ambientes físicos e permitem que diferentes dispositivos forneçam suporte coordenado aos usuários baseado em suas preferências e na atual situação do ambiente físico (contexto) [Smirnov et al. 2013]. Em outras palavras, em um nível de abstração bastante elevado, os espaços inteligentes podem ser tidos como ambientes de computação ubíqua que entendem e reagem às necessidades humanas [Lupiana et al. 2009].

Lupiana *et al.* [Lupiana et al. 2009] analisaram as diversas definições de espaços inteligentes, dadas por diferentes autores, para propor uma definição mais abrangente:

Um ambiente computacional e sensorial altamente integrado que efetivamente raciocine sobre os contextos físico e de usuário do espaço para agir transparentemente com base em desejos humanos.

Os autores, em seguida, fornecem mais detalhes sobre esta definição, argumentando que um ambiente é *altamente integrado* quando este é saturado com dispositivos de computação ubíqua e sensores completamente integrados com redes sem fio; o *raciocínio efetivo* pode ser atingido por um mecanismo pseudo-inteligente para o ambiente como um todo, e não somente para dispositivos ou componentes individuais; *contexto de usuário* refere-se a perfis individuais, políticas, localização atual e *status* de mobilidade; e *transparência* está relacionada com as ações humanas e com o suporte à mobilidade sem que, para isso, seja necessária a interação direta do usuário.

Dada a vasta quantidade de informação que pode trafegar em um espaço inteligente, os numerosos objetos inteligentes que o compõem, e a possibilidade da entrada e saída de usuários, que carregam consigo seus dispositivos móveis, a segurança é um dos aspectos críticos desses ambientes. Em [Al-Muhtadi et al. 2003], os autores elencam uma série de requisitos de segurança com os quais os espaços inteligentes devem se preocupar, incluindo:

- Acesso multinível, ou seja, disponibilizar diferentes níveis de acesso de acordo com políticas pré-definidas, com a situação atual do espaço inteligente e com os recursos disponíveis.
- Uma política de acesso descritiva, bem definida, flexível e de fácil configuração.
- A autenticação dos usuários humanos, bem como das aplicações e dos dispositivos móveis que entram e saem do espaço inteligente.

Outro aspecto importante a ser considerado em um espaço inteligente é a mobilidade do usuário. Os espaços inteligentes devem oferecer suporte à mobilidade e integrar

técnicas e dispositivos computacionais para possibilitar a interação do usuário com o ambiente de maneira transparente e intuitiva [Lupiana et al. 2009].

A pesquisa na área de espaços inteligentes é bastante pujante. Há mais de uma década os pesquisadores vêm propondo soluções para a área, como é o caso do trabalho de Johanson e Fox [Johanson and Fox 2002], que propõem o Event Heap, um espaço de trabalho colaborativo que estende o TSpaces, espaço de tuplas apresentado pela IBM Research [Wyckoff et al. 1998], no qual diferentes aplicações podem publicar e receber eventos de seu interesse.

O trabalho de Coen *et al.* [Coen et al. 2000] apresenta a Metaglue, uma linguagem de programação baseada em Java, desenvolvida nos laboratórios de inteligência artificial do Instituto de Tecnologia de Massachusetts (MIT). O objetivo da Metaglue é permitir a construção de aplicações para espaços inteligentes, oferecendo suporte a uma série de necessidades específicas deste tipo de aplicação, como, por exemplo, interconectar e gerenciar hardware e software heterogêneos, adicionar, remover, modificar e atualizar componentes em um sistema em tempo de execução, bem como controlar a alocação de recursos.

Mais recentemente, os trabalhos na área de espaços inteligentes que têm chamado atenção são as iniciativas de código-fonte aberto: openHAB[1] e Smart-M3[2].

O projeto openHAB (*open Home Automation Bus*) visa fornecer uma plataforma de integração universal para automação residencial. O openHAB é uma solução Java, completamente baseada em OSGi[3], o que facilita seu objetivo de possibilitar a interconexão de hardware de diversos fornecedores, mesmo que estes utilizem diferentes protocolos de comunicação. Smart-M3 é uma plataforma que oferece às aplicações distribuídas uma visão compartilhada das informações e serviços presentes em ambientes ubíquos, baseada no modelo arquitetural de quadro-negro (*blackboard*). Os espaços inteligentes são tidos como provedores de informações e as aplicações que compõem os espaços inteligentes, por sua vez, produzem e consomem informações.

Os espaços inteligentes tradicionais discutidos nesta subseção são referenciados neste trabalho como *espaços inteligentes fixos*. Em distinção a eles, é apresentado a seguir o conceito de *espaços inteligentes pessoais*.

4.2.4. Espaços Inteligentes Pessoais

Existem diversas iniciativas com objetivo de projetar os tradicionais espaços inteligentes, ou espaços inteligentes fixos, como os pioneiros Gaia [Román et al. 2002], Aura [Sousa and Garlan 2002], Olympus [Ranganathan et al. 2005] e a casa inteligente Gator Tech [Helal et al. 2005]. Além desses, existe também uma série de outros trabalhos mais recentes (*e.g.*, [Corredor et al. 2012, Freitas et al. 2014, Honkola et al. 2010]). Entretanto, essas propostas focam em fornecer serviços em espaços confinados e geograficamente limitados, e têm uma perspectiva centrada no sistema, na qual os usuários são atores externos que não estão contidos no espaço inteligente, ou seja, os usuários estão meramente localizados nos espaços inteligentes, não fazendo parte deles [Taylor 2011]. Esta visão

[1] `http://www.openhab.org`
[2] `http://sourceforge.net/projects/smart-m3/`
[3] `http://www.osgi.org/Main/HomePage`

contrasta com o conceito original de computação calma introduzido por Mark Weiser [Weiser and Brown 1997], que tem o usuário como o núcleo da computação.

Além disso, programar apenas espaços inteligentes fixos pode levar a ilhas de ubiquidade separadas por espaços vazios, onde o suporte à computação ubíqua é limitado, pois estes não possibilitam o compartilhamento de dispositivos e serviços com outros espaços inteligentes [Crotty et al. 2009].

Em contraste com os espaços inteligentes tradicionais, que são fixos e limitados a uma determinada área lógica ou física, um espaço inteligente pessoal (*Personal Smart Space* - PSS) é formado com base nos conceitos de computação ubíqua aliados a uma rede corporal. Os diversos sensores, atuadores e dispositivos que um usuário carrega formam uma rede corporal [Chen et al. 2011, Latré et al. 2011] e esta, por sua vez, apoiada em uma infraestrutura de software projetada para esta finalidade, compõe o seu espaço inteligente pessoal [Dolinar et al. 2008].

Em seu trabalho, Korbinian *et al.* [Roussaki et al. 2008] listam as principais características de um PSS:

1. **O PSS é móvel**: ao contrário das abordagens de espaços inteligentes tradicionais, os limites físicos de um PSS se movem com o usuário. Essa funcionalidade possibilita a sua sobreposição com outros espaços inteligentes (fixos ou pessoais).

2. **O PSS tem um "dono"**: o dono de um PSS é a pessoa sobre a qual o PSS opera. Isso possibilita que o PSS seja personalizável. As preferências de um PSS podem ser levadas em consideração na resolução de conflitos, por exemplo, para definir a melhor temperatura do ar-condicionado de uma sala com base na média das preferências dos usuários presentes.

3. **O PSS deve oferecer suporte a um ambiente *ad-hoc***: um PSS deve ser capaz de operar em ambientes de redes estruturadas e também *ad-hoc*, promovendo o uso do PSS como integrador de dispositivos.

4. **O PSS deve ser capaz de se adaptar**: as aplicações dentro um PSS devem ser capazes de se adaptar à situação corrente. Além disso, o PSS deve facilitar a interação de suas aplicações com o ambiente.

5. **O PSS pode aprender a partir de interações anteriores**: ao minerar as informações armazenadas, o PSS pode detectar tendências e inferir as condições em que as mudanças de comportamento ou preferências do usuário são manifestadas. Isso permite que recomendações sejam feitas quando o PSS interage com outros PSS ou até mesmo para agir proativamente com base nas intenções do usuário.

As duas primeiras características possibilitam que o PSS acompanhe seu *dono*, estando sempre disponível e permitindo a interação com outros espaços inteligentes, sejam fixos ou mesmo pessoais [Taylor 2008].

As características quatro e cinco habilitam o autoaperfeiçoamento, um dos principais objetivos de um PSS. O autoaperfeiçoamento é o aprendizado de tendências no comportamento do usuário, possibilitando recomendações e previsões para que adaptações ao

ambiente sejam realizadas automaticamente para o usuário [Gallacher et al. 2010]. Dessa forma, a sensibilidade ao contexto é um aspecto crucial para a funcionalidade de um PSS.

Uma infinidade de fontes de contexto fornece dados que precisam ser coletados, disseminados e gerenciados de maneira eficiente, como, por exemplo, informações relacionadas à localização e atividades do usuário, padrões de movimento, temperatura, nível de ruído do ambiente, dentre outros [Roussaki et al. 2015]. Em [Roussaki et al. 2012], os autores apresentam as características de um componente de gerenciamento de contexto para espaços inteligentes pessoais, construído com base em um conjunto de requisitos específicos para PSS, tais como: *(i)* gerenciamento das fontes de contexto; *(ii)* modelagem, gerenciamento, armazenamento e processamento de informações de contexto histórico; e *(iii)* mecanismos de gerenciamento de eventos.

Uma recente proposta de modelagem de espaços inteligentes foi apresentada em [Vieira 2016] e considera o conceito de espaços inteligentes pessoais explorado pelo projeto PERSIST (*Personal Self-Improving Smart Spaces*) [Dolinar et al. 2008], o qual apresenta a visão de que os espaços inteligentes pessoais fornecem uma interface entre o usuário e os vários serviços e dispositivos disponíveis. Dessa forma, o espaço inteligente pessoal de um usuário, formado pelos objetos inteligentes (sensores, atuadores e demais dispositivos) que ele carrega consigo, pode interagir com outros espaços inteligentes, sejam estes pessoais ou fixos.

O conceito de espaços inteligentes pessoais é considerado como uma alternativa para que os serviços computacionais estejam sempre disponíveis ao usuário, independentemente de sua movimentação pelos ambientes, minimizando o problema das ilhas de ubiquidade.

4.2.5. Objetos Inteligentes

A constante miniaturização dos dispositivos computacionais e o surgimento de tecnologias como o RFID (*Radio-Frequency IDentification*) possibilitaram o rastreamento de objetos do dia-a-dia em ambientes confinados, como lojas ou depósitos.

Os objetos inteligentes (*smart objects*), são uma evolução desses "objetos rastreáveis". Nesse conceito, os objetos inteligentes são entidades físico/digitais, aumentados com capacidades de sensoriamento, processamento e possibilidade de conexão em rede [Kortuem et al. 2010]. Um objeto inteligente pode perceber seu ambiente por meio de sensores e se comunicar com outros objetos próximos. Essas capacidades permitem que os objetos inteligentes trabalhem colaborativamente para determinar o contexto e adaptar seu comportamento [Siegemund 2004].

Em contraste com as *tags* RFID, os objetos inteligentes podem "sentir", registrar e interpretar o que está acontecendo com eles mesmos e com o ambiente físico, agir por conta própria, comunicar-se com outros objetos inteligentes e trocar informações com as pessoas [Kortuem et al. 2010].

Entretanto, os objetos inteligentes não se referem apenas aos objetos do dia-a-dia não digitais. Dispositivos computacionais tradicionais, como *smartphones*, PDAs, *players* de música, etc. podem ter ou ser aumentados com tecnologias sensitivas, tais como sensores, atuadores e algoritmos de percepção [Kawsar 2009].

Kawsar [Kawsar 2009] argumenta que um objeto inteligente deve possuir certas propriedades para garantir seu pleno funcionamento e interação com demais objetos inteligentes, tais como:

- **ID única**: é essencial poder identificar unicamente os objetos inteligentes no mundo digital. A identificação pode ser o endereço da interface de rede ou o endereço no nível de aplicação, considerando o serviço de resolução de nomes apropriado.
- **Autoconsciência**: espera-se que um objeto inteligente seja capaz de saber seu estado operacional e situacional, além de ser capaz de se descrever.
- **Sociabilidade**: um objeto inteligente deve ser capaz de se comunicar com outros objetos inteligentes e entidades computacionais (*e.g.*, uma aplicação sensível ao contexto) para compartilhar sua autoconsciência.
- **Autonomia**: um objeto inteligente deve ser capaz de tomar certas ações. Essas ações podem ser tão simples como mudar seu estado operacional (*e.g.*, mudar-se do estado desligado para ligado) ou tão complexas como adaptar seu comportamento por meio de tomadas de decisão autônomas.

4.2.6. Engenharia Dirigida por Modelos

O conceito de Engenharia Dirigida por Modelos (*Model-Driven Engineering* - MDE) considera que os modelos são os principais artefatos no desenvolvimento de um sistema. Segundo esta abordagem, os modelos não servem apenas para descrever ou documentar um software, mas também para atuar no seu desenvolvimento, manutenção e operação [Schmidt 2006, Seidewitz 2003]. Um modelo é uma representação gráfica ou textual de alto nível de um sistema, onde cada um de seus elementos é uma representação virtual de um componente presente no sistema real. Os relacionamentos e as abstrações utilizadas em um modelo são descritos por um metamodelo [Völter et al. 2013].

As técnicas de MDE, tais como Desenvolvimento Dirigido por Modelos (*Model-Driven Development* - MDD) e Arquitetura Dirigida por Modelos (*Model-Driven Architecture* - MDA) propõem o uso de abstrações mais próximas do domínio do problema como uma forma de mitigar a distância semântica existente entre o problema a ser solucionado e o ferramental (software) utilizado para tal.

A ênfase na integração entre a parte tecnológica e o conhecimento específico de um determinado domínio é um importante aspecto da MDE [Favre and Nguyen 2005], e a utilização de seus princípios aumenta a qualidade dos sistemas de software, o grau de reuso e, como resultado implícito, a eficiência em seu desenvolvimento [Bézivin et al. 2005].

Dada a popularidade do uso de modelos, surgiu a necessidade de uma padronização para a construção de metamodelos e modelos. Dessa forma, o *Object Management Group* (OMG)[4] apresentou uma arquitetura de metamodelagem de quatro camadas, denominada *Meta-Object Facility* (MOF). Na MOF, cada elemento de uma camada inferior é uma instância de um elemento de uma camada superior, conforme ilustrado na Figura 4.1.

[4] `http://www.omg.org`

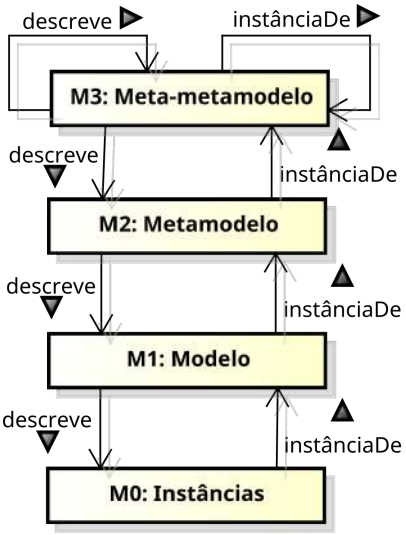


Figura 4.1. Camadas da arquitetura de metamodelagem MOF. Adaptado de [Völter et al. 2013].

As camadas MOF podem ser descritas da seguinte forma [Seidewitz 2003]:

- **Camada M3**: representa o meta-metamodelo da MOF, também chamado de Modelo MOF, utilizado para construção dos metamodelos. O modelo MOF formaliza suas próprias abstrações, eliminando a necessidade de um nível superior. Outro exemplo de membro desta camada é o Ecore, que é baseado no MOF.
- **Camada M2**: contém os metamodelos que podem ser utilizados para modelar sistemas de domínio específico. A *Unified Modeling Language* (UML) é um exemplo de membro desta camada.
- **Camada M1**: composta por modelos que descrevem sistemas utilizando as definições constantes em seus respectivos metamodelos presentes em M2.
- **Camada M0**: contém as entidades ou objetos que formam o sistema em execução, que são criadas a partir das definições presentes em M1.

Os metamodelos geralmente são construídos para permitir a criação de modelos que expressem conceitos específicos de um domínio [López-Fernández et al. 2015], como por exemplo, sistemas de distribuição de energia elétrica (*microgrids*) [Sampaio Junior 2014], *CrowdSensing* [Melo 2014], ou Linhas de Produto de Software [Carvalho 2013, Ferreira Filho 2014]. Sendo assim, um metamodelo pode ser considerado uma Linguagem de Modelagem Específica de Domínio (*Domain-Specific Modeling Language* - DSML). Uma DSML é "uma linguagem textual ou gráfica que oferece, por meio das notações e abstrações apropriadas, poder de expressividade com foco em um domínio de problema particular, para visualizar, especificar, construir e documentar artefatos de um sistema software" [Chiprianov et al. 2014, Van Deursen et al. 2000].

Como qualquer linguagem, as DSMLs possuem dois componentes principais [Ferreira Filho 2014]: sintaxe e semântica. A sintaxe de uma DSML pode ser dividida em sin-

taxe abstrata e sintaxe concreta. A sintaxe abstrata define seus conceitos e relacionamentos entre eles, enquanto a sintaxe concreta mapeia esses conceitos em elementos visuais que são usados nos modelos. A semântica de uma DSML é o significado das representações da sintaxe. A sintaxe abstrata é o componente mais importante de uma DSML. "É comum encontrar DSMLs sem definições formais de sua semântica ou sem uma representação concreta, mas sua sintaxe abstrata é imperativa" [Ferreira Filho 2014].

Em [Kolovos et al. 2006], os autores especificam uma série de requisitos gerais (de 1 a 7) e requisitos adicionais (8 e 9) para construção de linguagens específicas de domínio, a saber:

1. **Conformidade**: as construções devem corresponder a importantes conceitos do domínio.

2. **Ortogonalidade**: cada construção da linguagem deve ser usada para representar exatamente um conceito distinto do domínio.

3. **Suporte**: é interessante oferecer suporte à linguagem por meio de ferramentas para modelagem e gerenciamento de programação, *e.g.*, criação de código, edição e transformação de modelos.

4. **Integração**: a linguagem, e suas ferramentas podem ser utilizadas em conjunto com outras linguagens e ferramentas com o mínimo de esforço.

5. **Longevidade**: assume-se com esse requisito que o domínio em consideração persista por um período de tempo suficiente para justificar a construção da linguagem e suas ferramentas.

6. **Simplicidade**: uma linguagem deve ser o mais simples possível.

7. **Qualidade**: a linguagem deve fornecer mecanismos para construção de sistemas de qualidade.

8. **Escalabilidade**: a linguagem deve fornecer construções para ajudar a gerenciar descrições de larga escala, além de permitir a construção de sistemas menores.

9. **Usabilidade**: esse requisito diz respeito a conceitos como economia, acessibilidade e facilidade de compreensão. Essas características podem ser parcialmente cobertas pelos requisitos gerais, *e.g.*, simplicidade pode ajudar a promover a facilidade de compreensão.

4.3. Fundamentação Tecnológica

As subseções seguintes descrevem os conceitos tecnológicos envolvidos na construção dos metamodelos propostos e na implementação da ferramenta de modelagem gráfica.

4.3.1. *Eclipse Modeling Framework* (EMF)

O *Eclipse Modeling Framework* (EMF) [Steinberg et al. 2008] é um *framework* de modelagem construído sobre o ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE) *Eclipse*. O EMF fornece mecanismos para a criação, edição e validação de modelos e metamodelos, além de permitir a geração de código a partir dos modelos. Para tal, o EMF possibilita a geração de uma implementação em linguagem Java, de maneira que cada uma das classes do metamodelo (chamadas de metaclasses) corresponde a uma classe em Java. Dessa forma, estas classes podem ser instanciadas para criar modelos em conformidade com o metamodelo. O EMF também possibilita criar editores para modelos em conformidade com os seus metamodelos.

Os metamodelos construídos no EMF são instâncias do meta-metamodelo Ecore que, por sua vez, é baseado no meta-metamodelo MOF. Sendo assim, o Ecore é a linguagem central do EMF [Steinberg et al. 2008]. Um metamodelo com base no Ecore é definido por meio de instâncias de classes do tipo: `EClass`, `EAttribute`, `EReference`, `ESuperType` e `EDataType`. Uma `EClass` representa uma classe composta por atributos e referências. Um `EAttribute` é um atributo que possui um nome e um tipo. Uma `EReference` define uma associação entre classes e, no caso da definição de uma superclasse, para se valer dos conceitos de herança, essa associação é do tipo `ESuperType`. Por fim, um `EDataType` é o tipo de um atributo, cujo valor pode ser um tipo primitivo (número inteiro ou real, *string*, booleano, etc...), uma enumeração `EEnum` ou uma referência a uma `EClass`.

Uma visão geral dos componentes do meta-metamodelo Ecore [Merks and Sugrue 2009], com seus atributos, operações e relacionamentos, pode ser vista na Figura 4.2. A seguir é apresentado um breve detalhamento dos elementos que o compõem.

- `EClass`: modela classes. Classes são identificadas por um nome e podem conter um número de características, *i.e.*, atributos e referências. Para permitir o suporte a herança, uma classe pode referenciar outra classe como seu supertipo. A herança múltipla também é permitida e, nesse caso, diferentes classes são referenciadas como supertipos. Uma classe pode ser abstrata e, dessa forma, uma instância sua não pode ser criada. Caso uma classe seja definida como uma interface, sua implementação não é criada durante a geração de código.

- `EAttribute`: modela atributos e são identificados por um nome e possuem um tipo. Limites inferiores e superiores são especificadas para a multiplicidade do atributo.

- `EDataType`: modela tipos simples cuja estrutura não é modelada. Eles agem como empacotadores que denotam um tipo primitivo ou um tipo de objeto definido em Java. São identificados por um nome e são mais frequentemente utilizados como tipos de atributos.

- `EReference`: modela uma extremidade de uma associação entre duas classes. Elas são identificadas por um nome e um tipo, onde o tipo representa a classe na outra extremidade da associação. A bidirecionalidade é suportada ao parear uma

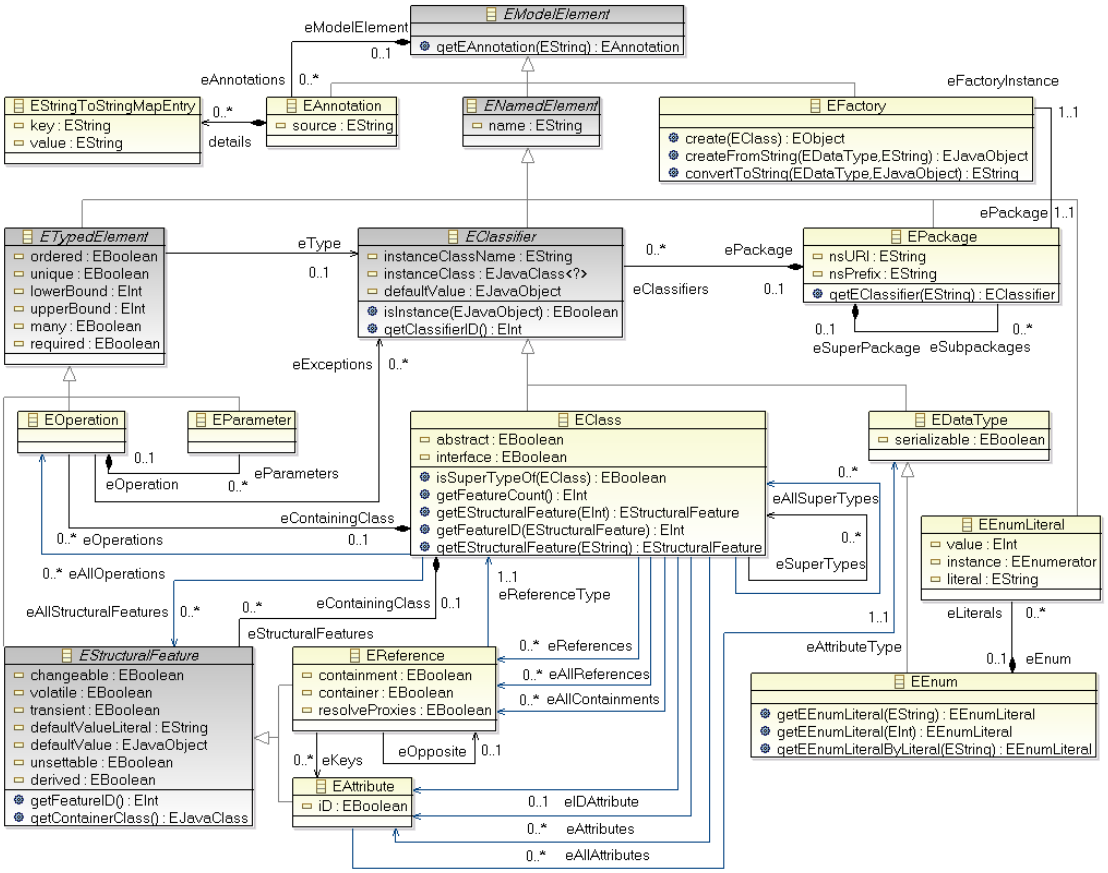


Figura 4.2. Meta-metamodelo Ecore [Merks and Sugrue 2009].

referência com seu oposto, *i.e.*, uma referência na classe representando a outra extremidade da associação. Limites inferiores e superiores são especificados na referência para denotar sua multiplicidade. Uma referência pode oferecer suporte a um tipo forte de associação, chamado *containment*, semelhante à associação do tipo "composição", da UML.

- `EModelElement`: modela os elementos de um modelo Ecore. É a raiz abstrata de uma hierarquia de classes Ecore.

- `EPackage`: modela pacotes, contêineres para classificadores, *i.e.*, classes e tipos de dados. O nome de um pacote não precisa ser único, pois seu *namespace URI* é utilizado para identificá-lo. Esse URI é usado na serialização de documentos de instâncias, juntamente com o prefixo do *namespace*, para identificar o pacote da instância.

- `EFactory`: modela fábricas para criação de instâncias de objetos. As fábricas permitem a criação de operações para instanciar classes e para converter valores de dados para *strings* e vice-versa.

- `EAnnotation`: modela anotações para associar informações adicionais a um elemento do modelo.
- `EClassifier`: modela os tipos dos valores. É a classe-base comum dos tipos de dados e para classes que servem como tipos de um elemento tipado, sendo portanto a base comum para os atributos, referências, operações e parâmetros.
- `ENamedElement`: modela elementos que são nomeados. A maior parte dos elementos no Ecore são modelos que são identificados por um nome e, portanto, estendem essa classe.
- `ETypedElement`: modela elementos que são tipados, *e.g.*, atributos, referências, parâmetros e operações. Todos os elementos tipados possuem uma multiplicidade associada especificada por seus limites inferiores (`lowerBound`) e limites superiores (`upperBound`). Um limite inferior indefinido é especificado pelo valor `-1` ou pela constante `ETypedElement.UNBOUNDED_MULTIPLICITY`.
- `EStructuralFeature`: modela as características que possuem valores em uma classe. É a classe-base comum para atributos e referências. Os seguintes atributos booleanos são usados para caracterizar atributos e referências.
  - `Changed`: indica se o valor da característica pode ser modificado.
  - `Derived`: indica se o valor da característica deve ser computado por meio de outras características relacionadas.
  - `Transient`: indica se o valor da característica é omitido da serialização persistente do objeto.
  - `Unsettable`: indica se o valor da característica tem um estado não definido, em distinção do estado poder ser definido por qualquer valor específico.
  - `Volatile`: indica se a característica não possui campo de armazenamento gerado em sua classe de implementação.
- `EOperation`: modela operações que podem ser invocadas em uma classe específica. Uma operação é definida por um nome e uma lista de zero ou mais parâmetros tipados, representando sua assinatura. Assim como todos os elementos tipados, uma operação especifica um tipo, que representa o seu tipo de retorno, que pode ser `null` para representar nenhum valor de retorno. Uma operação também pode especificar zero ou mais exceções especificadas como classificadores que podem representar os tipos de exceções que podem ser lançadas.
- `EParameter`: modela um parâmetro de entrada de uma operação. Um parâmetro é identificado por um nome e, assim como todos os elementos tipados, especifica um tipo, que representa o tipo de valor que pode ser passado como argumento correspondendo àquele parâmetro.
- `EEnumLiteral`: modela os membros do conjunto de enumeração de valores literais. Uma enumeração literal é identificada por um nome e possui um valor inteiro associado, assim como um valor literal usado durante a serialização, que é o seu próprio nome caso este valor seja *null*.

- `EEnum`: modela tipos de enumeração, os quais especificam conjuntos de enumeração de valores literais.

4.3.2. *Eclipse Graphical Modeling Framework* (GMF)

O *Eclipse Graphical Modeling Framework* (GMF)[5] é um *framework* com base no IDE *Eclipse*, que permite a construção de editores gráficos para criação de modelos que estejam em conformidade com um metamodelo específico.

O GMF requer que alguns modelos específicos sejam criados para possibilitar a geração de um editor gráfico: `GMFGraph`, `GMFTool` e `GMFMap`.

- O **modelo gráfico** (`GMFGraph`) especifica os elementos gráficos (formas, conexões, rótulos, decorações, etc.) usados no editor.
- O **modelo da ferramenta** (`GMFTool`) especifica as ferramentas para criação de elementos que estarão disponíveis na paleta do editor.
- O **modelo de mapeamento** (`GMFMap`) mapeia os elementos gráficos no modelo gráfico e as ferramentas de criação no modelo da ferramenta com a sintaxe abstrata dos elementos do metamodelo Ecore (classes, atributos, referências, etc.).

Assim que os três modelos referidos anteriormente estão criados, o GMF oferece funcionalidades para transformações M2M (modelo-para-modelo ou *model-to-model*) para criação do modelo gerador (`GMFGen`), que oferece customizações adicionais para o editor. Então, por último, transformações M2T (modelo-para-texto ou *model-to-text*) produzem um novo projeto (`.diagram`), que contém código em linguagem Java para instanciação do editor. O GMF oferece um assistente, chamado *GMF dashboard* e apresentado na Figura 4.3, para geração semi-automática das versões iniciais dos modelos necessários. A Figura 4.4 oferece uma visão geral do processo de criação destes modelos, na forma de um Diagrama de Atividades da UML.

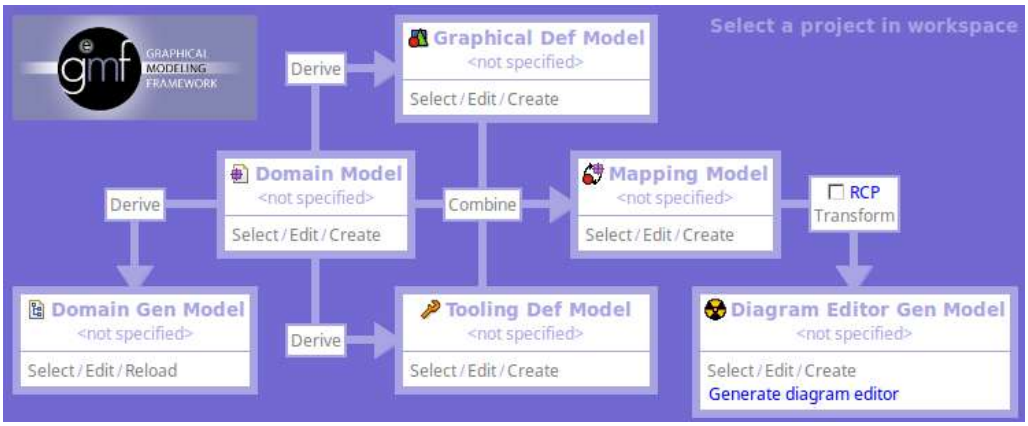


Figura 4.3. *GMF dashboard*: assistente para criação dos modelos GMF.

[5] `https://www.eclipse.org/gmf-tooling/`

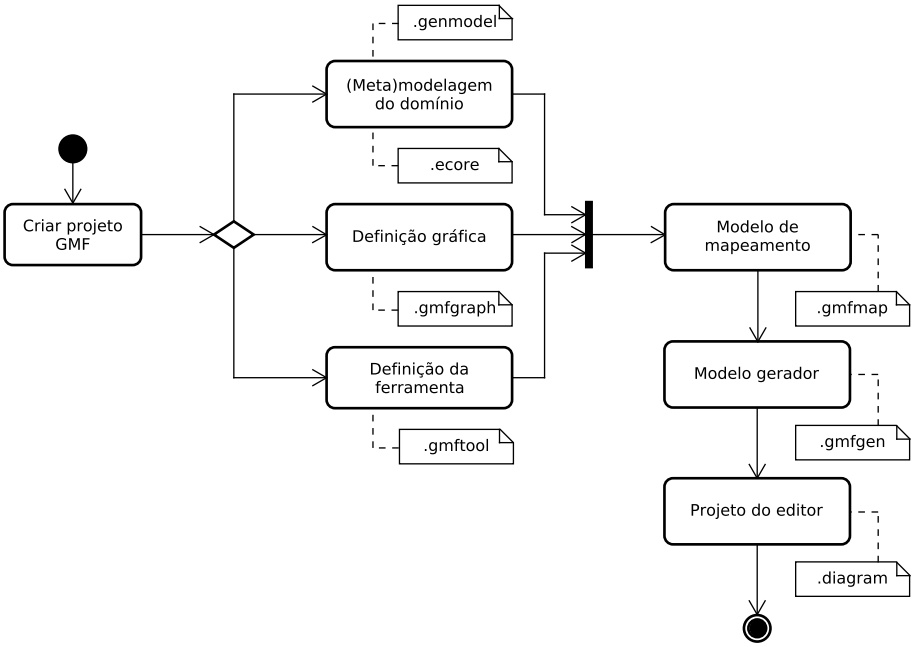


Figura 4.4. Processo de criação dos modelos GMF. Adaptado de [Eclipse Foundation, The 2015e].

Os modelos gerados podem requerer pequenos ajustes. Tais customizações são feitas editando manualmente os modelos com auxílio de um editor de textos ou do editor em árvore, ambos nativos do EMF. A Figura 4.5 apresenta a edição do modelo `GMFMap` utilizando o editor em árvore, enquanto a Figura 4.6 apresenta a edição do modelo usando o editor de texto. Além disso, a cada alteração no metamodelo Ecore, todos os modelos GMF devem ser gerados novamente, pois o GMF não oferece mecanismos para atualizar seus modelos automaticamente, diferentemente do EMF [Kolovos et al. 2015a].

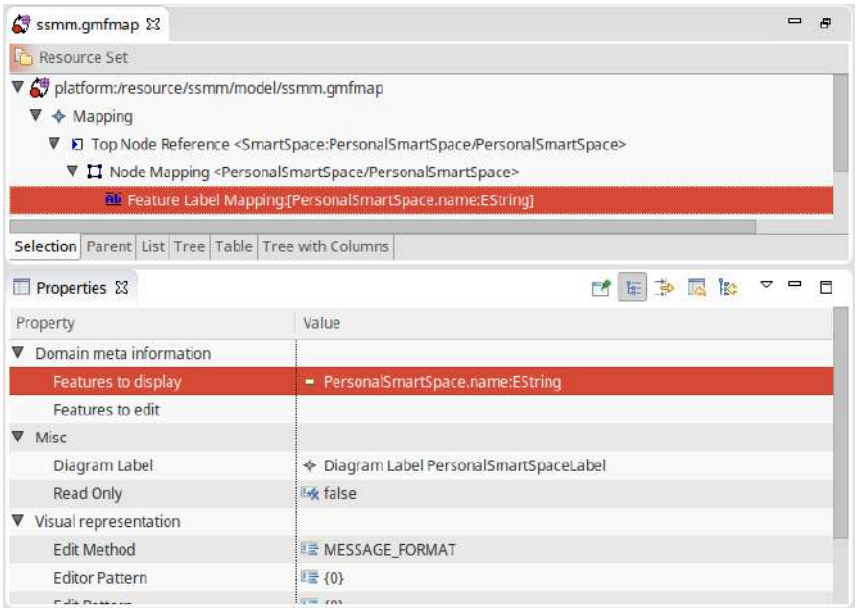


Figura 4.5. Edição do `GMFMap` usando editor em árvore.

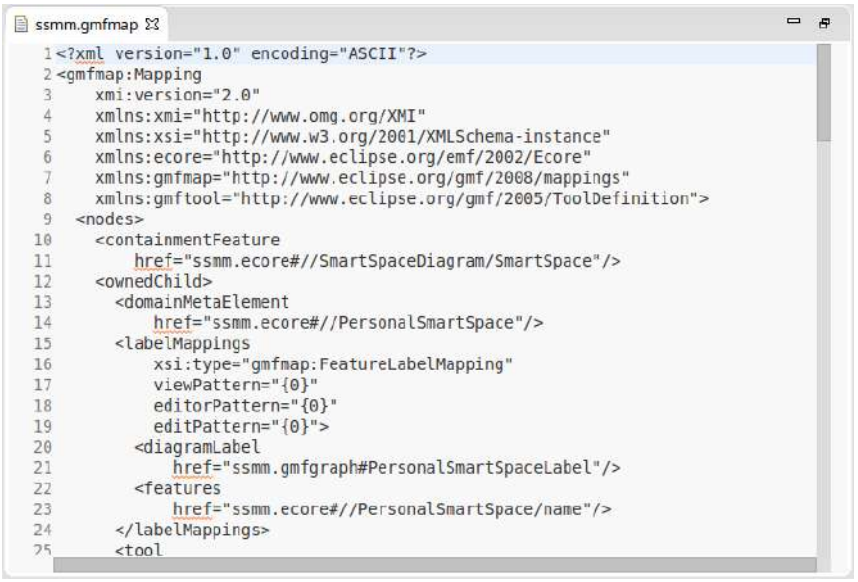


Figura 4.6. Edição do `GMFMap` usando editor de texto.

4.3.3. *Epsilon*

Epsilon[6] é o acrônimo para Plataforma Extensível de Linguagens Integradas para Gerenciamento de Modelos (em inglês, *Extensible Platform of Integrated Languages for mOdel maNagement*) [Kolovos et al. 2015b]. Trata-se de uma família de linguagens e ferramentas de suporte para atividades de manipulação de modelos, tais como, geração de código, transformação, comparação, união, refatoramento e validação de modelos.

Atualmente, as seguintes linguagens compõem a *Epsilon*:

- *Epsilon Object Language* (EOL)
- *Epsilon Validation Language* (EVL)
- *Epsilon Transformation Language* (ETL)
- *Epsilon Comparison Language* (ECL)
- *Epsilon Merging Language* (EML)
- *Epsilon Wizard Language* (EWL)
- *Epsilon Generation Language* (EGL)

Para cada uma de suas linguagens, existem ferramentas de desenvolvimento com base no IDE *Eclipse*, que oferecem funcionalidades como destaque de sintaxe e manipulação de erros, além de um interpretador para a linguagem. Os interpretadores de linguagens *Epsilon* podem inclusive ser executados de maneira independente em aplicações Java ou Android [Kolovos et al. 2015b], por meio de suas Interfaces de Programação

[6] `https://www.eclipse.org/epsilon/`

de Aplicações (*Application Programming Interface* - API)[7], possibilitando seu uso em aplicações que não foram construídas com uso do ambiente de desenvolvimento *Epsilon*.

Nas subseções seguintes, encontra-se uma breve descrição das duas primeiras linguagens (EOL e EVL), além da ferramenta *Eugenia*, que também compõe a família *Epsilon*. Essas tecnologias são utilizadas na construção da ferramenta de modelagem gráfica que é objetivo deste minicurso. Informações detalhadas sobre as linguagens *Epsilon* podem ser obtidas em [Kolovos et al. 2015b].

4.3.4. *Epsilon Object Language* - EOL

A linguagem EOL[8] é o núcleo das linguagens da família *Epsilon*. Todas as demais linguagens da família estendem a EOL, tanto em sintaxe quanto em semântica. Sendo assim, a EOL oferece um conjunto de funcionalidades sobre as quais as demais linguagens são implementadas. Além disso, a EOL pode ser usada independentemente, como uma linguagem de propósito geral para gerenciamento de modelos, automatizando tarefas que não são específicas das demais linguagens da família, tais como criar, consultar e modificar modelos EMF [Kolovos et al. 2015b].

Dentre as principais características da EOL, destacam-se [Eclipse Foundation, The 2015b]:

- Todas as construções usuais em programação, como laços `while` e `for`, variáveis, etc.
- Possibilidade de criar e realizar chamadas a métodos de objetos Java.
- Suporte para a adição dinâmica de operações em metaclasses em tempo de execução.
- Suporte a interação com o usuário.
- Possibilidade de criação de bibliotecas de operações para serem importadas e utilizadas em outras linguagens *Epsilon*.

Programas escritos em EOL são organizados em módulos (*modules*), conforme ilustrado pela Figura 4.7. Cada módulo define um corpo (*body*) e um número de operações (*operations*). O corpo é um bloco de declarações que são avaliadas quando o módulo é executado. Cada operação define o tipo de objeto sobre o qual ela é aplicável (*contexto*), um nome (*name*), um conjunto de parâmetros (*parameters*) e um tipo de retorno (*return type*), que é opcional. Módulos também podem importar (*import*) outros módulos.

Além de possibilitar a construção de operações, a linguagem EOL também oferece uma série de operações previamente definidas para cada um de seus tipos de dados, apresentados na Figura 4.8, além de permitir a utilização de tipos nativos (*native*), definidos pelo usuário em sua linguagem-base, por exemplo, uma classe Java. O Código 4.1 apresenta um exemplo de utilização da linguagem EOL.

[7] `https://www.eclipse.org/epsilon/download/`
[8] `https://www.eclipse.org/epsilon/doc/eol/`

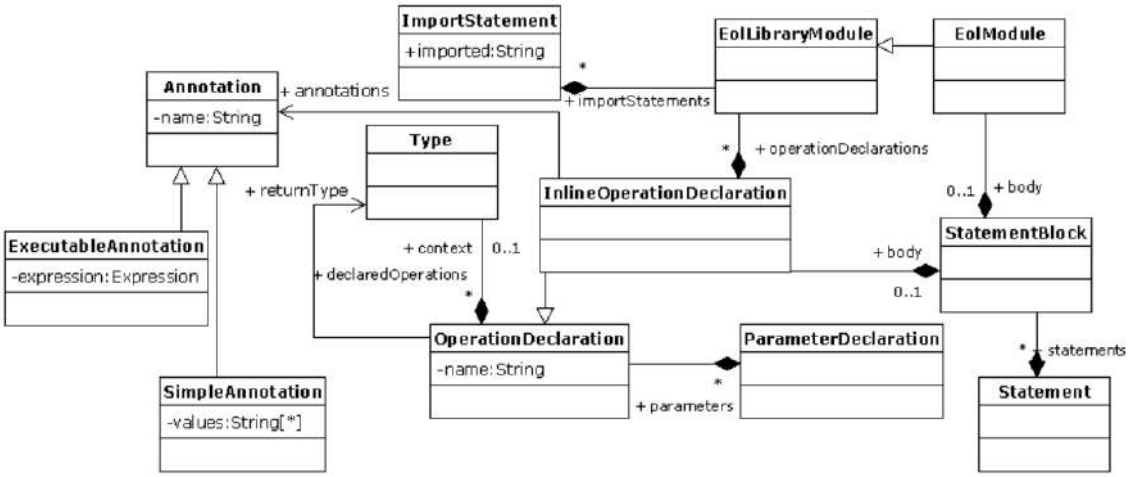


Figura 4.7. Estrutura de módulos da linguagem EOL [Kolovos et al. 2015b].

Código 4.1. Exemplo de código em linguagem EOL para calcular e imprimir a profundidade de cada árvore (`Tree`) [Kolovos et al. 2015b].

```
1  var depths = new Map;
2
3  for (n in Tree.allInstances.select(t|not t.parent.isDefined()))
       {
4      n.setDepth(0);
5  }
6
7  for (n in Tree.allInstances) {
8      (n.name + " " + depths.get(n)).println();
9  }
10
11 operation Tree setDepth(depth : Integer) {
12     depths.put(self,depth);
13     for (c in self.children) {
14         c.setDepth(depth + 1);
15     }
16 }
```

4.3.5. *Epsilon Validation Language* - EVL

O objetivo da linguagem EVL[9] é oferecer funcionalidades de validação à família *Epsilon*. Dessa forma, a linguagem EVL pode ser utilizada para especificar e avaliar restrições – também chamadas de invariantes – em modelos de um metamodelo específico. As restrições escritas em EVL são similares às restrições OCL (*Object Constraint Language*) [Object Management Group 2015], contudo, a linguagem EVL também oferece suporte a dependências entre as restrições (*e.g.*, se a restrição A falhar, então não avalie a restrição B), mensagens de erro customizáveis e a especificação de consertos (*quick fixes*) que podem ser acionados para reparar inconsistências [Eclipse Foundation, The 2015c].

[9] `https://www.eclipse.org/epsilon/doc/evl/`

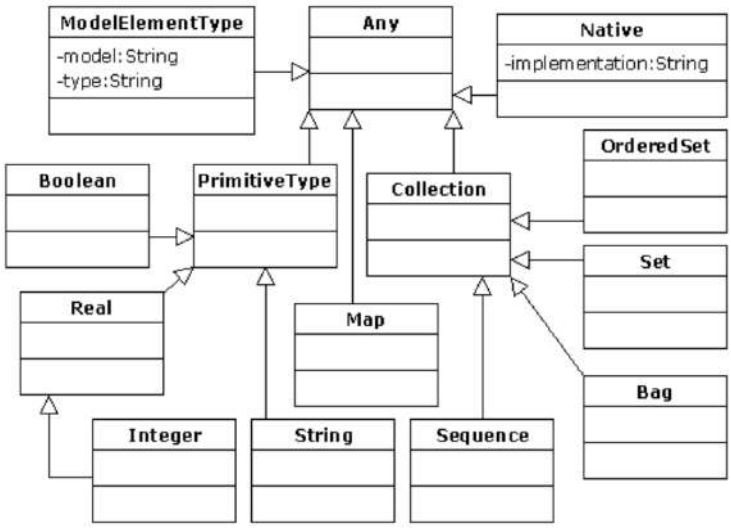


Figura 4.8. Visão geral do sistema de tipos de dados da linguagem EOL [Kolovos et al. 2015b].

As principais características da linguagem EVL são, dentre outras [Eclipse Foundation, The 2015c]:

- Distinção entre erros (*error*) e avisos (*warning*) durante a validação.
- Especificação de consertos (*quick fix*) para erros encontrados pelas restrições.
- Todas as funcionalidades oferecidas pela linguagem EOL.
- Suporte a operações lógicas de primeira ordem oriundas da OCL (`select`, `reject`, `collect`, etc.).
- Suporte a interação com usuário.

Em EVL, as especificações das validações são organizadas em módulos (`EvlModule`). Além de operações, um módulo EVL também pode conter um conjunto de invariantes agrupadas pelo contexto no qual serão aplicadas, juntamente com um número de blocos anteriores (`pre`) e posteriores (`post`). A Figura 4.9 ilustra os elementos da sintaxe abstrata da linguagem EVL, descritos a seguir.

- `Context`: um contexto especifica o tipo de instâncias sobre as quais as invariantes (`invariant`) serão avaliadas. Cada contexto pode opcionalmente definir uma guarda (`guard`) que limita sua aplicabilidade para um subconjunto menor de instâncias de um tipo especificado. Dessa forma, se uma guarda falha para uma instância de um tipo específico, nenhuma de suas invariantes é avaliada.
- `Invariant`: como ocorre em OCL, cada invariante EVL define um nome (`name`) e um corpo (`check`). Entretando, ela pode opcionalmente definir uma guarda (`guard`). Como forma de oferecer *feedback* para o usuário, cada invariante pode definir uma mensagem (`message`) que contém uma descrição da razão pela qual

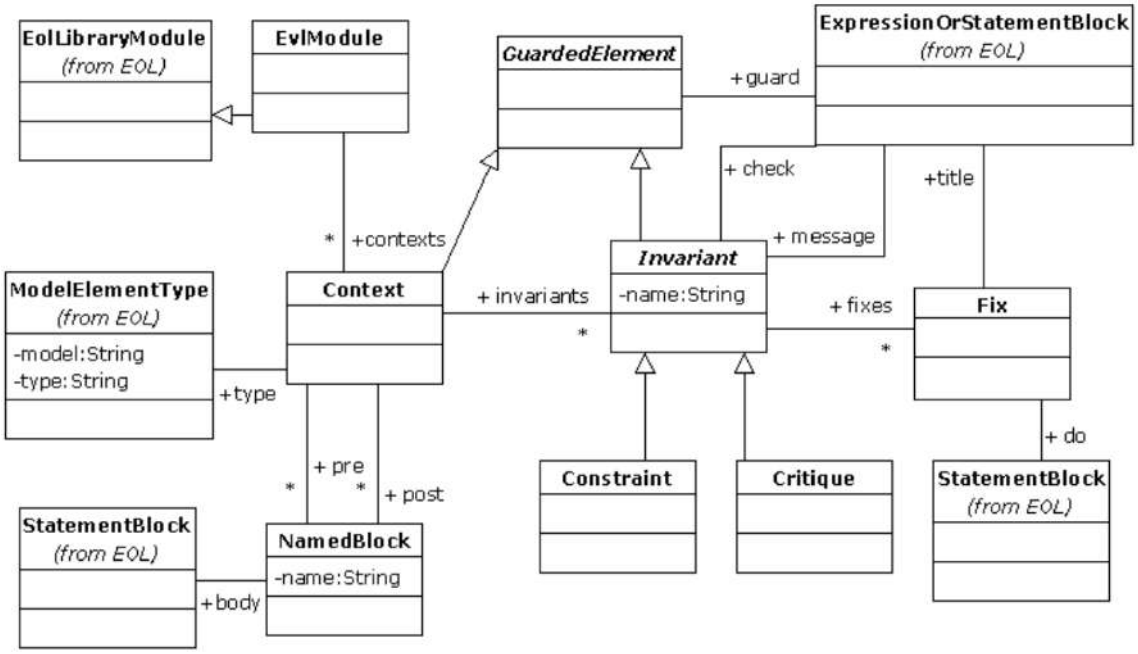


Figura 4.9. Sintaxe abstrata da linguagem EVL [Kolovos et al. 2015b].

cada restrição falhou em um elemento em particular. Para permitir conserto semi-automático de erros, uma invariante pode opcionalmente definir um ou mais consertos (`fixes`), tratados no *Eclipse* como *"quick fix"*. Cada invariante pode ser uma restrição (`constraint`) ou crítica (`critique`), permitindo lançar mensagens de erro (*error*) ou aviso (*warning*), respectivamente.

- `Guard`: guardas são usadas para limitar a aplicabilidade das invariantes.
- `Fix`: permite a correção semi-automática de erros encontrados durante a validação. É possível definir um título (`title`) e um bloco `do`, onde a funcionalidade de conserto será definida usando a linguagem EOL.
- `Constraint`: restrições em EVL são usadas para capturar erros críticos que invalidam o modelo. Restrições (`constraint`) são uma subclasse das invariantes (`invariant`), herdando, portanto, todas as suas características.
- `Critique`: em distinção às restrições, críticas (`critiques`) são usadas para capturar situações que não invalidam o modelo, mas devem ser consideradas para melhorar sua qualidade. Assim como as restrições, é uma subclasse das invariantes (`invariant`).
- `Pre` e `Post`: contém declarações EOL que podem ser executadas antes (`pre`) ou após (`post`) a avaliação das invariantes.

O Código 4.2 apresenta um exemplo de regra EVL que impede que uma metaclasse em particular possua uma referência para si mesma, sugerindo como correção (*quick fix*) apagar o autorreferenciamento encontrado, conforme ilustrado na Figura 4.12.

Código 4.2. Exemplo de código em linguagem EVL para impedir auto-referenciamento em uma determinada metaclasse.

```
1  context HasSubFSS {
2    constraint CannotSelfReference {
3      check : self.source.name <> self.target.at(0).name
4      message : 'Cannot make a self reference'
5        fix {
6          title : "Delete self reference"
7          do {
8            delete self;
9          }
10       }
11   }
12 }
```

Figura 4.10. Erro encontrado no modelo por uma regra escrita em linguagem EVL.

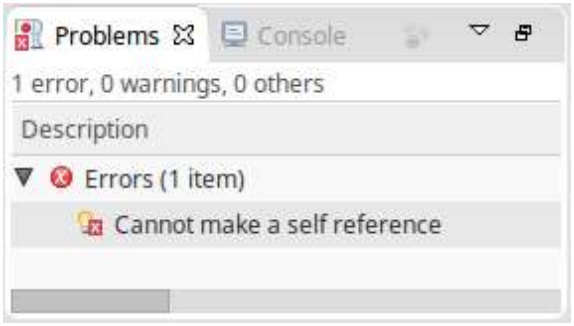


Figura 4.11. Descrição do erro.

4.3.6. *Eugenia*

Eugenia[10] é uma ferramenta da família *Epsilon* que gera automaticamente os modelos `GMFGraph`, `GMFTool` e `GMFMap`, necessários para a implementação de um editor GMF, utilizando como base em um metamodelo Ecore anotado, escrito em linguagem *Emfatic*[11]. A linguagem *Emfatic* permite a representação de metamodelos Ecore de forma textual e possui sintaxe similar à da linguagem Java [Daly and Eclipse Foundation, The 2015a, Kolovos et al. 2015a]. A IDE *Epsilon* possui funcionalidades que permitem a transformação de modelos Ecore em código *Emfatic* e vice-versa.

O principal objetivo da *Eugenia* é diminuir a complexidade na criação de ferramentas de modelagem gráficas com base no GMF, por meio de anotações de alto nível,

[10] `https://www.eclipse.org/epsilon/doc/eugenia/`
[11] `https://www.eclipse.org/epsilon/doc/articles/emfatic/`

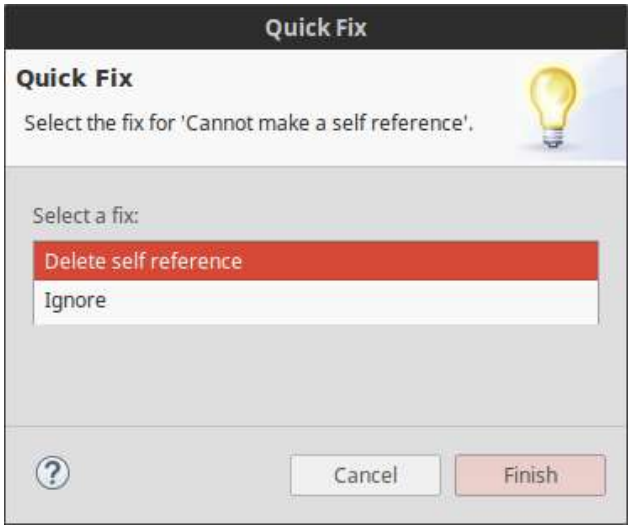


Figura 4.12. Sugestão de correção (*quick fix*).

feitas diretamente no código *Emfatic* [Daly and Eclipse Foundation, The 2015b]. Wienands e Golm [Wienands and Golm 2009] realizaram um experimento industrial e chegaram à conclusão de que implementar um editor gráfico utilizando puramente EMF e GMF é um processo propenso a erros, principalmente por requerer que os desenvolvedores escrevam e mantenham, em baixo nível, uma série de complexas relações entre os modelos GMF. Além disso, "mais desafiador do que construir um editor GMF, é sua manutenibilidade, uma vez que o GMF, ao contrário do EMF, não oferece mecanismos para atualizar seus modelos automaticamente quando o metamodelo é modificado" [Kolovos et al. 2015a].

A *Eugenia* permite transformações automatizadas entre o código *Emfatic* anotado, que representa o metamodelo, e os modelos de baixo nível necessários para geração da ferramenta GMF, aumentando o nível de abstração no qual os desenvolvedores devem trabalhar para construir suas ferramentas gráficas para edição de modelos com base em um metamodelo [Kolovos et al. 2015a]. Em suma, a *Eugenia* oferece seis categorias de anotações [Kolovos et al. 2015a], que são utilizadas nos códigos *Emfatic* para guiar a criação da ferramenta gráfica de modelagem. Os atributos suportados em cada tipo de anotação e sua respectiva lista de valores não serão aqui abordados, podendo, entretanto, ser obtidos em [Eclipse Foundation, The 2015d]. A seguir está a lista de anotações suportadas pela *Eugenia*:

- `@gmf.diagram`: usada para especificar configurações do diagrama, tais como o tipo do elemento raiz do modelo e a extensão do arquivo do editor gráfico.
- `@gmf.node`: usada para indicar quais elementos da sintaxe abstrata irão representar nós (vértices) na sintaxe gráfica, além da sua forma, cor, tamanho, rótulo, etc.
- `@gmf.link`: usada para indicar quais elementos da sintaxe abstrata irão representar arestas na sintaxe gráfica, além de especificar sua espessura, cor, estilo, tipo de pontas das setas, rótulos, etc.

- `@gmf.compartment`: são usadas para indicar quais nós podem ser aninhados dentro de outros nós na sintaxe gráfica (*e.g.*, atributos são aninhados dentro das classes em um Diagrama de Classes da UML).
- `@gmf.affixed`: são usadas para indicar quais nós devem estar anexos à borda de outros nós na sintaxe gráfica.
- `@gmf.label`: usada para especificar rótulos adicionais para um nó na sintaxe gráfica.

O Código 4.3 traz o código *Emfatic* de um metamodelo para representar sistemas de arquivos [Eclipse Foundation, The 2015d]. Por meio deste código e suas anotações, é possível gerar um editor GMF completamente funcional utilizando a ferramenta *Eugenia*, o qual é apresentado na Figura 4.13.

Código 4.3. Código em linguagem Emfatic de um metamodelo de sistemas de arquivos [Eclipse Foundation, The 2015d].

```
1  @namespace(uri="filesystem", prefix="filesystem")
2  @gmf
3  package filesystem;
4
5  @gmf.diagram
6  class Filesystem {
7      val Drive[*] drives;
8      val Sync[*] syncs;
9  }
10
11 class Drive extends Folder {
12
13 }
14
15 class Folder extends File {
16     @gmf.compartment
17     val File[*] contents;
18 }
19
20 class Shortcut extends File {
21     @gmf.link(target.decoration="arrow", style="dash")
22     ref File target;
23 }
24
25 @gmf.link(source="source", target="target", style="dot", width=
       "2")
26 class Sync {
27     ref File source;
28     ref File target;
29 }
30
31 @gmf.node(label = "name")
32 class File {
33     attr String name;
34 }
```

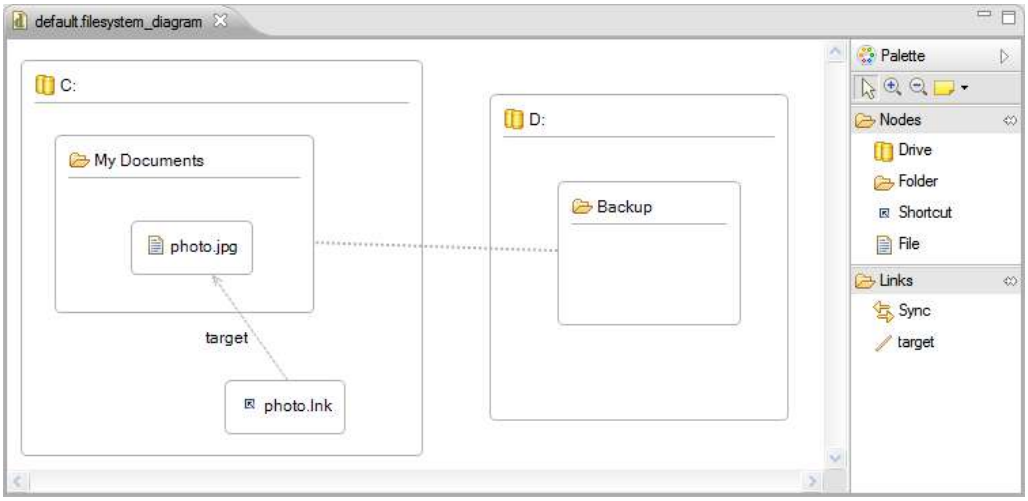


Figura 4.13. Editor GMF gerado com apoio da ferramenta Eugenia [Eclipse Foundation, The 2015d].

Por fim, a *Eugenia* oferece meios de realizar ajustes finos na aparência da ferramenta de modelagem, por meio da definição de regras de transformação em modelos independentes (*e.g.*, `ECore2GMF.eol`), utilizando a linguagem EOL. As regras desses modelos são executadas logo após a derivação dos modelos GMF pela *Eugenia*. Sendo assim, ao contrário do que ocorre quando se trabalha diretamente com GMF, não é necessário realizar os ajustes na aparência da ferramenta sempre que houver novas mudanças no metamodelo. O Código 4.4 apresenta o conteúdo do arquivo `ECore2GMF.eol`, cujo objetivo é remover as bordas das figuras nos modelos gerados em um editor GMF, o qual foi implementado com apoio da ferramenta *Eugenia*.

Código 4.4. Código em linguagem EOL (`ECore2GMF.eol`) para remover as bordas das figuras dos modelos, em um editor GMF criado com auxílio da Eugenia [Eclipse Foundation, The 2015a].

```
1 -- Find the attribute figure
2 var attributeFigure = GmfGraph!Rectangle.all.
3     selectOne(r|r.name = 'AttributeFigure');
4
5 -- ... delete its border
6 delete attributeFigure.border;
```

4.4. Tutorial para Construção de Ferramenta de Modelagem com base em um Metamodelo Ecore

Este tutorial tem objetivo de sintetizar os conhecimentos adquiridos nas seções anteriores na forma de um produto de software que consiste em uma ferramenta de modelagem para produzir modelos em conformidade com um determinado metamodelo Ecore.

Dessa forma, considera-se que o leitor tenha conhecimento prévio das tecnologias EMF, GMF e da família de linguagens *Epsilon*, apresentadas nas Subseções 4.3.1, 4.3.2 e 4.3.3, respectivamente. Será utilizado o IDE *Eclipse Neon* (versão 4.6), juntamente com

a *Epsilon* versão 1.4[12].

O tutorial está organizado da seguinte forma. Na Subseção 4.4.1 é apresentado o passo-a-passo para construção da ferramenta de modelagem gráfica. A Subseção 4.4.2 traz informações sobre como fazer refinamentos estéticos na ferramenta, a saber: *(i)* adicionar ícones para os nós, *(ii)* modificar os ícones da paleta (barra de ferramentas) e *(iii)* mudar o estilo da fonte dos rótulos. A Subseção 4.4.3 apresenta como adicionar regras de validação em linguagem EVL. A Subseção 4.4.4 traz algumas lições aprendidas para contornar comportamentos errôneos e inesperados do IDE. Por último, a Subseção 4.4.5, apresenta brevemente um exemplo de ferramenta de modelagem gráfica construída com os conhecimentos adquidiros neste minicurso.

4.4.1. Construção da Ferramenta Gráfica

Esta subseção apresenta o passo-a-passo para construção da ferramenta de modelagem gráfica.

Construção do metamodelo Ecore

1. Criar o projeto para o metamodelo Ecore
   (a) Acessar o menu "*File*" > "*New*" > "*Other...*"
   (b) Selecionar "*Ecore Modeling Project*"
   (c) Clicar em "*Next*"
   (d) Digitar um nome para o projeto e clicar em "*Finish*"
   (e) Aceitar a mudança da perspectiva, caso lhe seja sugerido
2. Construir um metamodelo válido
   (a) Construir o metamodelo
      i. Utilizar as ferramentas da paleta para construir as metaclasses e as suas relações
      ii. Dica: adicione um atributo "*name*", do tipo `EString`, para cada metaclasse que represente um nó.
   (b) Validar o metamodelo
      i. Assim que terminado, certifique-se de que seu metamodelo seja válido, acessando o menu "*Diagram*" > "*Validate*"

Conversão do arquivo Ecore (`.ecore`) para Emfatic (`.emf`)

1. Criar um novo projeto para a ferramenta

[12] `https://www.eclipse.org/epsilon/download/`

(a) Acessar o menu "*File*" > "*New*" > "*Project*"

(b) Selecionar a opção "*General*" > "*Project*"

(c) Definir nome do projeto. Exemplo: "*minhaFerramenta*"

(d) Clicar em "*Finish*"

2. Criar pasta para armazenar os (meta)modelos

   (a) Selecionar pasta raiz do projeto recém criado

   (b) Acessar o menu "*File*" > "*New*" > "*Folder*"

   (c) Nomear a pasta como "*model*"

   (d) Clicar em "*Finish*"

3. Copiar o arquivo `.ecore` do projeto do metamodelo para o projeto da ferramenta

   (a) Navegar até a pasta "*model*" no projeto do metamodelo já criado

   (b) Selecionar o arquivo `.ecore`

   (c) Acessar o menu "*Edit*" > "*Copy*"

   (d) Navegar até a pasta "*model*" no projeto da ferramenta

   (e) Acessar o menu "*Edit*" > "*Paste*"

4. Gerar arquivo Emfatic (`.emf`) a partir do Ecore (`.ecore`)

   (a) Clicar com o botão direito do mouse sobre o arquivo *.ecore*

   (b) Selecionar "*Generate Emfatic Source*"

5. Adicionar as anotações necessárias ao arquivo `.emf`

   (a) Clicar duas vezes sobre o arquivo `.emf` e adicionar as anotações necessárias

   (b) A Subseção 4.3.6 contém mais detalhes sobre as anotações disponíveis

Geração da ferramenta gráfica

1. Gerar a ferramenta de modelagem

   (a) Clicar com o botão direito do mouse sobre o arquivo `.emf`

   (b) Selecionar "*Eugenia*" > "*Generate GMF Editor*"

   (c) Aguardar a mensagem de conclusão: "*Code generation completed successfully*"

   (d) Clicar em "*OK*"

   - Ao final, o Eugenia irá criar quatro novos projetos, com terminação `.diagram`, `.edit`, `.editor` e `.tests`, além dos modelos `GenModel`, `GMFGen`, `GMFGraph`, `GMFTool` e `GMFMap`, necessários para o GMF instanciar a ferramenta de modelagem, conforme detalhado na Subseção 4.3.6.

Acessando a ferramenta de modelagem gráfica

1. Iniciar a ferramenta de modelagem
   (a) Clicar com o botão direito na pasta raiz do projeto da ferramenta
   (b) Selecionar "*Run As*" > "*Eclipse Application*"
2. Criar projeto na ferramenta de modelagem para armazenar o(s) modelo(s) em conformidade com o metamodelo
   (a) Acessar o menu "*File*" > "*New*" > "*Project*"
   (b) Selecionar a opção "*General*" > "*Project*"
   (c) Definir nome do projeto. Exemplo: "*meuModelo*"
   (d) Clicar em "*Finish*"
3. Criar modelo na ferramenta de modelagem
   (a) Acessar o menu "*File*" > "*New*" > "*Example...*"
   (b) Selecionar a opção que representa o seu metamodelo (1^a opção) > "*Next*"
   (c) Definir o nome do modelo ou aceitar a sugestão
   (d) Clicar em "*Finish*"

4.4.2. Customizando a Ferramenta de Modelagem

Esta subseção apresenta como realizar refinamentos estéticos na ferramenta de modelagem. A Subseção 4.4.2.1 detalha como adicionar figuras em formato vetorial para representar os nós; a Subseção 4.4.2.2 apresenta como modificar os ícones da paleta (barra de ferramentas); e a Subseção 4.4.2.3 como usar regras de transformação escritas em linguagem EOL para modificar o estilo da fonte dos rótulos dos nós e vértices dos modelos construídos na ferramenta de modelagem.

4.4.2.1. Adicionar Ícones SVG para Representar as `EClasses`

Esta subseção demonstra como utilizar figuras vetoriais em formato SVG[13] para representar os nós, como alternativa às formas padrão (*e.g.*, retângulo (*rectangle*), elipse (*ellipse*) e retângulo com cantos arredondados (*rounded*)).

Considera-se que os ícones SVG já foram construídos pelo leitor. Existem vários *websites*[14] que disponibilizam ícones SVG gratuitos. A ferramenta *Inkscape*[15] pode ser usada para construir ou editar os ícones.

[13] `https://pt.wikipedia.org/wiki/SVG`
[14] *e.g.*, `http://www.flaticon.com/`, `http://www.freepik.com/free-icons`
[15] `https://inkscape.org/pt/`

1. Criar pasta para armazenar os ícones SVG
   (a) Selecionar pasta raiz do projeto da ferramenta
   (b) Acessar o menu "*File*" > "*New*" > "*Folder*"
   (c) Nomear a pasta como "*icons*"
   (d) Clicar em "*Finish*"
2. Adicionar os ícones em formato SVG à pasta recém criada
3. Adicionar a pasta de ícones ao *build path* do projeto
   (a) Abrir o arquivo "*build.properties*"
   (b) Selecionar a pasta "*icons*" em "*Binary Build*"
   (c) Salvar o arquivo "*build.properties*": Acessar o menu "*File*" > "*Save*"
4. Atualizar as anotações dos nós no arquivo `.emf`
   (a) Exemplo:
   ```
   @gmf.node(figure="svg", label.icon="false", label="name",
   svg.uri="platform:/plugin/NOMEDOSEUMETAMODELO/icons/NOMEDOICONE.svg")
   ```
5. Gerar novamente a ferramenta de modelagem
6. Iniciar a ferramenta de modelagem

4.4.2.2. Modificar os Ícones da Paleta

É possível modificar os ícones padrões da paleta (barra de ferramentas) da ferramenta de modelagem para melhor representar os nós (*Objects*) e vértices (*Connections*). Os ícones devem estar em formato GIF[16]. Os aplicativos Inkscape ou Gimp[17] podem ser usados para criar ou editar figuras nesse formato.

1. Adicionar os ícones em formato .GIF à pasta
   `/NOMEDOSEUMETAMODELO.edit/icons/full/obj16`
   (a) Cada arquivo de ícone deve ter exatamente o mesmo nome da metaclasse que irá representar, acompanhado da extensão `.gif`. Exemplo: "*Person.gif*"
2. Gerar novamente a ferramenta de modelagem

[16] `https://pt.wikipedia.org/wiki/Graphics_Interchange_Format`
[17] `https://www.gimp.org/`

4.4.2.3. Alterar o Estilo da Fonte dos Rótulos

Esta subseção apresenta como modificar o estilo dos rótulos dos nós e vértices dos modelos construídos na ferramenta de modelagem. Para tal, faz-se necessário escrever regras de transformação utilizando a linguagem EOL, detalhada na Subseção 4.3.4.

1. Criar um arquivo chamado "*ECore2GMF.eol*" na mesma pasta onde se encontra o arquivo `.emf`
2. Editar o arquivo "*ECore2GMF.eol*", de acordo com o desejado. Exemplos:
   - Mudar o texto do rótulo para itálico

```
1  var label = GmfGraph!Label.all.selectOne(l|l.name = '
       NOMEDOROTULO');
2  label.font = new GmfGraph!BasicFont;
3  label.font.style = GmfGraph!FontStyle#ITALIC;
```

   - Mudar o texto do rótulo para negrito

```
1  var label = GmfGraph!Label.all.selectOne(l|l.name = '
       NOMEDOROTULO');
2  label.font = new GmfGraph!BasicFont;
3  label.font.style = GmfGraph!FontStyle#ITALIC;
```

3. Gerar novamente a ferramenta de modelagem

Obter o nome do rótulo que deseja alterar

1. Abrir o modelo `.gmfgraph` com Exeed
   (a) Clicar com o botão direito sobre o arquivo `.gmfgraph`
   (b) Acessar "*Open With*" > "*Other...*"
   (c) Selecionar "*Exeed Editor*" e clicar em "*OK*"
2. Acessar menu "*Exeed*" > "*Show Structural Info*"
3. Navegar pelos nós "*Canvas*" > "*Figure Gallery Default*"
4. Expandir o rótulo que deseja alterar. Deve ser algo como "*Figure Descriptor NOMEDAMETACLASSELabelFigure*"
5. Acessar as propriedades do "filho" do tipo `Label` e clicar com o botão direito do mouse e acessar "*Show Properties View*"
6. Copiar o valor da propriedade "*name*", que corresponde ao nome do rótulo. Por exemplo "*NOMEDAMETACLASSELabelFigure*"

4.4.3. Regras de Validação em Linguagem EVL

Esta subseção descreve como adicionar regras de validação adicionais aos modelos gerados com a ferramenta de modelagem gráfica, utilizando a linguagem EVL, descrita na Subseção 4.3.5.

1. Crie um novo *plugin* para guardar o arquivo `.evl`
   - Acesse o menu "*File*" > "*New*" > "*Other...*"
   - Escolha a opção "*Plug-in Project*" e clique em "*Next*"
   - Escolha um nome para o *plugin*. Exemplo: "*NOMEDOSEUMETAMODELO.validation*"
   - Clique em "*Next*" e, em seguida, em "*Finish*"
2. Configure as dependências
   (a) Abra o arquivo "*MANIFEST.MF*", que se encontra na pasta "*META-INF*"
   (b) Acesse a aba "*Dependencies*"
   (c) Adicione à lista de dependências:
      i. `org.eclipse.ui.ide`
      ii. `org.eclipse.epsilon.evl.emf.validation`
3. Crie uma pasta chamada "*validation*" na raiz do projeto do *plugin*
4. Crie, dentro dessa pasta, um arquivo `.evl` para escrever suas regras de validação.
   (a) Clique com o botão direito sobre a pasta "*validation*"
   (b) Acesse o menu "*New*" > "*File*"
   (c) Escreva o nome do arquivo. Exemplo: `NOMEDOSEUMETAMODELO.evl`
   (d) Adicione nesse arquivo as regras de validação utilizando a linguagem EVL. Exemplo:

```
1  context MetaClasseX {
2    constraint CannotSelfReference {
3      check : self.source.name <> self.target.at(0).name
4      message : 'Cannot make a self reference'
5        fix {
6          title : "Delete self reference"
7          do {
8            delete self;
9          }
10       }
11   }
12 }
```

5. Vincule as regras de validação à ferramenta de modelagem

(a) Abra o arquivo "*MANIFEST.MF*", que se encontra na pasta "*META-INF*"

(b) Acesse a aba "*Extensions*"

(c) Adicione a extensão: `org.eclipse.epsilon.evl.emf.validation`

i. Clique com o botão direito do mouse na extensão adicionada

ii. Acesse "*New*" > "*constraintsBinding*"

iii. Selecione o "*constraintsBinding*" adicionado e edite seus valores:

A. "*namespaceURI*": deve ser o mesmo do projeto da ferramenta de modelagem.
Exemplo: `http://www.example.org/NOMEDOSEUMETAMODELO`

B. "*constraints*": localização do arquivo que contém as regras EVL, criado no passo 4.
Exemplo: `validation/NOMEDOSEUMETAMODELO.evl`

(d) Adicione a extensão: `org.eclipse.ui.ide.markerResolution`

i. Clique com o botão direito do mouse na extensão adicionada

ii. Acesse "*New*" > "*markerResolutionGenerator*". Faça isso duas vezes.

iii. Selecione o primeiro "*markerResolutionGenerator*" e edite seus valores:

A. "*class*": `org.eclipse.epsilon.evl.emf.validation.EvlMarkerResolutionGenerator`

B. "*markerType*": `NOMEDOSEUMETAMODELO.diagram.diagnostic`

iv. Selecione o segundo "*markerResolutionGenerator*" e edite seus valores:

A. "*class*": `org.eclipse.epsilon.evl.emf.validation.EvlMarkerResolutionGenerator`

B. "*markerType*": `org.eclipse.emf.ecore.diagnostic`

6. Teste as regras de validação

(a) Inicie a ferramenta de modelagem

(b) Crie um modelo que inflija alguma regra de validação definida

(c) Valide o modelo acessando o menu "*Diagram*" > "*Validate*"

- Caso sejam encontrados erros, eles estarão visíveis na *view Problems*. Para visualizá-la acesse o menu "*Window*" > "*Show View*" > "*Problems*"

4.4.4. Lições Aprendidas

Nesta subseção estão descritas possíveis soluções para problemas comuns que podem ser encontrados no processo de desenvolvimento da ferramenta de modelagem gráfica.

1. Gerar a ferramenta de modelagem a cada nova mudança nos arquivos do projeto da ferramenta

2. Funcionamento anormal da ferramenta de modelagem (1)

(a) Apagar todos os projetos gerados (`.diagram`, `.edit`, `.editor` e `.tests`)

(b) Apagar todos os arquivos da pasta model, exceto o `.ecore` e o `.emf`

(c) Gerar a ferramenta de modelagem novamente

3. Funcionamento anormal da ferramenta de modelagem (2)
    (a) Fechar a ferramenta de modelagem
    (b) Fechar o IDE Eclipse Eugenia
    (c) Iniciar o IDE Eclipse Eugenia
    (d) Iniciar a ferramenta de modelagem
4. Objetos ou Conexões (*links*, referências ou associações) não aparecem na paleta da ferramenta de modelagem
    (a) Certifique-se de que todos os nós e links estão relacionados na classe raiz do arquivo `.emf`
        - A classe raiz é a que começa com a anotação `@gmf.diagram`
5. Erro ao gerar ferramenta de modelagem
    (a) Verificar se há erros de compilação e resolvê-los
        i. Acessar menu "*Window*" > "*Show View*" > "*Problems*"
    (b) Caso não haja erro de compilação, tentar os passos 2 ou 3
6. Modelo gráfico não aparece na ferramenta
    (a) Certifique-se de que todos os projetos gerados pelo Eugenia (*i.e.*, projeto da ferramenta, `.diagram`, `.edit`, `.editor`, `.tests` e `.validation` (se for o caso)) estão abertos
    (b) Se for preciso, gere novamente a ferramenta de modelagem

4.4.5. Uma Ferramenta de Modelagem Gráfica de Cenários de Computação Ubíqua

Essa subseção apresenta uma ferramenta de modelagem gráfica para cenários de computação ubíqua, para permitir a criação de modelos em conformidade com o metamodelo para modelagem de cenários de computação ubíqua, apresentado em [Vieira 2016], apresentado na Figura 4.14.

O editor gráfico foi implementado utilizando o *Eclipse Graphical Modeling Framework* (GMF). O editor gráfico foi incrementado com o uso de linguagens da família *Epsilon* e sua construção foi apoiada pela ferramenta *Eugenia*, que também integra a família *Epsilon*. As tecnologias utilizadas nesta subseção foram apresentadas na Subseções 4.3.2 e 4.3.3. Os códigos-fonte aqui apresentados são apenas trechos.

A Figura 4.15 apresenta a ferramenta de modelagem sendo utilizada para a configuração de um cenário de computação ubíqua. A janela da ferramenta é dividida em duas partes: *(i)* e *(ii)*. Em *(i)*, há a área reservada para a construção do modelo. Em *(ii)*, estão os ícones que representam os conceitos do metamodelo, agrupados em uma barra de ferramentas (ou paleta). A barra de ferramentas é dividida em duas partes: *(a)* ferramentas para criação de elementos, ou seja, instâncias dos tipos definidos no metamodelo; e *(b)* ferramentas para criação de associações entre os elementos.

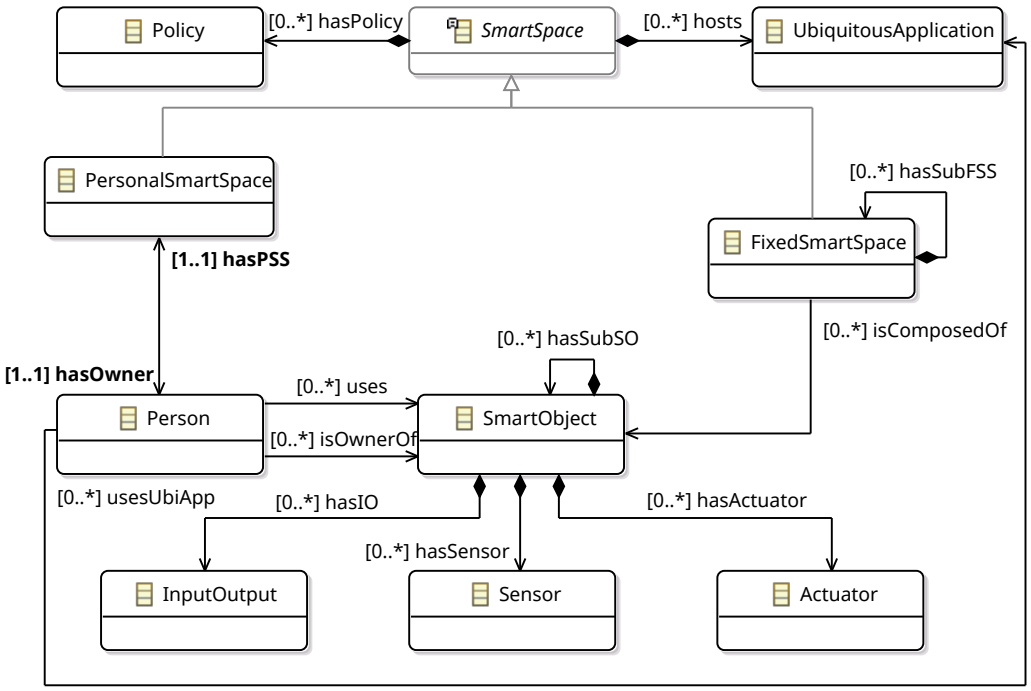


Figura 4.14. Metamodelo proposto para modelagem de cenários compostos por espaços inteligentes pessoais e fixos [Vieira 2016].

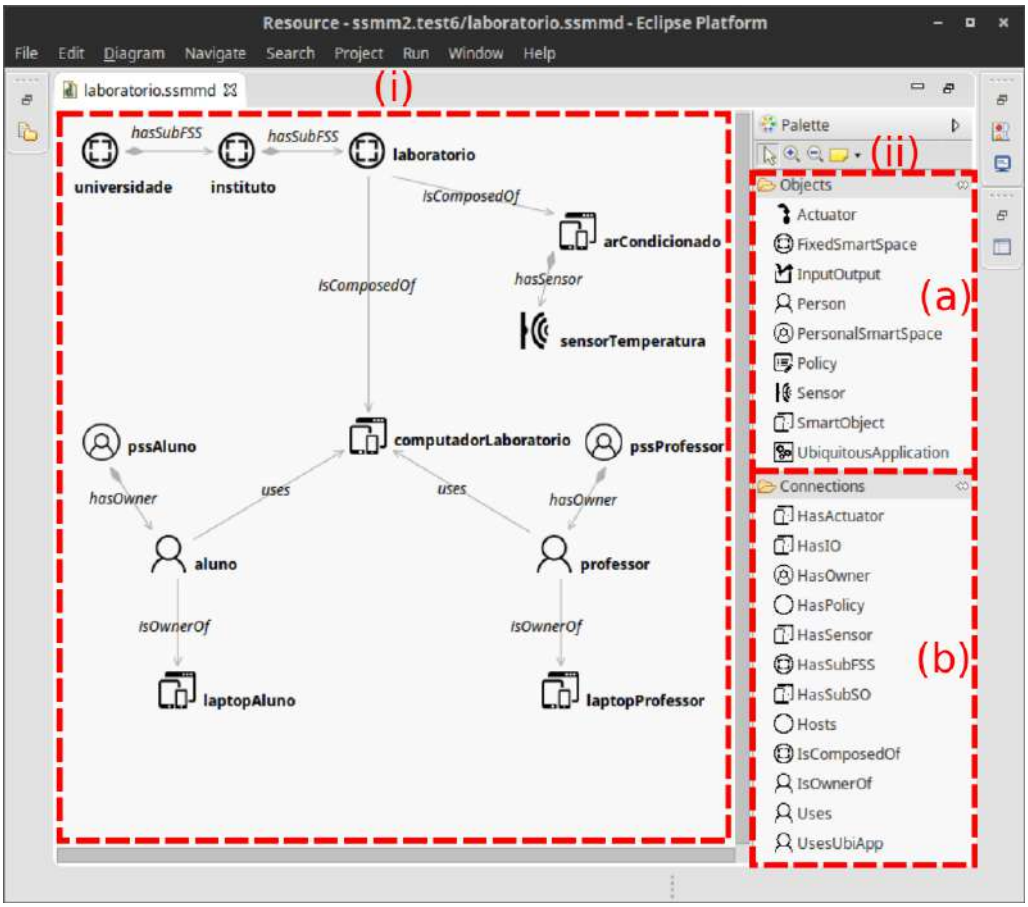


Figura 4.15. Ferramenta de modelagem sendo utilizada para construção de um cenário de computação ubíqua.

Os modelos construídos pela ferramenta de modelagem podem ser salvos em dois arquivos XML com extensões distintas: `.ssmm` e `.ssmmd`. O arquivo com extensão `.ssmm` representa a instância do metamodelo e contém os elementos, seus atributos e as associações entre os elementos; o arquivo com extensão `.ssmmd`, por sua vez, serve para indicar a posição de cada elemento e associação no diagrama gráfico, de maneira que suas posições sejam preservadas ao fechar e abrir novamente o modelo.

A ferramenta de modelagem permite validar o modelo construído para garantir sua conformidade com o seu metamodelo. Caso alguma inconsistência seja encontrada, ela é apontada, permitindo sua correção. Para estender a validação dos modelos, foram definidas regras por meio da linguagem EVL (*Epsilon Validation Language*), permitindo incrementar as regras sintáticas definidas pelo metamodelo com restrições adicionais. A Tabela 4.1 apresenta as regras EVL que foram implementadas.

Tabela 4.1. Regras EVL implementadas na ferramenta gráfica de modelagem.

Regra	**Tipo**	**Descrição**	**Elemento ao qual se aplica**	***Quick fix(es)***
RV1	*Error*	O elemento deve possuir nome definido	Todos	Atribuir nome ao elemento
RV2	*Warning*	O nome do elemento deve iniciar com letra minúscula	Todos	Sugere o nome com inicial minúscula Opção para renomear o elemento
RV3	*Error*	Não pode existir outro elemento do mesmo tipo com mesmo nome	Todos	Opção para renomear o elemento
RV4	*Error*	Não pode haver autorreferenciamento para o elemento	`FixedSmartSpace`, `SmartObject`	Remover a autorreferência

O Código 4.5 apresenta as regras RV1, RV3 e parte da regra RV4: a regra RV1 exige que o elemento tenha um nome; a regra RV3 impede que dois elementos do mesmo tipo tenham nomes iguais; o trecho da regra RV4 apresentado impede o autorreferenciamento a elementos do tipo `FixedSmartSpace`, ou seja, impede que um FSS esteja contido nele próprio. Foram definidos *quick fixes* (indicados no código pelo bloco *fix*) que, quando ativados, realizam a correção do erro encontrado de maneira simplificada. O *quick fix* da regra RV3 solicita a entrada de um novo nome para o elemento duplicado e, então, aplica o novo nome ao elemento. O *quick fix* da regra RV4 apaga o autorreferenciamento para o nó, conforme ilustrado na Figura 4.16.

Código 4.5. Regras EVL: RV1, RV3 e parte da regra RV4.

```
1 //regra RV1
2 constraint HasName {
3     check : self.name.isDefined()
4     message : 'Unnamed ' + self.eClass().name + ' not allowed'
5     fix {
6         title : 'Set a name...'
7         do {
```

```
8               self.name := System.user.prompt('Please enter a name
                    ');
9           }
10      }
11  }
12  //regra RV3
13  constraint CheckUniqueName {
14      guard : not self.~checked.isDefined()
15
16      check {
17          var others := Actuator.all.select(c|c.name = self.name
                and c <> self);
18          if (others.size() > 0) {
19              for (other in others) {
20                  other.~checked := true;
21              }
22              return false;
23          } else {
24              return true;
25          }
26      }
27
28      message : 'Duplicated ' + self.eClass().name + ' name'
29      fix {
30          title : 'Rename...'
31          do {
32              self.name := System.user.prompt('Please enter a new
                    name', self.name);
33          }
34      }
35  }
36  //regra RV4 (parcial)
37  context HasSubFSS {
38      constraint CannotSelfReference {
39          check : self.source.name <> self.target.at(0).name
40          message : 'Cannot make a self reference'
41          fix {
42              title : "Delete self reference"
43              do {
44                  delete self;
45              }
46          }
47      }
48  }
```

Por último, foram feitos ajustes finos na ferramenta de modelagem para melhorar a aparência dos rótulos dos elementos e das associações entre os elementos dos modelos construídos, com o objetivo de facilitar a leitura dos modelos e evitar confusões entre rótulos de associações e rótulos de elementos. Assim sendo, duas regras foram escritas em linguagem EOL (*Epsilon Object Language*) para *(i)* mudar os rótulos das associações para itálico e *(ii)* mudar os rótulos dos elementos para negrito. O Código 4.6 apresenta a regra que muda para itálico os rótulos das associações.

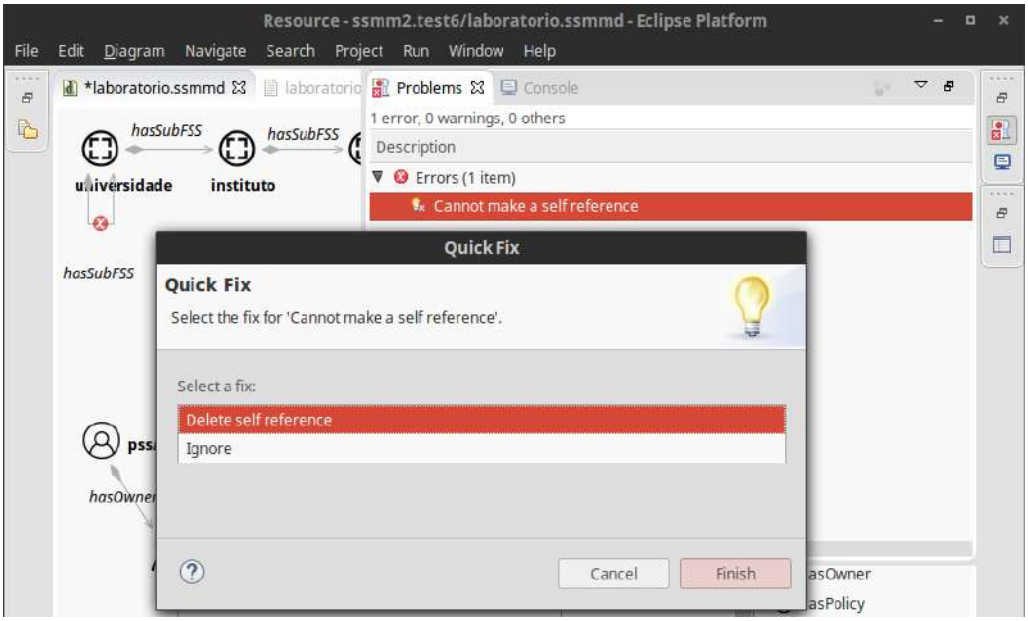


Figura 4.16. Sugestão de correção (*quick fix*) para autorreferenciamento não permitido por meio de regras EVL.

Código 4.6. Regras de transformação escritas em linguagem EOL para estilização da ferramenta de modelagem.

```
1  for (c in GmfGraph!Label.all) {
2    if(c.name = 'HasActuatorLabelLabel' or
3       c.name = 'HasIOLabelLabel' or
4       c.name = 'PersonalSmartSpaceHasOwnerExternalLabel' or
5       c.name = 'HasPolicyLabelLabel' or
6       c.name = 'HasSensorLabelLabel' or
7       c.name = 'HasSubFSSLabelLabel' or
8       c.name = 'HasSubSOLabelLabel' or
9       c.name = 'HostsLabelLabel' or
10      c.name = 'FixedSmartSpaceIsComposedOfExternalLabel' or
11      c.name = 'PersonIsOwnerOfExternalLabel' or
12      c.name = 'PersonUsesExternalLabel' or
13      c.name = 'PersonUsesUbiAppExternalLabel'){
14      c.font = new GmfGraph!BasicFont;
15      c.font.style = GmfGraph!FontStyle#ITALIC;
16   }
17 }
```

4.5. Considerações Finais

Este minicurso introdutório visa, não só introduzir os participantes à área da engenharia dirigida a modelos, como também capacitá-los quanto à construção de um modelo completo, envolvendo desde a concepção do metamodelo até a construção de uma ferramenta de modelagem gráfica com a qual é possível construir modelos derivados de seu metamodelo. Esses modelos poderão, então, ser usados para documentar e manter sistemas de diferentes domínios.

Referências

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer.

Al-Muhtadi, J., Ranganathan, A., Campbell, R., and Mickunas, M. D. (2003). Cerberus: a context-aware security scheme for smart spaces. In *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.(PerCom 2003).*, pages 489–496. IEEE.

Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey . *Computer Networks*, 54(15):2787 – 2805.

Bézivin, J., Jouault, F., and Touzet, D. (2005). Principles, standards and tools for model engineering. In *Engineering of Complex Computer Systems, 2005. ICECCS 2005. Proceedings. 10th IEEE International Conference on*, pages 28–29. IEEE.

Carvalho, S. T. (2013). *Modelagem de Linha de Produto de Software Dinâmica para Aplicações Ubíquas*. PhD thesis, Universidade Federal Fluminense, Niterói, RJ, Brasil.

Carvalho, S. T., Copetti, A., and Loques Filho, O. G. (2011). Sistema de computação ubíqua na assistência domiciliar à saúde. *Journal Of Health Informatics*, 3(2).

Cetina, C., Giner, P., Fons, J., and Pelechano, V. (2009). Autonomic computing through reuse of variability models at runtime: The case of smart homes. *Computer*, 42(10):37–43.

Chagas, J., Ferraz, C., Alves, A. P., and Carvalho, G. (2010). Sensibilidade a contexto na gestão eficiente de energia elétrica. *IN: Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Anais do XXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Gramado: UFRGS*, pages 145–158.

Chen, M., Gonzalez, S., Vasilakos, A., Cao, H., and Leung, V. (2011). Body Area Networks: A Survey. *Mobile Networks and Applications*, 16(2):171–193.

Chiprianov, V., Kermarrec, Y., Rouvrais, S., and Simonin, J. (2014). Extending enterprise architecture modeling languages for domain specificity and collaboration: application to telecommunication service design. *Software & Systems Modeling*, 13(3):963–974.

Coen, M. H., Phillips, B., Warshawsky, N., Weisman, L., Peters, S., and Finin, P. (2000). Meeting the computational needs of intelligent environments: The metaglue system. In *Managing Interactions in Smart Environments*, pages 201–212. Springer.

Cook, D. J. and Das, S. K. (2007). How smart are our environments? An updated look at the state of the art. *Pervasive and mobile computing*, 3(2):53–73.

Corredor, I., Bernardos, A. M., Iglesias, J., and Casar, J. R. (2012). Model-driven methodology for rapid deployment of smart spaces based on resource-oriented architectures. *Sensors*, 12(7):9286–9335.

Crotty, M., Taylor, N., Williams, H., Frank, K., Roussaki, I., and Roddy, M. (2009). A Pervasive Environment Based on Personal Self-improving Smart Spaces. In Gerhäuser, H., Hupp, J., Efstratiou, C., and Heppner, J., editors, *Constructing Ambient Intelligence*, volume 32 of *Communications in Computer and Information Science*, pages 58–62. Springer Berlin Heidelberg.

Daly, C. and Eclipse Foundation, The (2015a). Emfatic Language Reference. "https://www.eclipse.org/epsilon/doc/articles/emfatic/".

Daly, C. and Eclipse Foundation, The (2015b). EuGENia. "https://www.eclipse.org/epsilon/doc/eugenia/".

Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7.

Dolinar, K., Porekar, J., McKitterick, D., Roussaki, I., Kalatzis, N., Liampotis, N., Papaioannou, I., Papadopoulou, E., Burney, S. M., Frank, K., Hayden, P., and Walsh, A. (2008). PERSIST Deliverable D3.1: Detailed Design for Personal Smart Spaces. http://www.ict-persist.eu/?q=content/persist-deliverables-and-publications. [Online; acessado em Abril-2015].

Eclipse Foundation, The (2015a). Customizing a GMF editor generated by EuGENia. "https://www.eclipse.org/epsilon/doc/articles/eugenia-polishing/".

Eclipse Foundation, The (2015b). Epsilon Object Language. "https://www.eclipse.org/epsilon/doc/eol/".

Eclipse Foundation, The (2015c). Epsilon Validation Language. "https://www.eclipse.org/epsilon/doc/evl/".

Eclipse Foundation, The (2015d). EuGENia GMF Tutorial. "https://www.eclipse.org/epsilon/doc/articles/eugenia-gmf-tutorial/".

Eclipse Foundation, The (2015e). Graphical Modeling Framework Documentation. "https://wiki.eclipse.org/Graphical_Modeling_Framework/Documentation/Index".

Erthal, M. S. (2014). Uma proposta para interpretação de contexto em ambientes inteligentes. Master's thesis, Universidade Federal Fluminense, Niterói, RJ, Brasil.

Favre, J.-M. and Nguyen, T. (2005). Towards a megamodel to model software evolution through transformations. *Electronic Notes in Theoretical Computer Science*, 127(3):59–74.

Ferreira Filho, J. B. (2014). *Leveraging model-based product lines for systems engineering*. PhD thesis, Université Rennes 1, Paris, France.

Freitas, L. A., Costa, F. M., Rocha, R. C., and Allen, A. (2014). An architecture for a smart spaces virtual machine. In *Proceedings of the 9th Workshop on Middleware for Next Generation Internet Computing*, page 7. ACM.

Gallacher, S. M., Papadopoulou, E., Taylor, N. K., and Williams, M. H. (2010). Putting the 'Personal' into Personal Smart Spaces. In *Proceedings of Pervasive Personalisation Workshop*, volume 2010, pages 10–17.

Guinard, D., Trifa, V., Pham, T., and Liechti, O. (2009). Towards Physical Mashups in the Web of Things. In *Proceedings of INSS 2009 (IEEE Sixth International Conference on Networked Sensing Systems)*, pages 196–199, Pittsburgh, USA.

Helal, S., Mann, W., El-Zabadani, H., King, J., Kaddoura, Y., and Jansen, E. (2005). The gator tech smart house: A programmable pervasive space. *Computer*, 38(3):50–60.

Honkola, J., Laine, H., Brown, R., and Tyrkko, O. (2010). Smart-M3 information sharing platform. In *ISCC*, pages 1041–1046.

Johanson, B. and Fox, A. (2002). The Event Heap: a coordination infrastructure for interactive workspaces. In *Mobile Computing Systems and Applications, 2002. Proceedings Fourth IEEE Workshop on*, pages 83–93.

Kavanagh, J. and Hall, W. (2008). Grand challenges in computing research 2008. `http://www.ukcrc.org.uk/grand-challenge/`. [Online; acessado em Abril-2015].

Kawsar, F. (2009). A document-based framework for user centric smart object systems. *PhD in Computer Science, Waseda University, Japan*.

Kolovos, D. S., GarcÃa-DomÃnguez, A., Rose, L. M., and Paige, R. F. (2015a). Eugenia: towards disciplined and automated development of gmf-based graphical model editors. *Software & Systems Modeling*, pages 1–27.

Kolovos, D. S., Paige, R. F., Kelly, T., and Polack, F. A. (2006). Requirements for domain-specific languages. In *Proceedings of the First ECOOP Workshop on Domain-Specific Program Development*.

Kolovos, D. S., Rose, L. M., GarcÃa-DomÃnguez, A., and Paige, R. F. (2015b). The Epsilon Book. "`https://www.eclipse.org/epsilon/doc/book/`".

Kortuem, G., Kawsar, F., Fitton, D., and Sundramoorthy, V. (2010). Smart objects as building blocks for the internet of things. *Internet Computing, IEEE*, 14(1):44–51.

Latré, B., Braem, B., Moerman, I., Blondia, C., and Demeester, P. (2011). A survey on wireless body area networks. *Wireless Networks*, 17(1):1–18.

López-Fernández, J. J., Cuadrado, J. S., Guerra, E., and de Lara, J. (2015). Example-driven meta-model development. *Software & Systems Modeling*, 14(4):1323–1347.

Lupiana, D., O'Driscoll, C., and Mtenzi, F. (2009). Taxonomy for ubiquitous computing environments. In *Networked Digital Technologies, 2009. NDT '09. First International Conference on*, pages 469–475.

Melo, P. C. F. (2014). CSVM: Uma Plataforma para CrowdSensing Móvel Dirigida por Modelos em Tempo de Execução. Master's thesis, Instituto de Informática - Universidade Federal de Goiás, Goiânia-GO, Brasil.

Merks, E. and Sugrue, J. (2009). Essential EMF. "`https://dzone.com/refcardz/essential-emf`". "[Online; acessado em Junho-2015]".

Object Management Group (2015). Object Constraint Language (OCL). "`http://www.omg.org/spec/OCL/`".

Ranganathan, A., Chetan, S., Al-Muhtadi, J., Campbell, R. H., and Mickunas, M. D. (2005). Olympus: A high-level programming model for pervasive computing environments. In *Third IEEE International Conference on Pervasive Computing and Communications, 2005. PerCom 2005.*, pages 7–16. IEEE.

Román, M., Hess, C., Cerqueira, R., Campbell, R. H., and Nahrstedt, K. (2002). Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, 1:74–83.

Roriz Junior, M. P. (2013). C3S: Uma Plataforma de Middleware de Compartilhamento de Conteúdo para Espaços Inteligentes. Master's thesis, Instituto de Informática - Universidade Federal de Goiás, Goiânia-GO, Brasil.

Roussaki, I., Kalatzis, N., Liampotis, N., Frank, K., Sykas, E. D., and Anagnostou, M. (2012). Developing context-aware personal smart spaces. In Alencar, P. and Cowan, D., editors, *Handbook of Research on Mobile Software Engineering: Design, Implementation, and Emergent Applications*, chapter 35, pages 659–676. IGI Global, Hershey, PA, USA.

Roussaki, I., Kalatzis, N., Liampotis, N., Kosmides, P., Anagnostou, M., and Sykas, E. (2015). Putting Personal Smart Spaces into Context. In Díaz, V. G., Lovelle, J. M. C., and García-Bustelo, B. C. P., editors, *Handbook of Research on Innovations in Systems and Software Engineering*, chapter 27, pages 710–730. IGI Global, Hershey, PA, USA.

Roussaki, I., Kalatzis, N., Liampotis, N., Papaioannou, I., Pils, C., Crotty, M., AlanWalsh, Frank, K., Whitmore, J., McKitterick, D., Taylor, N., McBurney, S., Papadopoulou, E., Williams, H., Dolinar, K., Porekar, J., Venezia, C., and Bucchiarone, A. (2008). PERSIST Deliverable D2.1: Scenario Description and Requirements Specification. `http://www.ict-persist.eu/?q=content/persist-deliverables-and-publications`. [Online; acessado em Abril-2015].

Sampaio Junior, A. R. (2014). Controle de Microgrids Dirigido por Modelos. Master's thesis, Instituto de Informática - Universidade Federal de Goiás, Goiânia-GO, Brasil.

Schilit, B. N. and Theimer, M. M. (1994). Disseminating active map information to mobile hosts. *Network, IEEE*, 8(5):22–32.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):0025–31.

Seidewitz, E. (2003). What models mean. *IEEE software*, 20(5):26–32.

Shi, Y., Xie, W., Xu, G., Shi, R., Chen, E., Mao, Y., and Liu, F. (2003). The smart classroom: merging technologies for seamless tele-education. *IEEE Pervasive Computing*, 2(2):47–55.

Siegemund, F. (2004). A context-aware communication platform for smart objects. In *Pervasive Computing*, pages 69–86. Springer.

Smirnov, A., Kashevnik, A., Shilov, N., and Teslya, N. (2013). Context-based access control model for smart space. In *Cyber Conflict (CyCon), 2013 5th International Conference on*, pages 1–15. IEEE.

Sousa, J. P. and Garlan, D. (2002). Aura: An Architectural Framework for User Mobility in Ubiquitous Computing Environments. In Bosch, J., Gentleman, M., Hofmeister, C., and Kuusela, J., editors, *Software Architecture*, volume 97 of *IFIP - The International Federation for Information Processing*, pages 29–43. Springer US.

Steinberg, D., Budinsky, F., Merks, E., and Paternostro, M. (2008). *EMF: Eclipse Modeling Framework*. Pearson Education.

Sztajnberg, A., Rodrigues, A. L. B., Bezerra, L. N., Loques, O. G., Copetti, A., and Carvalho, S. T. (2009). Applying context-aware techniques to design remote assisted living applications. *International Journal of Functional Informatics and Personalised Medicine*, 2(4):358–378.

Taylor, N. (2008). Personal eSpace and Personal Smart Spaces. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 156–161.

Taylor, N. (2011). Personal Smart Spaces. In Ferscha, A., editor, *Pervasive Adaptation: The Next Generation Pervasive Computing Research Agenda*, pages 79–80. Institute for Pervasive Computing, Johannes Kepler University Linz, Linz, AUS.

Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. *Sigplan Notices*, 35(6):26–36.

Vieira, M. A. (2016). Modelagem de espaços inteligentes pessoais e espaços inteligentes fixos no contexto de cenários de computação ubíqua. Master's thesis, Universidade Federal de Goiás, Goiânia, Goiás, Brazil.

Völter, M., Stahl, T., Bettin, J., Haase, A., and Helsen, S. (2013). *Model-driven software development: technology, engineering, management*. John Wiley & Sons.

Weiser, M. (1991). The computer for the 21st century. *Scientific american*, 265(3):94–104.

Weiser, M. and Brown, J. S. (1997). The coming age of calm technology. In *Beyond calculation*, pages 75–85. Springer.

Wienands, C. and Golm, M. (2009). Anatomy of a visual domain-specific language project in an industrial context. In *Model Driven Engineering Languages and Systems*, pages 453–467. Springer.

Wyckoff, P., McLaughry, S. W., Lehman, T. J., and Ford, D. A. (1998). T spaces. *IBM Systems Journal*, 37(3):454–474.

Yau, S. S., Gupta, S. K., Karim, F., Ahamed, S. I., Wang, Y., and Wang, B. (2003). Smart classroom: Enhancing collaborative learning using pervasive computing technology. In *ASEE 2003 Annual Conference and Exposition*, pages 13633–13642. sn.

Biografia Resumida dos Autores

Prof. Me. Marcos Alves Vieira
Professor do Instituto Federal de Educação, Ciência e Tecnologia Goiano - Campus Iporá, onde atualmente é coordenador do curso superior de Tecnologia em Análise e Desenvolvimento de Sistemas. É Bacharel em Informática pelo Instituto Federal de Educação, Ciência e Tecnologia de Goiás (IFG) e Mestre em Ciência da Computação pelo Instituto de Informática (INF) da Universidade Federal de Goiás (UFG). Durante seu mestrado, desenvolveu pesquisa com foco em Espaços Inteligentes para Computação Ubíqua e Engenharia Baseada em Modelos. Participante em comunidades de software livre e colaborador em eventos de software livre regionais.

Prof. Dr. Sérgio Teixeira de Carvalho
Professor do Instituto de Informática (INF) da Universidade Federal de Goiás (UFG) desde 2006, atuando no mestrado, na pós-graduação lato sensu e na graduação. Por muitos anos atuou principalmente como coordenador de projetos de software, ocupando posições como Diretor de Sistemas de Informação e Diretor de Suporte Tecnológico. Por mais de dezesseis anos atuou como profissional de T.I. de diversas empresas públicas e privadas. Possui graduação em Ciência da Computação pela Universidade Federal de Goiás (UFG), Mestrado e Doutorado em Computação pela Universidade Federal Fluminense (UFF). Seus principais campos de atuação são Computação Ubíqua/Pervasiva, em especial voltada para aplicações na área de Sistemas de Informação em Saúde, Informática em Saúde, Arquitetura de Software, Linha de Produto de Software Dinâmica e Arquiteturas Adaptativas.

Capítulo 5

Use of Automatic Speech Recognition Systems for Multimedia Applications

Marcos Valadão Gualberto Ferreira[1] and Jairo Francisco de Souza[1]

[1] Universidade Federal de Juiz de Fora. Departamento de Ciência da Computação (DCC/UFJF)

marcos.valadao@ice.ufjf.br, jairo.souza@ufjf.edu.br

Abstract

The need to retrieve information in multimedia content increases the demand for systems that use automatic speech recognition. A speech recognition system enables the computer to interpret audio signals, generating approximate textual transcriptions. These systems are based on probabilistic models that create a robust and correct model for human speech. In this paper we present a speech recognition systems architecture and we detail its basic components: the acoustic model, language model, lexical and decoder. The training process of acoustic and language models is also presented. Finally, we discuss how these systems can be used in some applications.

Resumo

A necessidade de recuperar informações em conteúdo multimídia aumenta a demanda por sistemas que usam o reconhecimento automático de fala. O sistema de reconhecimento de voz permite que o computador interprete sinais de áudio, gerando transcrições textuais aproximadas. Esses sistemas são baseados em modelos probabilísticos que criam um modelo robusto e correto para a fala humana. Neste artigo, é apresentada uma arquitetura de sistemas de reconhecimento de fala e uma descrição de seus componentes básicos: modelo acústico, modelo de linguagem, lexical e decodificador. O processo de treinamento de modelos acústicos e de linguagem também é apresentado. Finalmente, é apresentado como esses sistemas podem ser usados em algumas aplicações.

5.1. Introdução

Um sistema de Processamento de Linguagem Natural (PLN) abrange aspectos da comunicação humana como a fala, palavras, textos e sentenças considerando todo o contexto em que se encontra e se baseando em estruturas de linguagem [Gonzalez and Vera, 2003]. Estes sistemas são utilizados, de maneira geral, para possibilitar que o computador entenda e se comunique em linguagem humana de forma automática. O PLN é baseado em modelagens realistas que sempre necessitam de melhorias por parte das estruturas de comportamento de linguagem.

Existem diversas aplicações na área de PLN. Entre elas está a técnica de reconhecimento automático de fala ASR (do inglês *automatic speech recognition*), que será apresentado neste capítulo.

Tecnologias de ASR permitem que computadores interpretem a fala humana a partir da síntese de sinais de áudio, isto é, transcrever áudio em textos de linguagem natural [Coelho and Souza, 2014]. Este processo geralmente apresenta grandes dificuldades, pois possui limitações quanto ao reconhecimento por questões técnicas, tais como: tamanho do vocabulário, reconhecimento de fala contínua e quanto a estrutura complexa da voz humana, que depende de fatores como: sotaque, entonação, velocidade da voz, estado emocional etc [Gonzalez and Vera, 2003]. Na última década, o interesse nesta área aumentou, principalmente quando técnicas de *Deep Learning* começaram a ser utilizadas com sucesso em ASR [Dahl et al, 2012]. Desde então, diversos sistemas comerciais (i.e. Microsoft[1], IBM[2] e Google[3]) começaram a se popularizar.

Sistemas para reconhecimento de fala são construídos, geralmente, a partir de decodificadores e de modelagem acústica e de linguagem. Os decodificadores desempenham o papel de interpretação do sinal de áudio e de identificação, dentro da modelagem, por algum termo de equivalência. Já os modelos acústicos e de linguagem são responsáveis pelo mapeamento de toda a estrutura linguística [Cuadros, 2007]. É praticamente impossível que um sistema seja perfeito a ponto de abranger as diversas formas que a voz humana pode tomar e de como interpretá-la. Por isto, os sistemas de reconhecimento de voz conhecidos não possuem acurácia total. Porém, os avanços alcançados nos últimos anos tem permitido que essa tecnologia venha a ser utilizada com sucesso em diversos tipos de aplicação.

A questão central envolvendo um sistema para reconhecimento de fala é possuir um modelo acústico e de linguagem que seja satisfatório [Coelho and Souza, 2014]. Estes modelos são treinados utilizando uma base de dados pré-processada e técnicas de descrição probabilística para os termos da linguagem [Neto et al., 2005].

Este capítulo apresenta um resumo dos principais componentes de um sistema de reconhecimento de fala, entre eles modelo acústico, modelo de linguagem, léxico e

[1] https://docs.microsoft.com/pt-br/azure/cognitive-services/Speech/API-Reference-REST/BingVoiceRecognition

[2] https://www.ibm.com/watson/developercloud/speech-to-text.html

[3] https://cloud.google.com/speech/

decodificador. Também serão apresentadas as principais técnicas para treinamento de modelos, ferramentas de apoio e algumas aplicações destes sistemas.

5.2. Arquitetura de sistemas para reconhecimento de fala

Os sistemas ASR atuais são baseados no reconhecimento estatístico de padrões. Por trás desses sistemas podemos identificar minimamente quatro grandes componentes: **modelo acústico**, que mapeia o sinal original que está sendo processado em palavras e sentenças; o **modelo de linguagem**, que é o responsável por caracterizar a língua e condicionar a combinação de palavras, descartando frases agramaticais; o **léxico**, que representa as palavras do dicionário utilizado, com suas respectivas transcrições fonéticas; e o **decodificador**, que procura a melhor sequência de palavras num conjunto de hipóteses possíveis. A Figura 5.1. representa a arquitetura genérica de um sistema de reconhecimento de fala.

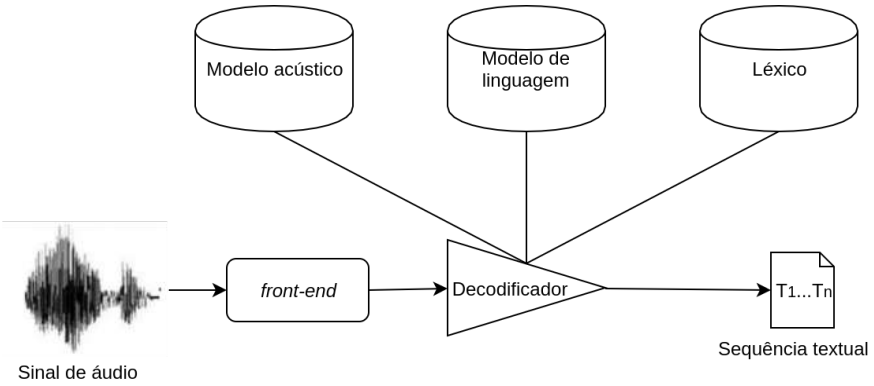


Figura 5.1. Arquitetura de sistemas ASR

Em alguns casos, devido às diferenças de arquiteturas, não se observa uma diferença tão rígida entre um ou mais desses componentes. Contudo, via de regra, as funcionalidades de cada um deles está sempre presente em um sistema de reconhecimento de fala. Nesta seção, vamos discuti-los separadamente.

5.2.1. Modelos Acústicos

Os modelos acústicos geralmente são HMM (*Hidden Markov Model*) [Rabiner and Juang, 1986] de fonemas com 3 estados. Podem ser modelos sem contexto, monofones, ou com contexto: difones (contexto à direita ou à esquerda), trifones (contexto à direita e à esquerda) ou n-fones. No *Hidden Markov Model*, o objetivo é encontrar parâmetros ocultos a partir de parâmetros observados.

Um HMM é definido pelos seguintes parâmetros: número de estados, matriz de probabilidade de função entre estados e, para cada estado, uma função de densidade de probabilidade, que tem como objetivo a caracterização acústica deste estado [da Veiga, 2013]. Por se tratar de um sistema com amplo vocabulário, as palavras modeladas são decompostas em suas respectivas transcrições fonéticas.

Cada fonema é representado por uma HMM que contém três estados emissores e uma topologia simples do tipo esquerda-direita que é comumente utilizada em sistemas ASR

[da Veiga, 2013]. Estados de entrada e saída, não emissores, são acrescentados para auxiliar na junção de modelos. Assim, o estado de saída de um fonema pode se juntar ao estado de entrada de outro fonema criando um HMM composto, como apresentado na Figura 5.2. Assim, é possível que os modelos se juntem formando palavras e estas sejam unificadas formando frases.

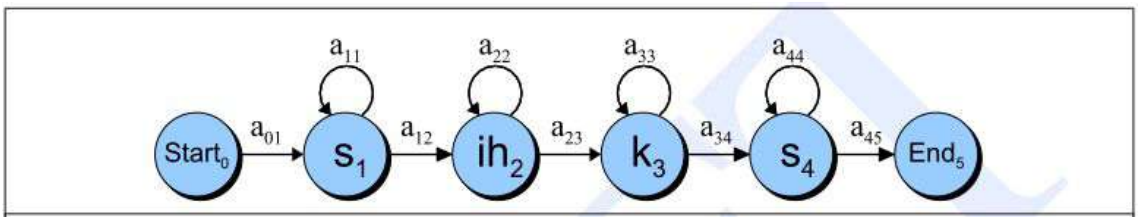


Figura 5.2. Junção de fonemas para representação da palavra "six" ("s ik k s") [Jurafsky, 2009]

O objetivo da modelagem acústica é definir quais as sequências de estados e, com isto, a sequência de fonemas, que possuem uma melhor verossimilhança a um vetor de características acústicas que está sendo processado, definindo assim uma série de probabilidades de palavras e sequências de palavras para o segmento.

Os modelos acústicos geralmente utilizam o HMM para lidar com a variabilidade temporal da fala e os Modelos de Misturas de Gaussianas (GMM - *Gaussian Mixture Model*) na representação da relação entre os estados do HMM e a entrada acústica. Uma abordagem alternativa de representação da relação é o uso de uma rede neural *feed-forward,* a qual recebe vários quadros de coeficientes de entrada e produz probabilidade sobre os estados HMM como saída. Esses modelos baseados em HMM e redes neurais profundas (DNN - *Deep Neural Networks*), mostraram superar o GMM em diversos *benchmarks* para reconhecimento automático de fala [Hinton et al., 2012].

Apesar de todas a vantagens que os GMM's trouxeram para o reconhecimento de fala, eles apresentam uma grave deficiência: são estatisticamente ineficientes para modelar dados que se situam sobre ou próximo de um coletor não-linear no espaço de dados. Isso implica diretamente no entendimento da estrutura que a fala produz. Com isso, acredita-se que outros tipos de modelos, como o de redes neurais, podem possuir resultados melhores para a modelagem acústico se for possível explorar de forma mais efetiva informações em uma grande janela de quadros da estrada acústica. As redes neurais têm a capacidade de extração de informação de dados sobre, ou próximo de um, coletor não-linear muito melhor que o GMM.

A técnica para treinamento de modelos acústicos usando redes neurais faz uso dos novos algoritmos de aprendizagem de máquinas e a capacidade de hardware para a formação de redes neurais profundas que contém muitas camadas de unidades não-lineares escondidas e uma camada de saída grande. Esta camada de saída grande é necessária para acomodar o grande número de estados do HMM que surgem quando cada fonema é modelado a partir das ligações dos modelos HMM's de fonemas que surgem no decorrer do treinamento.

O ajuste dos DNN's no treinamento é realizado em duas etapas. Na primeira etapa são inicializados os detectores de características, uma camada de cada vez, criando uma pilha de modelos, cada um dos quais tem uma camada de variáveis latentes. Na segunda fase, cada modelo da pilha é usado para inicializar uma camada de unidades ocultas em um DNN e então a rede é aperfeiçoada para prever os estados alvos. Estes alvos são obtidos utilizando um sistema-base HMM-GMM para o alinhamento forçado. Esta técnica é compartilhada por grupos de pesquisa da Universidade de Toronto, Microsoft Research, Google and IBM Research [Hinton et al., 2012].

Em linhas gerais, o modelo de treinamento com DNN funciona da seguinte forma:

- Uma janela com alguns milissegundos do sinal de fala é enviada para uma DNN.
- A DNN calcula a probabilidade daquela janela ser um determinado fonema e envia essa informação para o HMM.
- O HMM então "funde" essa informação com aquela que recebeu de outras janelas do sinal de fala para retornar qual o texto mais provável para aquele sinal de fala.

Esse modo de funcionamento está ilustrado na Figura 5.3. O sinal de áudio, indicado na figura com o rótulo *Observation*, está representado pelo seu espectro. Trechos desse espectro são enviados, um de cada vez, para a DNN, que os processa através das M camadas h. A saída da rede é então utilizada como probabilidade de observação (*Observation Probabilities*) pelo HMM. Ao final, o HMM produzirá qual a sequência de fonemas mais provável para aquela dada amostra de fala. Note nessa figura a presença dos pesos W da rede e as probabilidades de transição (*Transition Probabilities*) do HMM. Estes parâmetros precisam ser estimados no treinamento do modelo acústico.

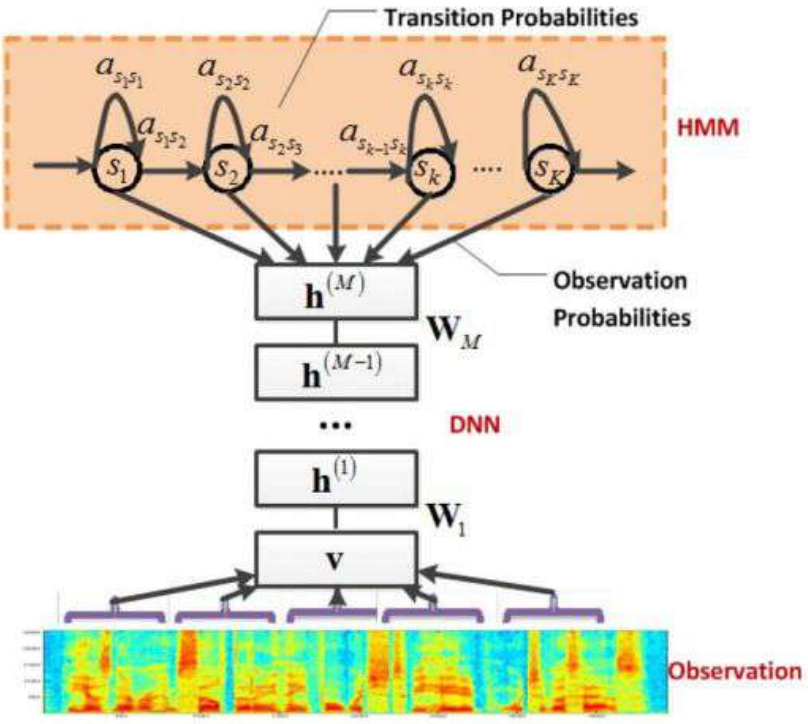


Figura 5.3. DNN aplicada para reconhecimento de fala [Dahl et al., 2012].

5.2.2. Modelos de Linguagem

O modelo de linguagem (LM - *Language Model*) não é necessário para o pleno funcionamento de sistemas ASR. Porém, utilizar somente informações acústicas não é suficiente para o desempenho satisfatório destes sistemas, uma vez que uma palavra poderia ser seguida por qualquer outra sem restrição [da Silva, 2010]. Assim, é consenso na área a utilização de modelos de linguagem para sistemas que lidam com amplo vocabulário, trazendo vantagens para o processo, como a redução do espaço de busca, redução do tempo de reconhecimento e a resolução de ambiguidades acústicas [de Sá Pessoa et al., 1999; da Silva, 2010].

O propósito do LM é prover a probabilidade de ocorrência de uma palavra $P(w_k)$ dada uma sequência de palavras que a antecedem, como é descrito na fórmula abaixo:

$$P(w_k) = P(w_k)P(w_k|w_{k-1})P(w_k|w_{k-1}w_{k-2})...P(w_k|w_{k-1}w_{k-2}...w_1)$$

Onde w_k, é uma palavra da sequência de palavras. Assim, a probabilidade de uma sequência de palavras w_k^1 ocorrer é dado por:

$$P(w_k^1) = P(w_1)\prod_{i=2}^{k} P(w_i|w_1...w_{i-1})$$

Pode-se observar que o cálculo destas probabilidades para grandes vocabulários é algo inviável, pois a probabilidade de ocorrência será calculada em relação a todas as palavras da frase. Para resolver este problema é utilizado n-gramas, onde se assume a probabilidade de uma dada palavra somente até n-1 palavras que a antecedem. Assim, o cálculo de $P(W_k^1)$ é dado por:

$$P(W_k^1) = P(w_1) \prod_{i=2}^{k} P(w_i|w_{i-n+1})$$

Os *n*-gramas são bastante eficientes em línguas onde a ordem das palavras é importante, pois o modelo de linguagem abrange características de sintaxe e semântica e evita a necessidade da criação de regras e de uma gramática formal. Em sistemas ASR é comumente utilizado bigramas (n = 2) e trigramas (n = 3), pois a maioria das palavras possuem forte dependências das duas que a antecedem.

Para avaliação de um modelo de linguagem, utiliza-se a medida de perplexidade. A perplexidade representa o quão indeciso o modelo pode ficar dada uma sequência textual. Quanto menor o valor da perplexidade, melhor é o modelo [Young et al., 2006]. Esta medida é obtida utilizando uma base de teste contendo frases, onde cada frase é avaliada segundo o modelo de linguagem construído. Com isso, um bom modelo de linguagem deve apresentar uma perplexidade alta para uma frase sintaticamente incorreta e uma perplexidade baixa para uma frase sintaticamente correta.

Como o modelo de linguagem é baseado em probabilidades geradas a partir da frequência de palavras, existem diversos métodos para amortizar os efeitos destes números. Isto é feito, geralmente, para diminuir a diferença de probabilidades entre palavras muito frequentes e pouco frequentes, como é demonstrado na Figura 5.4.

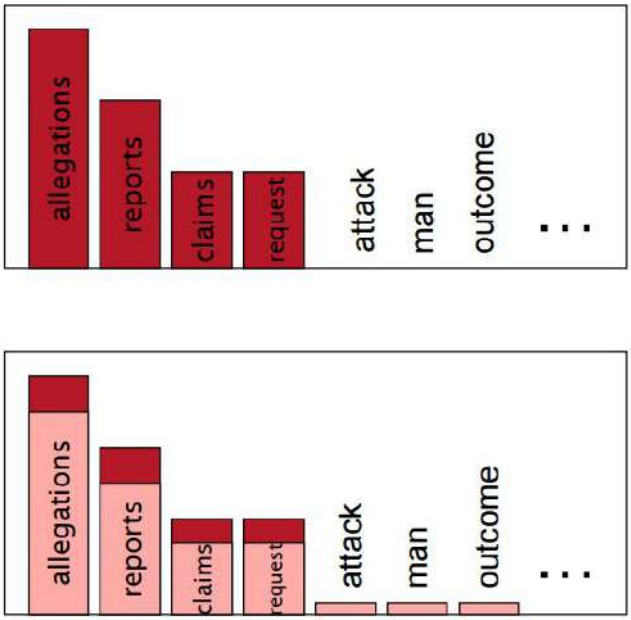


Figura 5.4. Efeito de descontos em modelos de linguagem. No primeiro gráfico é ilustrado as probabilidades sem o uso de métodos de desconto e no segundo com o uso de desconto [Jurafsky, 2017].

Na imagem é exemplificado os efeitos dos métodos de descontos no modelo de linguagem. A diferença entre as palavras com maior probabilidade (*allegations, reports claims* e *request*) e as com menor probabilidade (*attack, man* e *outcome*) diminui com a aplicação do desconto. Com isso, o modelo se torna mais justo para palavras com baixa frequência aumentando a chance de ocorrência das mesmas. Os métodos de descontos mais conhecidos são Add Estimation, Laplace (Add-one), Good-Turing [Katz, 1987], Kneser-Ney [Ney and Essen, 1991] e Witten-Bell [Witten and Bell, 1991].

Esta amortização do modelo de linguagem induz a diferenças no resultado da avaliação do modelo. Os métodos de desconto tanto podem aumentar a perplexidade (piorar o modelo) quanto diminuir a perplexidade (melhorar o modelo) dependendo do contexto de aplicação. Porém, uma prática que geralmente melhora os resultados do modelo de linguagem (diminui a perplexidade) é o aumento da base de dados para treino. Isto é avaliado em [Silva et al., 2004] onde é discutido, dentre outros aspectos, a influência no aumenta da base de texto para treinamento de modelos de linguagem para sistemas de reconhecimento de fala para grandes vocabulários, como é apresentado na Figura 5.5.

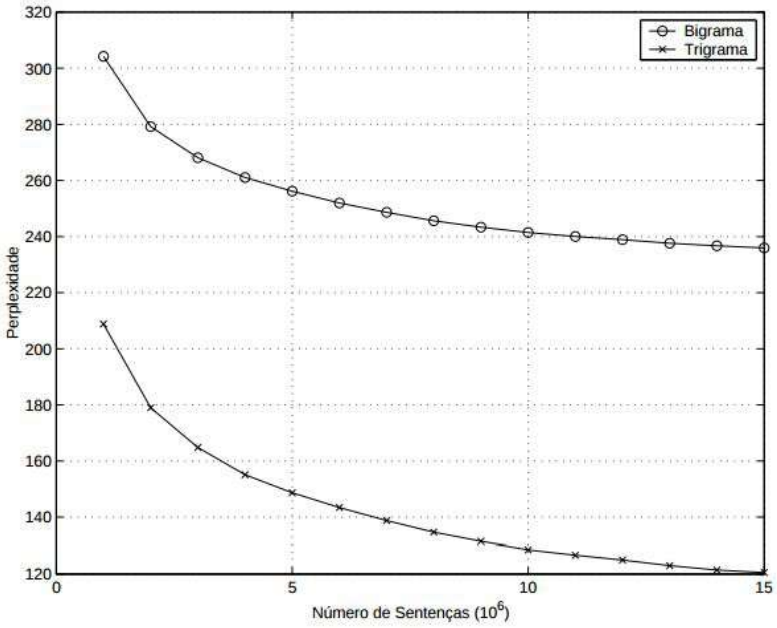


Figura 5.5. Evolução da perplexidade com aumento dos dado para treino, para bigramas e trigramas [Silva et al., 2004]

Pode-se observar a queda significativa que a perplexidade teve a medida que o número de sentenças aumentou. Isto influencia diretamente no sistema de reconhecimento de fala, pois a medida que o modelo de linguagem produz melhores probabilidades, os resultados serão melhores.

5.2.3. Léxico

O léxico é composto por um sequência de palavras e suas respectivas transcrições fonéticas. Esta técnica também é conhecida como conversão grafema fonema, ou seja, conversão de símbolos gráficos utilizados na construção de palavras da língua em unidades sonoras usadas para formar e distinguir palavras. A Tabela 5.1 mostra um exemplo de 5 palavras que compõem um léxico em um sistema de reconhecimento automático de fala.

Tabela 5.1. Componente léxico de um sistema ASR

Palavra	Transcrição fonética
catou	k a t o w
cisne	s i z n y
elevador	e l e v a d o r
maçãs	m a s ɐ̃ s
polímeros	p o l i m e ɾ ʊ s

Este componente é um importante pré-requisito para a construção de sistemas ASR. Porém, a conversão de uma sequência de caracteres em sequência de fones não é uma tarefa trivial. Sabe-se que a transcrição fonética gerada automaticamente apresenta algumas limitações. Uma dessas fraquezas é a dificuldade em tratar as combinações formadas pelo final de uma palavra com início de uma outra. Por exemplo, a palavra

"dois" termina com um "s" e soa como "ss" sempre, exceto quando a palavra falada após o "dois" começar com vogal. Nesse caso, o último fonema de dois se transforma em "z". Há diversos trabalhos que lidam com esses e outros problemas relacionados à transcrição fonética automática, como descrito apresentado em [Goel et. al.,2009].

Existem ainda trabalhos relacionados com a criação do dicionário fonético a as dificuldades da tarefa de conversão de sequência de caracteres em sequência de fones como em [Siravenha et al., 2008], onde são apresentadas as duas técnicas de destaques na literatura para a conversão grafema fonema: *data-driven* e baseadas em regras. O trabalho apresenta, primeiramente, a importância da elaboração de um dicionário fonético de grande vocabulário na construção de um sistema de reconhecimento automático de fala e como este componente, quando bem construído, impacta positivamente na acurácia dos modelos. Com isso, é proposto a contribuição de um algoritmo baseado em regras para a conversão grafema fonema. Os conversores baseados em regras apresentam a vantagem de não fazer uso de alinhamento lexical, visto que não há necessidade de treinamento para gerar as próprias regras. A proposta do trabalho é fornecer ao algoritmo regras fonológicas pré-estabelecidas de acordo com a língua no qual o sistema é baseado. Por fim são apresentados os desafios e resultados da abordagem proposta, bem como os ganhos atingidos em comparação com as abordagens tradicionais. Os melhores resultados foram obtidos em sistemas que utilizaram um dicionário baseado em regras, apresentando a menor taxa de WER (*Word Error Rate*).

Uma ferramenta interessante na construção do léxico é o Multilingual G2P[4], que é uma ferramenta de código aberto e que disponibiliza a criação de um léxico de forma simplificada. O arquivo de entrada para a criação do Léxico contém uma lista de palavras que representam o vocabulário de abrangência do sistema de reconhecimento de fala. Na seção 5.5.2., onde é apresentado o SRILM, é mostrado uma ferramenta simples para geração de vocabulário a partir de uma base de dados contendo frases.

O Multilingual G2P é baseado na ferramenta eSpeak[5] e, para utilização, é necessário a instalação do eSpeak. Em uma máquina Linux (Ex: Ubuntu 16.04), utilize o comando abaixo:

$ sudo apt-get install espeak

Após a instalação da dependência, utilize o arquivo executável, disponibilizado no GitHub do Multilingual G2P, usando o comando abaixo:

$ wget https://raw.githubusercontent.com/jcsilva/multilingual-g2p/master/g2p.sh

Após o término do download, utilize o comando abaixo para gerar o Léxico.

$ ./g2p.sh -w [vocabulario] > lexico.txt

[4] https://github.com/jcsilva/multilingual-g2p
[5] http://espeak.sourceforge.net/

O Multilingual G2P permite criar léxico para diversos idiomas. O idioma padrão é o Português do Brasil, porém existem outros idiomas que devem ser indicados como parâmetro na linha de execução acima. Um exemplo de execução é demonstrado abaixo para criação de um léxico em Inglês (en).

$ ./g2p.sh -w [vocab] -l en

Outros idiomas podem ser encontrados na página do Multilingual G2P no GitHub.

5.2.4. Extração de Parâmetros (*front-end*)

A extração de parâmetros é um processo recorrente em todos os sistemas que envolvem reconhecimento de padrões. Esse processo tem como principal objetivo extrair somente as informações do áudio que serão utilizadas no reconhecimento, de forma que seja um processo objetivo e robusto [da Veiga, 2013]. Assim, a função prioritária deste componente é dividir o sinal que está sendo processado em blocos e de cada um destes blocos derivar uma estimativa suavizada do espectro [Tevah, 2006].

Em sistemas ASR, os parâmetros mais utilizados são coeficientes *mel-cepstrais* (MFCC – *Mel-Frequency Cepstral Coefficients*) [Davis and Mermelstein, 1980]. Algumas outras abordagens de parâmetros conhecidas são: coeficientes cepstrais de predição linear (LPCC – *Linear Predictive Cepstral Coefficients*) e coeficientes dinâmicos (parâmetros delta e delta-delta) que também podem ser usados em conjunto com as demais abordagens.

Um exemplo de aplicação da extração de parâmetros em sistemas ASR é compensar o efeito de longa duração de espectros, causado principalmente pela forma acústica como o áudio foi construído (equipamentos, canais de áudio, distância entre o locutor e o microfone etc). Neste caso é utilizada uma técnica de atualização da média espectral para cada segmento de forma a proteger o sistema contra variações do canal [Tevah, 2006].

5.2.5. Decodificador

O decodificador é o componente dos sistemas ASR responsável por manipular todos os outros componentes descritos anteriormente e gerar a melhor sequência textual aproximada para um dado segmento de áudio de entrada. O decodificador terá a disposição todas as informações de dados observados que foram usados na construção do modelo e, desta forma, poderá decidir qual sequência textual mais adequada para determinada entra acústica.

A Figura 5.6. representa um sistema de reconhecimento de dígitos e ilustra a rede de probabilidades que o decodificador possui a partir dos estados HMM e do Léxico.

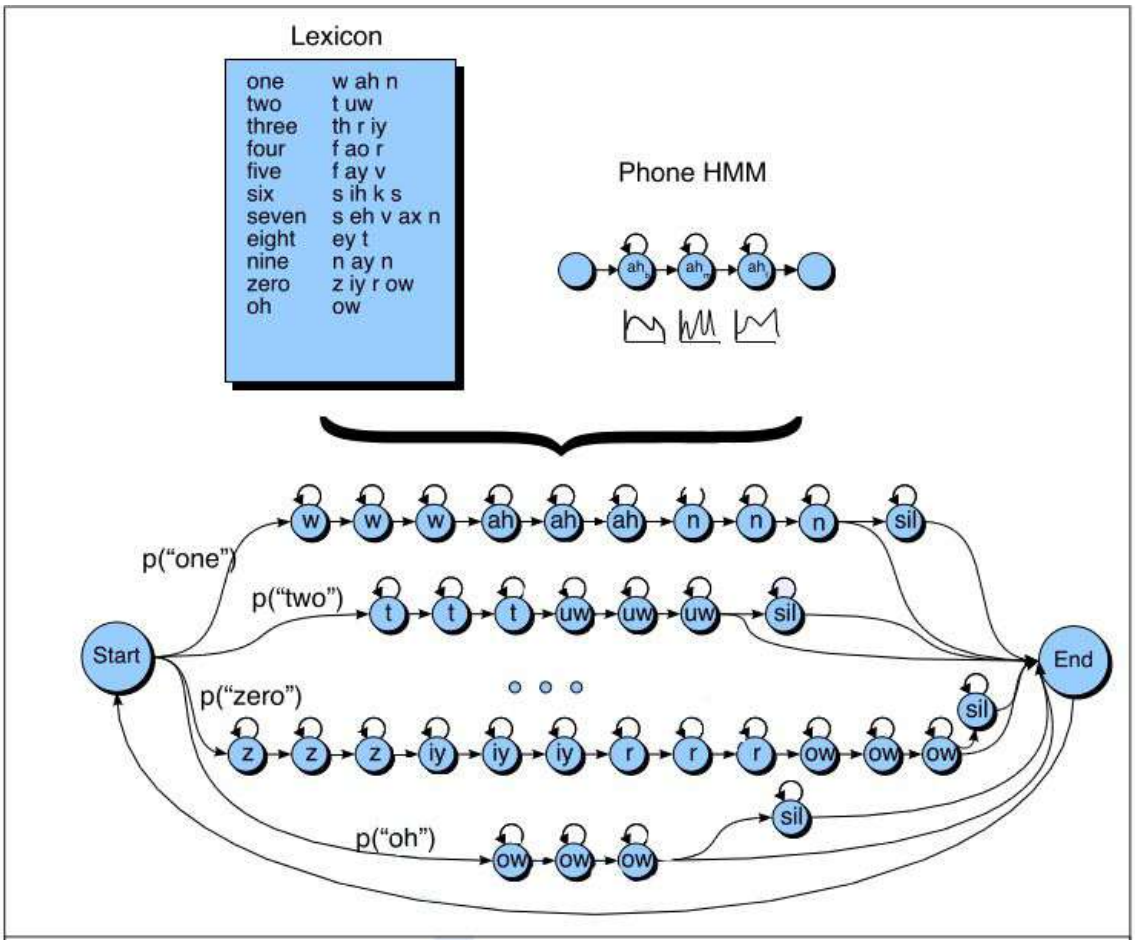


Figura 5.6. Rede de probabilidade de palavras para um sistema de reconhecimento de dígitos. As palavras são representadas por fonemas descritos no léxico [Jurafsky, 2009]

A tarefa do decodificador é obter a melhor sequência de palavras que descreve uma dada sequência de características acústicas (gerada pelo *front-end* a partir de um dado áudio). O processo de decodificação é construído por um rede de palavras geradas pela modelagem do sistema. Esta rede é definida por um conjunto de nós (palavras) conectados por arcos, onde cada arco possui uma probabilidade de ocorrência (transição) [da Silva, 2010]. As palavras são representadas por sequências de modelos acústicos e a escolha das palavras obedece ao modelo de linguagem que define um universo de hipóteses para as sequências de palavras [da Veiga, 2013]. Após encontrar o melhor caminho na rede (aquele cuja probabilidade é mais alta), o decodificador transcreve a sequência de fonemas encontrados em suas respectivas representações grafemáticas no léxico e por fim gera a sequência textual, em linguagem natural, e finaliza o sistema. Com todas as informações do modelo acústico, modelo de linguagem e léxico, forma-se um grafo, com ilustrado na Figura 5.7. Cabe ao algoritmo implementado no decodificador percorrer esse grafo em busca do melhor caminho para um dado sinal de fala.

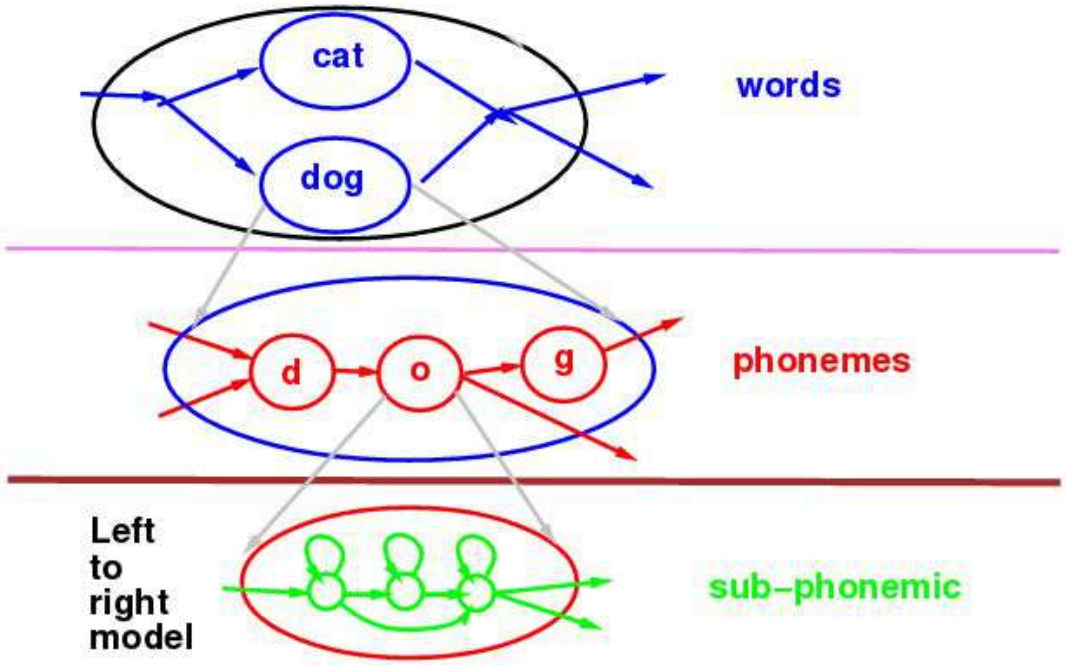


Figura 5.7. O decoder executa um algoritmo de busca em um grafo que agrega conhecimentos do modelo acústico, do modelo de língua e do léxico [Bengio, 1999].

5.3. Avaliação de sistemas de reconhecimento de fala

Os sistemas de reconhecimento de fala são comumente avaliados segundo a taxa de erro de palavras (WER - *Word Error Rate*). Esta métrica representa o quanto a transcrição gerada pelo sistema de reconhecimento de fala (hipótese) difere da transcrição original (referência) [Jurafsky, 2009].

A WER é calculada a partir da definição de quais palavras da hipótese estão corretas, quais foram inseridas incorretamente, quais foram excluídas e quais foram substituídas em comparação com a referência. A WER é calculada pela seguinte fórmula:

$$WER = 100 \times \frac{Inserções + Substituições + Deleções}{Total\ de\ Palavras\ na\ Referência}$$

Outra métrica, menos utilizada, mas também importante é a Taxa de Erro de Frases (SER - *Sentence Error Rate)*, que representa quantas frases possuem pelo menos um erro.

$$SER = 100 \times \frac{Qtd\ de\ sentenças\ com\ pelo\ menos\ um\ erro}{Qtd\ total\ de\ sentenças}$$

Existem ferramentas próprias para o cálculo da WER, como por exemplo a ferramenta Sclite[6], que é um programa para avaliar a saída de sistemas de reconhecimento de fala. Este programa recebe como entrada dois arquivos, um contendo a hipótese e o outro a

[6] http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm#sclite_name_0

referência. A Figura 5.8 apresenta um exemplo de avaliação através da ferramenta Sclite

```
id: (bit_bytes-1215)
Scores: (#C #S #D #I) 8 3 2 0
REF: conversão de caracter para BITS ESSA  primeira parte QUE é UMA parte padrão
HYP: conversão de caracter para BIPS NESSA primeira parte *** ** A   parte padrão
Eval:                             S    S                   D   D  S
```

Figura 5.8. Alinhamento entre referência (REF) e hipótese (HYP) para avaliação de sistemas de reconhecimento de fala. Nesse caso, foram 8 acertos de palavras, 3 substituições, 2 deleções e 0 inserções.

Para este caso, o cálculo da taxa de erro de palavra seria:

$$WER = 100 \times \frac{3+2+0}{13} = 38,46\%$$

O padrão para os arquivos de hipóteses e referências é [ID + "\t" + texto]. É possível enviar também um repositório de arquivos de hipótese e referência. Desta forma, é possível enviar diversos segmentos de transcrição para avaliar a WER. O Sclite também disponibiliza outros arquivos com alguns dados sobre a avaliação feita. Um deles apresenta um resumo geral de todos os arquivos que foram enviados e a avaliação final do sistema. A Figura 5.9. representa um exemplo deste arquivo.

SPKR	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err
bit_bytes	99	2633	61.8	25.1	13.1	4.9	43.1	100.0
complexidade_de_algoritmos	63	1768	67.6	23.5	8.9	7.0	39.4	100.0
dnd_e_redes_p2p	112	3139	56.7	28.3	15.0	4.6	47.9	100.0
espacos_vetoriais	42	1399	50.7	30.8	18.5	3.3	52.6	100.0
introducao_a_computacao_grafica	48	1484	56.3	26.5	17.3	3.2	47.0	100.0
introducao_a_engenharia_sf	21	798	57.6	28.6	13.8	2.3	44.6	100.0
introducao_a_linguagem_html	21	654	52.3	27.8	19.9	2.8	50.5	100.0
introducao_ao_computador	21	569	75.0	15.8	9.1	4.4	29.3	100.0
introducao_a_redes_de_computadores	28	773	63.9	24.3	11.8	3.9	40.0	100.0
processo_agil	47	1454	41.7	30.8	27.4	1.0	59.2	100.0
subsistemas_de_memoria	21	804	47.5	39.8	12.7	5.1	57.6	100.0
Sum/Avg	523	15475	57.4	27.4	15.2	3.9	46.5	100.0
Mean	48.5	1289.6	57.4+	27.4+	15.2+	3.9+	46.5+	100.0
S.D.	32.1	829.5	9.4+	5.9+	5.4+	1.6+	8.6+	0.0
Median	42	1399	56.7+	27.8+	13.8+	3.9+	47.0+	100.0

Figura 5.9. Porcentagens fornecidas para avaliação de saídas do sistemas de reconhecimento de fala.

Na Figura 5.9, a primeira coluna representa cada arquivo que foi enviado para o Sclite avaliar a transcrição. A segunda coluna representa o número de sentenças, a terceira o número de palavras, a quarta a média de acerto em cada arquivo, a quinta a média substituições depois deleções depois inserções e, por fim, a WER ("Err") por arquivo e a SER ("S.Err"). Nas duas últimas linhas temos o desvio padrão da média e a média, respectivamente. Portanto, o valor final de WER dessa amostra de dados é de 47% e o valor de SER é de 100%.

Um outro arquivo do Scilite disponibiliza os principais erros que ocorrem na amostra de dados, como entrada para inserção, remoção e substituição. A Figura 5.10. ilustra uma parte deste arquivo.

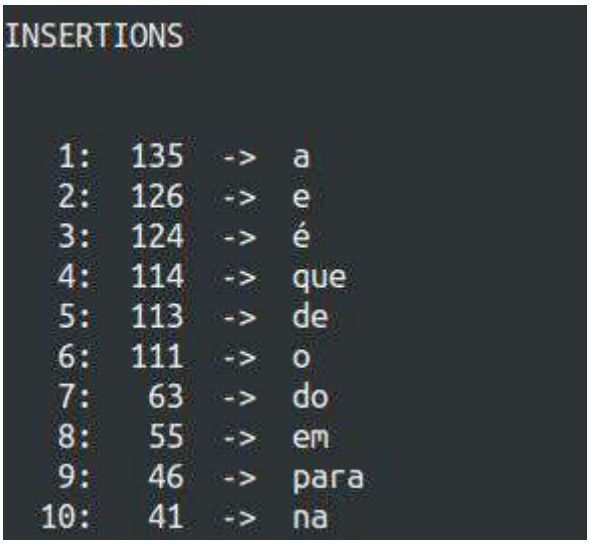


Figura 5.10. Principais erros de inserção em uma dada amostra de dados de hipóteses.

Assim, o principal erro de inserção foi "a" com 135 ocorrências, seguido de "e" com 126 ocorrências e assim por diante. Estes dados também são fornecidos para os erros de deleções e substituições.

Com estes dados é fácil avaliar um sistema de reconhecimento de fala, extrair informações de principais erros e adaptar o sistema para diminuí-los.

5.3. Treinamento de modelos

O principal fator de relevância na criação de sistemas de reconhecimento automático de fala é a elaboração de uma modelagem robusta e que seja adequada para o contexto de aplicação. Para a elaboração dos modelos acústico e de linguagem, é utilizada a técnica de treinamento supervisionado. O treinamento supervisionado constrói a modelagem a partir da observação de dados que possua as características que se deseja obter.

O treinamento do modelo acústico faz uso de uma base de áudio com texto associado. Esta base deve estar bem alinhada, o que quer dizer que áudio e transcrição devem ser bem segmentados. Já o treinamento do modelo de linguagem faz uso de uma base de textos (frases) que estejam sintaticamente corretos.

Para o treinamento de modelos de linguagem, algumas ferramentas implementam os principais algoritmos presentes na literatura. Entre elas, podemos citar o CMU-SLM [Rosenfeld and Clarkson, 1997] e o SRILM [Stolcke, 2002] como poderosas ferramentas para construção de modelos de linguagem.

No treinamento de modelos acústicos, os parâmetros do modelo são ajustados de forma que, para os áudios da base acústica, a saída gerada pelo sistema de reconhecimento de fala seja o mais próximo possível das transcrições originais. Esses ajustes são feitos por algoritmos (como Baum-Welch e Viterbi) que estão implementados em diversas ferramentas de uso livre, como HTK[7], RASR[8] e Kaldi[9].

O Kaldi é um dos *toolkits* para treinamento de modelo acústico mais usados pela comunidade. A ferramenta está sendo constantemente atualizada e contém os algoritmos do estado da arte. Além disso, o Kaldi divulga os procedimentos para o treinamento de sistemas de reconhecimento de fala feitas por grandes grupos de pesquisa ao redor do mundo. Dessa forma, estes procedimentos permitem que usuários inexperientes possam se orientar nos primeiros passos para criar modelos mais simples.

O pacote de ferramentas disponibilizado pelo Kaldi é um instrumento na elaboração de elementos básicos para um sistema de reconhecimento de fala. O *toolkit* possui ferramentas para treinamento de modelos utilizando os mais diversos métodos, incluindo o uso de Redes Neurais [Miao, 2014]. Este também possui métodos para alinhamento de texto e áudio [Ochshorn and Hawkins, 2017], ferramentas para criação de decodificadores e métodos para processamento de áudio.

5.3.1. Treinamento de modelos acústicos

O treinamento de modelos acústicos tem como objetivo a geração de modelos para sistemas ASR influenciando em um acurácia satisfatória destes. Para o treinamento de modelos acústicos HMM é necessário o fornecimento das seguintes informações: conjunto de arquivos de áudio bem segmentados; conjunto de arquivos de texto com as respectivas transcrições originais dos arquivos de áudio; dicionário de abrangência do modelo e respectivas transcrições fonéticas.

A partir disto, os passos para o treinamento de modelos acústicos são:

1. Extração de parâmetros dos arquivos separados para treino: nesta etapa são geradas características acústicas importantes para o treinamento
2. Inicialização dos modelos de fonemas: neste ponto, utiliza-se da técnica *Flat Start* para segmentação automática da base. Esta técnica é dividida em três etapas: a primeira é desconsiderar a pausa existente entre as palavras, gerando uma primeira estimativa dos modelos de fonemas. O segundo é a criação de um modelo de pausa a partir do modelo de silêncio de início e fim de frase (justificando um base de dados bem segmentada, ou seja, respeitando início e fim de frase), após isso re-estimam-se os modelos. O terceiro e último é o realinhamento da base de treino em função dos modelos e estes são re-estimados novamente [Tevah, 2006; Young et al., 2006].

[7] http://htk.eng.cam.ac.uk/
[8] https://www-i6.informatik.rwth-aachen.de/rwth-asr/
[9] http://kaldi-asr.org/

3. Conversão do modelo de fonemas para trifones: os arquivos de texto são novamente transcritos agora com trifones, levando-se ou não em conta, os trifones entre-palavras. O fones centrais dos trifones referenciam os fonemas treinados no passo anterior [da Silva, 2010].
4. Compartilhamento dos estados a partir de árvores de decisão: a lista de todos os trifones necessários ao modelo acústico é criada e no final é gerado um arquivo mapeando modelos que compartilham distribuições [Tevah, 2006; Young et al., 2006].

A Figura 5.11. apresenta, de forma simplificada, como é estruturada as características extraídas da base de treinamento e o resultado do treinamento do modelo acústico.

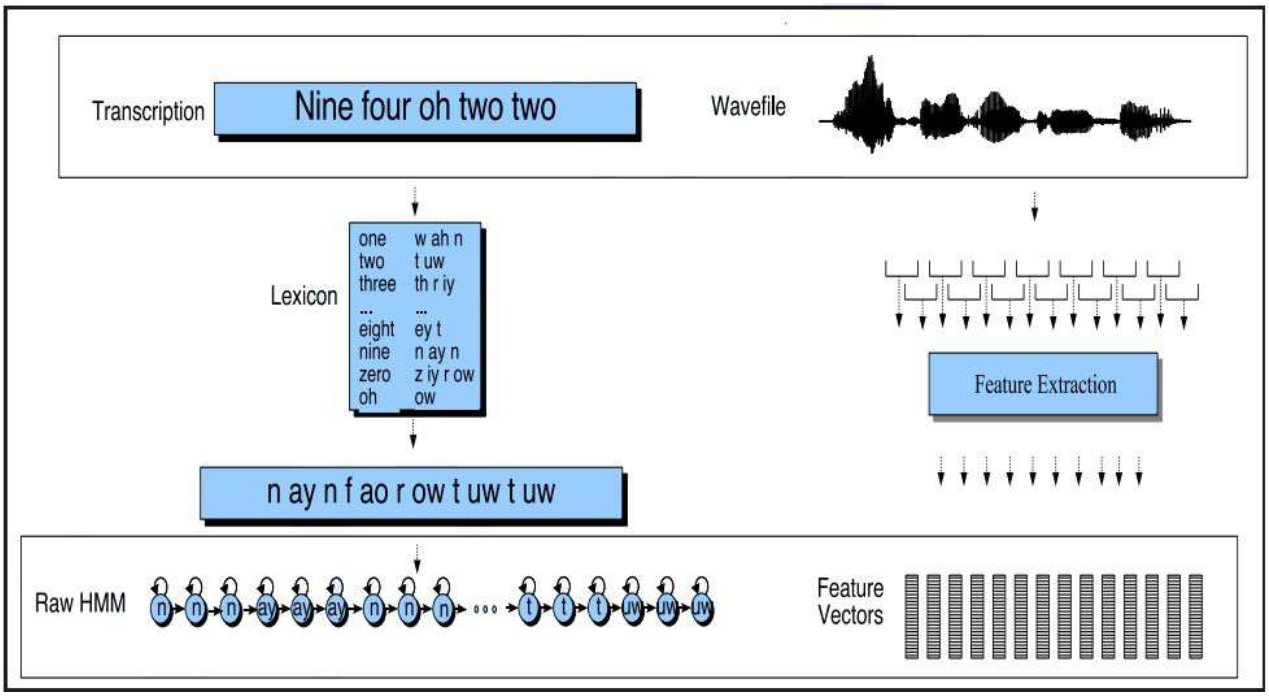


Figura 5.11. Entrada para o algoritmo de treinamento contando com, arquivo de áudio (Wavefile) e respectiva transcrição associada. O arquivo de áudio é processado para extração de características acústicas enquanto a transcrição passa pelo processo de conversão grafemática através do Léxico. Por fim é gerado um vetor de característica acústicas e uma rede de estados HMM [Jurafsky, 2009].

5.3.2. Treinamento de modelos de linguagem

O treinamento de modelos de linguagem tem como objetivo a geração de modelos para sistemas ASR que auxiliarão no reconhecimento de fala e na construção gramatical dos resultados. Para o treinamento do modelo de linguagem são requeridos o dicionário fonético e os arquivos de texto de onde serão extraídas as frequências de palavras e o relacionamento entre elas.

Algumas alterações são necessárias nos arquivos de texto para que o treinamento seja satisfatório. É de fundamental importância que os arquivos de texto tenha marcadores de início e final de frase (<s>, </s>) e que as frases estejam normalizadas (números e siglas), livres de pontuações(. , ! ? … etc) e caracteres especiais ($ % # @ etc) [Young et al., 2006].

A seguir serão descritos os passos para a criação de modelos de linguagem do tipo unigrama, bigrama e trigrama.

1. Geração da frequência das palavras existentes nos arquivos de texto e do relacionamento entre cada palavra com até duas anteriores, incluindo início e final de frase [Tevah, 2006; Young et al., 2006].
2. Identificação de palavras fora do vocabulário (OOV - *Out of Vocabulary*) que são excluídas do treinamento.
3. Treinamento dos n-gramas: Computação dos modelos unigrama, bigrama e trigrama incluindo probabilidades de sequências de palavras e fatores de escalonamento.
4. Medição da perplexidade dos modelos n-gramas gerados.

5.4. Bases de dados para treinamento

Para obter sistemas ASR que sejam robustos e com acurácia satisfatória é necessário uma base de dados bem estruturada e de grande porte para o treinamento de modelos acústicos e de linguagem.

O termo *corpus* (plural *corpora*) de voz será utilizado neste capítulo para representar a base de dados de áudio com suas respectivas transcrições textuais, utilizadas no treinamento de modelos acústicos, e o *corpus* de texto para representar a base de dados de sentenças (frases) estruturadas, utilizadas para treinar o modelo de linguagem.

Uma grande dificuldade no treinamento de modelos acústicos e de linguagem é a obtenção destes *corpora* de grande porte e gratuitos [Silva et al., 2005]. Isto é uma carência na área que difere de outras línguas como o inglês. Além disso, é importante que estes *corpora* sejam especializados, ou seja, que áudios e textos contendo informações de diferentes áreas de conhecimento sejam utilizados no treinamento. Isto para que o sistema não seja "viciado" em termos de uma única área (a não ser que este seja o foco) ou de nenhuma.

A dificuldade de possuir o *corpus* de voz e texto adequados é sem dúvida um dos principais problemas encontrados pelos pesquisadores da área.

Sistema típicos de reconhecimento de fala são treinados usando centenas de horas de dados de treinamento acústico cuidadosamente transcritos, como é dito em [Evermann et al., 2005]. Este trabalho descreve as dificuldades no reconhecimento automático de fala para sistemas CTS (*Conversation Telephone Speech*) e como o aumento da base de dados em milhares de horas melhora a acurácia dos modelos acústicos e de linguagem. Mais especificamente o trabalho compara dois sistemas de transcrição treinados com as mesmas técnicas e quantidades de dados diferentes, um com 360 e o outro com 2000 horas de conversas telefônicas e descreve quais os métodos de limpeza nos dados de áudio e de transcrições associados. Foram aplicados regras para normalizar o texto, corrigir ortografia e descartar palavras que foram ditas somente uma vez. Além disso foi mapeado, na base de áudio, blocos de silêncio e falas masculinas e femininas. O trabalho conclui apresentando as melhorias que foram adquiridas no incremento da base e os valores de ganho na acurácia e taxa de erro de palavras (WER).

Para o Português do Brasil existe uma dificuldade grande em possuir uma base extensa e gratuita. Visando contornar este problema, [Wessel and Ney, 2005] expõem um estudo da influência da quantidade de dados transcritos, inicialmente disponíveis, no desempenho dos modelos acústicos. Neste trabalho é proposto um método diferente para o treinamento de modelos com poucos dados iniciais. Visto que existe muita quantidade de dados acústicos e poucos desses estão associados à transcrições, o trabalho utiliza-se de poucos dados transcritos manualmente para treinar um sistema de base que será usado para reconhecer outros dados acústicos e assim utilizá-los também no treinamento. Para validação da proposta é feita uma marcação de confiança nas palavras transcritas, ou seja, cada palavra transcrita pelo sistema base é associada a uma medida de confiança, identificando que somente palavras acima de um limiar serão utilizadas no treinamento. Nessa abordagem, primeiramente, são estimados os parâmetros de treinamento de um modelo acústico com pequenas quantidades de fala transcritas manualmente. Este modelo acústico é então usado para reconhecer *corpora* de voz grandes que não possuem transcrições associadas e gerar um gráfico de palavras, que é computado a fim de gerar as medidas de confiança. Assim, um novo *corpus* de voz é criado unindo aquele utilizado no primeiro treinamento e o novo gerado. Com isso, o mesmo procedimento padrão de treinamento é executado com o *corpus*, combinado às medidas de confiança geradas. É ressaltado ainda que este processo gera melhorias nas medidas de confiança conforme novos modelos vão surgindo. Ainda, no trabalho, é apresentado os ganhos obtidos na acurácia dos modelos e na taxa de erro de palavras (WER - *Word Error Rate*) a partir dos experimentos realizados.

Existem ainda outros trabalhos que abordam o problema de treinar modelos com poucos recursos. Em [Riccardi and Hakkani-Tür, 2003] é apresentada também uma técnica para transcrever e gerar uma medida de confiança para as transcrições. Os segmentos que não possuem transcrições com boa confiança são transcritos manualmente construindo assim uma base de dados com qualidade melhor. Neste trabalho é descrito um método para combinar aprendizagem ativa e não supervisionada para um sistema de reconhecimento de fala. O objetivo é minimizar a supervisão humana para o treinamento de modelos acústicos e de linguagem e maximizar o desempenho com os dados de áudio com transcrições e sem transcrições associadas. A aprendizagem ativa visa diminuir o número de dados de áudio a serem transcritos manualmente processando-os, e selecionando os mais precisos com relação a uma função de custo. Para a aprendizagem não supervisionada foi utilizada as transcrições automáticas e uma medida de confiança por palavra. O trabalho é concluído apresentando os ganhos e benefícios da combinação de aprendizagem ativa e não supervisionada.

Há também trabalhos para o aproveitamento de mídias legendadas como em [Chan e Woodland, 2004], onde é aproveitada a disponibilidade de legendas em noticiários para treinar modelos acústicos e gerar frases foneticamente balanceadas para o modelo de linguagem. Neste trabalho é apresentado os problemas em se utilizar legendas como base para treino. O principal deles é que, as transcrições geralmente são oriundas de *closed caption* e legendas de comerciais, que são construídas manualmente e são parcialmente corretas. Embora estes transcritos contenham uma série de erros e não possam ser usados diretamente como dados para treino, é proposto no trabalho o usos destes dados como fonte de supervisão para o treino de modelos acústicos, técnica

conhecida como treinamento levemente supervisionado. Na abordagem proposta do treinamento levemente supervisionado, foi utilizado modelos de linguagem tendenciosos para as transcrições de *closed caption* que foram usados no processamento da base de áudio. Com isso, todas as transcrições adquiridas foram utilizadas para treinamento com a técnica de máxima verossimilhança ou para fornecer as transcrições corretas no treinamento discriminativo. Os resultados apresentados mostram que é possível a diminuição da taxa de erro de palavras utilizando esta abordagem, e que a construção de uma base de treinamento com este tipo de mídia é viável para um sistema de reconhecimento dentro desse contexto.

Outro trabalho relacionado a mídias legendas é apresentado em [Panayotov et al., 2015], onde é aproveitado o conteúdo de audiobooks para alimentar a base de dados de treino. Neste trabalho é proposto a criação de uma base de treino para modelos de sistemas de reconhecimento de fala, utilizando conteúdo de audiobooks em inglês disponibilizados gratuitamente pelo projeto LibriVox[10]. A grande questão de se utilizar audiobooks para o treinamento de modelos, é o alinhamento correto do áudio com o texto escrito. Os processos de treinamento, geralmente, requerem que os dados venham segmentados até algumas dezenas de segundos e contenham o texto relacionado ao segmento. A abordagem proposta no trabalho é composta de dois estágios. No primeiro estágio é utilizado um algoritmo de alinhamento para encontrar a melhor região entre o áudio reconhecido e o texto do capítulo. A partir disso é tomada a maior região de similaridade, que geralmente representa um capítulo inteiro, e descartado o resto. Nessa região de similaridade é destacadas palavras que, a partir do reconhecimento, tenham uma alta confiança. Depois disso o áudio é dividido em segmentos de algumas dezenas de segundos utilizando um algoritmo de programação dinâmica. Esta divisão é feita a partir do reconhecimento de regiões de silêncio. Assim é gerado um texto candidato para um segmento de áudio. Na segundo etapa é filtrado os segmentos onde o texto candidato tem uma alta probabilidade de ser impreciso. Também, é criado um gráfico de decodificação para cada segmento. Nesta etapa é rejeitado qualquer segmento que tenha desvio da transcrição original. Isto é feito utilizando técnicas para comparar palavras com blocos de áudio de fala e identificar variações bruscas dos fones de cada palavra e do sinal processado. Além disso foi filtrado segmentos com áudios barulhentos (ruído). O trabalho conclui demonstrado o resultado superior do sistema construído a partir de audiobooks em comparação a outro sistema treinado com bases de dados tradicionais, transcritos manualmente, porém de menor tamanho.

Existem ainda, bases gratuitas que foram construídas por trabalhos acadêmicos, como no desenvolvido pelo Laboratório de Processamento de Sinais (LaPS[11]) da Universidade Federal do Pará, onde foi construído um *corpus* de voz de aproximadamente 11 horas de áudio e um *corpus* de texto com 120 mil frases que são disponibilizados gratuitamente. Outra base disponibilizada gratuitamente é o CETENFolha, que consiste em textos extraídos do jornal Folha de São Paulo e disponibilizados gratuitamente em formato adequado para processamento de texto. A base é mantida pelo projeto Linguateca[12] e

[10] https://librivox.org
[11] http://www.laps.ufpa.br/falabrasil/descricao.php
[12] http://www.linguateca.pt/ACDC/

contém quase 2 milhões de frases em formato ideal para o processamento de modelos de linguagem.

Além de possuir *corpus* que sejam suficientemente grandes, outras questões são levantadas sobre este fator de vital importância na construção de sistemas ASR. É necessário que os textos estejam normalizados, ou seja, dígitos e siglas sejam traduzidos para suas respectivas representações textuais, também é de vital importância realizar uma varredura palavra por palavra para validação ortográfica (são utilizados corretores ortográficos para esta tarefa) [Tevah, 2006] e que as frases sejam foneticamente balanceadas (distribuição fonética similar à encontrada na fala) [Ynoguti, 1999]. Isto deve ser respeitado pois o modelo de linguagem precisa possuir um padrão de escrita. Por exemplo, se no *corpus* de texto usado no treinamento existir tanto a *string* "2" quanto a *string* "dois", a probabilidade será dividida entre as duas *strings* sendo que elas possuem o mesmo significado ("dois" é a representação textual do dígito 2). Outro exemplo clássico é com relação erros de ortografia (ou reformas na escrita das palavras, como aconteceu no Brasil, em 2009, com o Novo Acordo Ortográfico). Se no *corpus* de texto possuir as *strings* "ideia" e "idéia", a probabilidade também será dividida entre estas palavras, gerando contextos diferentes. Além de dígitos e erros de ortografia, é importante observar outros fatores no *corpus* de texto: devem ser removidos caracteres especiais e de pontuação; conversão de número romanos em representação textual; conversão de números ordinais em representação textual; conversão de abreviações e representação textual completa; conversão de valores monetários e de temperatura; demarcação de início e fim de frase (<s> e </s> respectivamente); capitalização de texto em minúsculo. A Figura 5.12. é um exemplo de uma parte de arquivo, ideal para o modelo de linguagem, pertencente ao *corpus* de texto.

```
<s> o clima na minha cidade é chuvoso chove a maior parte do tempo e pouco sol </s>
<s> bom dia hoje é dia vinte e sete de janeiro de dois mil e quartoze e o sol já está predominante </s>
<s> o preço do arroz está três vezes mais caro que o normal chegando a dezoito reais e treze centavos </s>
<s> a ideia principal do jogo é que ele possa simular a realidade </s>
<s> maria foi aprovada em décimo lugar no curso de medicina na universidade de são paulo </s>
<s> a temperatura máxima prevista para hoje é de trinta e seis graus <s>
```

Figura 5.12. Arquivo para treinamento de modelos de linguagem normalizados.

Ainda, para a *corpus* de voz é necessário que o começo e fim de segmentos sejam respeitados com silêncio e que os arquivos de áudio tenham uma mesma configuração (amostragem, quantidade de canais e *codec*). As transcrições manuais são as mais adequadas para serem utilizadas no *corpus* de voz, pois uma série de regras devem ser respeitadas na escrita do texto. A utilização de legendas, por exemplo, gera uma série de dificuldades pois, geralmente, o tempo de início e fim de segmentos de legenda estão em desordem com o áudio, o autor das legendas omite algumas palavras que foram faladas pelo locutor por questões sintáticas da frase e não existe um padrão de escrita. Com relação às configurações de áudio, é necessário que exista um padrão pois palavras iguais ditas em áudios diferentes com configurações diferentes iriam gerar um vetor de características diferentes para uma mesma palavra, diminuindo assim a probabilidade de ocorrência da palavra. Portanto, é importante que os áudios possuam a melhor configuração de áudio possível e que seja comum a todos.

5.5. Ferramentas para construção de componentes básicos de sistemas ASR

A partir da definição de conceitos básicos para o reconhecimento de fala, pode-se iniciar o processo de construção destes sistemas. Os recursos e o tempo computacional necessários para a criação destes sistemas estão diretamente ligados à base de dados utilizada no treinamento. Por ser um processo de geração de probabilidades a partir de dados observados, o custo computacional é levado em consideração. O principal componente envolvido neste custo computacional é o modelo acústico. Desta forma, dependendo do tamanho *corpus* de voz utilizado, será necessário o uso de equipamentos melhores e até recursos específicos (GPUs por exemplo).

Para treinar, por exemplo, um modelo acústico com um *corpus* de voz de aproximadamente 100 horas é desejável contar com pelo menos uma máquina Linux (ex: Ubuntu 16.04) com processador Intel Core i7, 32 GB de RAM, 1 TB de HD e uma boa placa GPU (ex: Nvidia GTX-970). Com esta infraestrutura de hardware é possível treinar um modelo acústico em 5 dias.

Nesta seção, será apresentada ferramentas para criação dos componentes básicos descritos neste capítulo, bem como processo de instalação, dependências, receitas de treinamento e arquivos gerados. O kaldi, será a ferramenta responsável pelo treinamento do modelo acústicos e pela geração do sistema de decodificação. O SRILM, será a ferramenta responsável pela criação do modelo de linguagem. E por fim o Multilingual G2P, será a ferramenta responsável para construção do léxico em PT-BR.

5.5.1. Kaldi

O Kaldi é um *toolkit* que é usado principalmente por pesquisadores da área de reconhecimento de fala. Ele é semelhante a outro *toolkit* muito conhecido da área que é o HTK[13], porém o Kaldi propõe algumas facilidades e a simplificação dos processos envolvendo sistemas de reconhecimento de fala, principalmente com relação ao treinamento de modelos acústicos e decodificação. Suas principais características são: Integração a nível de código com Finite State Transducers (FSTs); Suporte a álgebra linear; algoritmos fornecidos de forma genérica; licença aberta; disponibilização de receitas completas para a construção de sistemas de reconhecimento de fala [Povey et al., 2011].

As ferramentas apresentados pelo Kaldi nesta seção, serão aquelas utilizadas para treinamento de modelos acústicos e na construção do decodificador Kaldi GStreamer [Alumäe, 2014].

O processo de instalação do Kaldi conta com os seguintes passos:

1. Faça o *clone* do repositório do Kaldi no GitHub
    a. *$ git clone https://github.com/kaldi-asr/kaldi*
2. Acesse a pasta kaldi/tools/
    a. Execute o comando abaixo para verificar dependências do Kaldi
        i. *$ ./extras/check_dependencies.sh*
    b. Se alguma dependência precisar ser instalada, o script irá indicá-la, caso contrário lançará “all OK”
    c. É importante que o passo de verificação de dependências seja seguido para evitar erros futuros

[13] http://htk.eng.cam.ac.uk/

   d. Execute o comando abaixo:
      i. $ *make*
   e. Este processo pode ser demorado. Caso esteja disponível múltiplos núcleos na máquina, é possível utilizar o parâmetro "-j" para paralelizar o processo. Ex: uso de 4 CPU's
      i. $ make -j 4
3. Agora, acesse a pasta kaldi/src/
   a. Execute os comandos abaixo:
      i. $ ./configure --shared
      ii. $ make depend
      iii. $ make
   b. Este processo pode ser demorado. Caso esteja disponível múltiplos núcleos na máquina, é possível utilizar o parâmetro "-j" para paralelizar o processo. Ex: uso de 8 CPU's
      i. $ make depend -j 8
      ii. $ make -j 8
4. Ainda, é necessário a anexação do SRILM ao Kaldi, pois algumas receitas disponibilizadas fazem uso da ferramenta para modelos de linguagem de forma integrada. Assim, baixe o código fonte do SRILM e o deposite na pasta *kaldi/tools/*. Após isto, execute o script dentro da pasta *$ ./install_srilm.sh*
5. Após estes passos, o kaldi está instalado
6. Para testar se tudo ocorreu bem, é recomendável o teste com algum dos exemplos disponíveis na pasta do kaldi. A maioria deles são receita prontas de treinamento para ajuda os desenvolvedores. Um destes exemplo de execução de treinamento é apresentado abaixo:
   a. Acesse a pasta kaldi/egs/yesno/s5/
   b. Execute o comando abaixo
      i. $ ./run.sh
   c. O treinamento de um sistema de transcrição simples será iniciado e ao final apresentará a WER deste sistema para uma base de testes. Caso a instalação tenha ocorrido com sucesso, o aparecimento da WER indicará que o Kaldi está funcionando plenamente para treinamento de modelos acústicos

Com o Kaldi instalado, é importante entender os principais passos de uma receita de treinamento de modelos acústicos, quais arquivos de entrada utilizar e em qual formato. Estes detalhes podem ser encontrados na seção 5.5.1.1. Com relação ao decodificador, a seção 5.5.1.2., apresentará um sistema de decodificação simples e os principais elementos para criação deste componente do sistema de reconhecimento de fala.

5.5.1.1. Kaldi como ferramenta para criação de modelos acústicos

Para treinar o modelo acústico são necessários alguns arquivos principais de indicação do *corpus* de voz. É importante enfatizar que o *corpus* de voz esteja bem segmentado para que o sistema de reconhecimento de fala possua uma acurácia melhor. Ainda, uma boa prática para preparação do *corpus* de voz, é aconselhável que cada arquivo de áudio tenha um identificador (id) próprio e que este seja padronizado. A Figura 5.13. ilustra um exemplo destes arquivos pertencentes ao *corpus* de voz.

```
A001.wav  A088.wav  A175.wav  A263.wav  A361.wav  A449.wav  A537.wav
A002.wav  A089.wav  A176.wav  A264.wav  A362.wav  A450.wav  A538.wav
A003.wav  A090.wav  A177.wav  A265.wav  A363.wav  A451.wav  A539.wav
A004.wav  A091.wav  A178.wav  A266.wav  A364.wav  A452.wav  A540.wav
A005.wav  A092.wav  A179.wav  A267.wav  A365.wav  A453.wav  A541.wav
A006.wav  A093.wav  A180.wav  A268.wav  A366.wav  A454.wav  A542.wav
A007.wav  A094.wav  A181.wav  A269.wav  A367.wav  A455.wav  A543.wav
A008.wav  A095.wav  A182.wav  A270.wav  A368.wav  A456.wav  A544.wav
A009.wav  A096.wav  A183.wav  A271.wav  A369.wav  A457.wav  A545.wav
A010.wav  A097.wav  A184.wav  A272.wav  A370.wav  A458.wav  A546.wav
A011.wav  A098.wav  A185.wav  A273.wav  A371.wav  A459.wav  A547.wav
```

Figura 5.13. Organização do *corpus* de voz

Neste caso, por exemplo, o identificador é constituído de um carácter alfabético e de um número de três dígitos, gerando assim mais de 18 mil combinações possíveis de identificadores. O áudios estão no formato .WAV.

Como foi descrito na seção 5.3., o *corpus* de voz é composto, além dos arquivos de áudio, das respectivas transcrições exatas de cada arquivo de áudio. Para esta informação, o Kaldi faz uso de quatro arquivos principais, três deles essenciais. Os arquivos principais são:

- **wav.scp**: arquivo de texto com várias linhas, sendo que cada linha contém um identificador de arquivo e o caminho do arquivo de áudio identificado.

  O arquivo **wav.scp** tem a seguinte estrutura:

  <identificador_do_audio> <caminho_para_audio>

  A Figura 5.14. ilustra um exemplo de arquivo **wav.scp**.

```
A001 /media/dados/resources/database/speech/pt-BR/rnp/8k/A001.wav
A002 /media/dados/resources/database/speech/pt-BR/rnp/8k/A002.wav
A003 /media/dados/resources/database/speech/pt-BR/rnp/8k/A003.wav
A004 /media/dados/resources/database/speech/pt-BR/rnp/8k/A004.wav
A005 /media/dados/resources/database/speech/pt-BR/rnp/8k/A005.wav
A006 /media/dados/resources/database/speech/pt-BR/rnp/8k/A006.wav
A007 /media/dados/resources/database/speech/pt-BR/rnp/8k/A007.wav
```

Figura 5.14. Arquivo wav.scp

  Pode-se observar que, por exemplo, o arquivo de áudio cujo identificador para o Kaldi é *A001*, é representado pelo arquivo de áudio *A001.wav*. Repare que o nome do identificador e do arquivo de áudio são iguais, porém isto não é regra. O arquivo de áudio cujo identificador é *A001* para o Kaldi poderia estar relacionado a um arquivo *videoaula.wav*, por exemplo;

- **segments**: arquivo de texto com várias linhas, sendo que cada linha é composta por um identificador de segmento, identificador do arquivo de áudio onde existe esse segmento e a região do arquivo de áudio onde esse segmento existe. Este arquivo não é necessário, quando cada arquivo de áudio representa um pequeno segmento de fala. Se cada arquivo de áudio tiver mais de 30 segundos, é

necessário que use o arquivo **segments** para indicar uma possível segmentação para o áudio.

Caso seja necessário utilizar o arquivo **segments**, é importante que os segmentos de áudio sejam pequenos (10~30 segundos) e que a transcrição em cada segmento seja bem alinhada. Caso as transcrições exatas estejam mal alinhadas, o modelo acústico vai ser construído associando característica acústicas da pronúncia de uma palavra à outra. Existem ferramentas para segmentação de áudio a partir da identificação de silêncio[14]. Estas ferramentas podem auxiliar na segmentação do áudio diminuindo o esforço manual para esta tarefa de preparação de dados. Existe também um projeto intitulado Gentle[15], que utiliza o Kaldi, para realizar o alinhamento forçado de palavras com áudio. Está técnica também pode auxiliar na preparação da base, caso as informações de tempos dos segmentos não estejam disponíveis.

O arquivo **segments** tem a seguinte estrutura:

<identificador_do_segmento> <identificador_do_audio> <início> <fim>

A Figura 5.15., ilustra um exemplo de arquivo **segments**.

```
A001-000080-041481-042021 A001 414.81 420.21
A001-000081-042054-042264 A001 420.54 422.64
A001-000082-042477-042960 A001 424.77 429.60
A002-000000-000000-000522 A002 0.00 5.22
A002-000001-000603-001479 A002 6.03 14.79
A002-000002-001503-001659 A002 15.03 16.59
A002-000003-001716-002106 A002 17.16 21.06
```

Figura 5.15. Arquivo segments

Neste exemplo, temos a primeira coluna que representa o identificador do segmento de áudio (ex: *A001-000080-041481-042021*), a segunda coluna representa o identificador do áudio onde aquele segmento se encontra (ex: *A001*), a terceira coluna o tempo (em segundos) onde aquele segmento se inicia no áudio (ex: *414.81*) e a última coluna, o tempo (em segundos) onde o segmento termina no áudio (ex: *420.21*). Repare que os segmentos de áudio são sempre pequenos com relação ao tempo de duração. O segmento, exemplificado acima, tem apenas 5.4 segundos (420.21s - 414.81s = 5.4 s).

- **text**: arquivo de texto com várias linhas. Cada linha contém o identificador de segmento seguido de sua transcrição textual. Não deve haver pontuação, dígitos, siglas, abreviações, etc, na transcrição de cada segmento. Todo o texto deve estar padronizado, ou tudo maiúsculo ou tudo minúsculo.

  O arquivo **text** tem a seguinte estrutura:

  <identificador_do_segmento> <frase>

[14] https://github.com/wiseman/py-webrtcvad
[15] https://lowerquality.com/gentle/

A Figura 5.16. ilustra um exemplo de arquivo **text**

```
A001-000080-041481-042021 grupos aglomerados e superaglomerados de galáxias
A001-000081-042054-042264 de fato nosso universo é extraordinariamente grande
A001-000082-042477-042960 e universo tão extraordinariamente grande
                          é difícil imaginar que estamos sós
A002-000000-000000-000522 a vida
A002-000001-000603-001479 como a conhecemos hoje na terra é espetáculo que assistimos
                          praticamente todos os ambientes conhecidos pelo homem
A002-000002-001503-001659 mas do que ela depende
A002-000003-001716-002106 ela é o resultado das condições e recursos disponíveis
```

Figura 5.16. Arquivo text

Neste exemplo, temos a primeira coluna que representa o identificador do segmento e a segunda o texto contido dentro daquele segmento. Comparando com o arquivo segments, exemplificado na Figura 5.15., a frase "*grupos aglomerados e superaglomerados de galáxias*" é dita no aúdio A001 no segmento que começa no tempo *414.81*s e termina no tempo *420.21*s. Este segmento é representado pelo identificador *A001-000080-041481-042021*.

- **utt2spk**: arquivo de texto com várias linhas contendo o identificador do segmento e a identificação do locutor que gravou tal segmento. Se essa informação não estiver disponível, não há problemas.

  O arquivo **utt2spk** tem a seguinte estrutura:

  <identificador_do_segmento> <identificador_de_locutor>

  A Figura 5.17. ilustra um exemplo de arquivo **utt2spk**

```
A001-000080-041481-042021 A00180
A001-000081-042054-042264 A00181
A001-000082-042477-042960 A00182
A002-000000-000000-000522 A00200
A002-000001-000603-001479 A00201
A002-000002-001503-001659 A00202
A002-000003-001716-002106 A00203
```

Figura 5.17. Arquivo utt2spk

Neste exemplo cada segmento tem um locutor diferente. Este locutor é representado por um identificador único. No exemplo da Figura 5.17., está informação não é conhecida para cada segmento de áudio, por isso cada segmento possui um identificador diferente. Mas esta informação é interessante quando se deseja treinar um sistema de reconhecimento de voz para uso próprio ou para separar fala de homens, mulheres, crianças e idosos por exemplo.

Estes quatro arquivos principais e os arquivos de áudio, são os principais elementos para o treinamento de modelos acústicos. Para executar o processo, deve se utilizar algumas das receitas disponibilizadas pelo Kaldi no GitHub[16]. As receitas de treinamento são todas padronizadas (arquivos executáveis, pastas de *corpus* e o

[16] https://github.com/kaldi-asr/kaldi

caminho dos arquivos gerados), assim é necessário apenas anexar o *corpus* de texto à receita e executar o seguintes comandos:

1. Execute o comando *$ ./run.sh*
2. O resultado do treinamento pode ser encontrado na pasta *exp/*

As receitas de treinamento podem ser encontrada em *kaldi/egs/*. Mais detalhes de implementação e ferramentas para sistemas de reconhecimento de fala podem ser encontradas na documentação do Kaldi.

5.5.1.2. Kaldi como ferramenta para criação de decodificador

O Kaldi GStreamer[17], é um sistema para decodificação de sinal de fala em sequência textual aproximada. Sua principal característica é ser um servidor full-duplex de reconhecimento de fala em tempo real onde é possível indicar qual modelo será utilizado pelo sistema. Neste passo, com o modelo acústico já implementado, é possível indicar o modelo para ser utilizado pelo servidor de reconhecimento de fala. O decodificador utiliza o Kaldi e GStreamer para implementação, porém foi implementado um *dockerfile*[18] do projeto do decodificador, com todas as dependências compiladas.

Além dos arquivos gerados pelo Kaldi, no treinamento de modelos, o decodificador faz uso de um arquivo principal de configuração (.yaml), para inicialização do serviço de reconhecimento de fala. Existem alguns exemplos[19] [20] [21] deste tipo de arquivo. A Figura 5.18. apresenta uma parte deste arquivo de configuração, onde deve ser indicado cada arquivo do modelo.

```
model : test/models/english/tedlium_nnet_ms_sp_online/final.mdl
word-syms : test/models/english/tedlium_nnet_ms_sp_online/words.txt
fst : test/models/english/tedlium_nnet_ms_sp_online/HCLG.fst
mfcc-config : test/models/english/tedlium_nnet_ms_sp_online/conf/mfcc.conf
ivector-extraction-config : test/models/english/tedlium_nnet_ms_sp_online/conf/ivector_extractor.conf
```

Figura 5.18. Arquivo de configuração do decodificador

Cada linha dessa, deve indicar o arquivo gerado pelo treinamento.

Para instalação e construção do decodificador, deve-se seguir os passos abaixo:

1. O Docker[22] deve estar instalado na máquina onde o decodificador será construído, para instalá-lo, siga os passo na página do Docker Docs.
2. Com o Docker instalado, dê um *pull* na imagem do Docker Hub
    a. *$ docker pull jcsilva/docker-kaldi-gstreamer-server*
3. Crie uma pasta, onde ficará localizado seus modelos. Uma sugestão de criação é:
    a. *$ sudo mkdir /media/kaldi_models/*
4. Execute o comando abaixo para criar o *container* e inicializar o serviço para o arquivo .yaml gerado

[17] https://github.com/alumae/kaldi-gstreamer-server
[18] https://github.com/jcsilva/docker-kaldi-gstreamer-server
[19] https://github.com/alumae/kaldi-gstreamer-server/blob/master/sample_worker.yaml
[20] https://github.com/alumae/kaldi-gstreamer-server/blob/master/estonian_worker.yaml
[21] https://github.com/alumae/kaldi-gstreamer-server/blob/master/sample_english_nnet2.yaml
[22] https://docs.docker.com/engine/installation/

   a. *$ docker run -it -p 8080:80 -v /media/kaldi_models:/opt/models jcsilva/docker-kaldi-gstreamer-server:latest /bin/bash*
   b. *$ /opt/start.sh -y /opt/models/model.yaml*
5. Após este passo, volte para a máquina *host* (Ctrl + P e Ctrl + Q). O serviço de reconhecimento de fala está ouvindo na porta 8080
6. Um exemplo de execução do serviço pode ser feito utilizando o *script* de cliente
   a. *$ wget https://raw.githubusercontent.com/alumae/kaldi-gstreamer-server/master/kaldigstserver/client.py -P /tmp*
7. Caso seja necessário parar o serviço, execute o comando abaixo dentro do *container*
   a. *$ /opt/stop.sh*

5.5.2. SRILM

O SRILM (do Inglês SRI Language Modeling toolkit) é um *toolkit* para construção e aplicação de modelos estatísticos de linguagem que é utilizado principalmente em Reconhecimento de Fala, Tradução Automática e Correção de Palavras.

No contexto de reconhecimento de fala, um exemplo de aplicação seria a utilização errada das palavras "sela" e "cela". Embora o som reproduzido pela fala para ambas seja idêntico, elas possuem significados completamente diferentes. A palavra "sela" é relacionado a sela de cavalo e a palavra "cela" é relacionado a cela de prisão. Assim, a probabilidade da frase "Pedro está em uma **cela** separada" é maior que a probabilidade da frase "Pedro está em uma **sela** separada". Esta informação é disponibilizada pelo modelo de linguagem e apresenta um reconhecimento de fala mais coerente.

Como descrito na seção 5.2.2., a probabilidade de ocorrência de uma palavra é dada a partir da probabilidade de ocorrência desta palavra dado que uma outra palavra ocorreu anteriormente. Com isso, a probabilidade de ocorrência de uma frase é dada pela produtório da ocorrência de cada uma das palavras desta frase. Por exemplo, $P(\text{"}Pedro\ está\ em\ uma\ cela\ separada\text{"})$ pode ser definido como[23]:

$$P(\text{"}Pedro\ está\ em\ uma\ cela\ separada\text{"}) = P(Pedro|<s>) \times$$
$$P(está|Pedro) \times P(em|está) \times P(uma|em) \times$$
$$P(cela|uma) \times P(separada|cela) \times P(</s>|separada)$$

A probabilidade de uma palavra é definido pela probabilidade de ocorrência da palavra dado que uma outra ocorreu. Isto quer dizer que somente um palavra, que aconteceu anteriormente, é levada em consideração na geração das probabilidades. Este modelo, então, é chamado Unigram. Existem também modelos Bigram, Trigam e N-gram. Para está mesma frase, um exemplo de modelo Bigram, resultaria no seguinte:

$$P(\text{"}Pedro\ está\ em\ uma\ cela\ separada\text{"}) =$$
$$P(está|<s>\ Pedro) \times P(em|Pedro\ está) \times P(uma|\ está\ em) \times$$
$$P(cela|\ em\ uma) \times P(separada|uma\ cela) \times P(</s>|cela\ separada)$$

[23]Os termos "<s>" e "</s>" representam início e fim de frase respectivamente

Para os modelos N-gram, é importante ressaltar que a medida que N cresce, o gasto computacional se torna muito grande e muitas vezes inviável para geração dos modelos. Assim é comum utilização de modelos Trigram, 4-gram e 5-gram.

O SRILM pode ser utilizado para treinar esses diversos modelos, com diferentes valores de n-gram, métodos de desconto, interpolação de modelos etc.

Para o processo de instalação do SRILM é necessário alguns passos descritos abaixo. As etapas descritas abaixo são para instalação em um ambiente Linux 64 bits.

1. Faça o download de todas as dependências descritas na página de Download do SRI
2. Faça o download da versão mais recente do SRILM preenchendo o formulário na página de Download o SRI
3. Extraia o arquivo na pasta onde deseja instalar o SRILM
4. Abra para edição, o arquivo "Makefile" dentro da pasta onde o SRILM foi descompactado.
5. Encontre uma linha comentada como esta:
    a. *# SRILM = /home/speech/stolcke/project/srilm/devel*
    b. Remova o caracter "*#*"
    c. Substitua "/home/speech/stolcke/project/srilm/devel" pelo caminho para a pasta onde o SRILM foi descompactado
6. Usando o terminal do Linux, navegue até a pasta onde o SRILM foi descompactado
7. Execute o comando **make**
8. Os arquivos binários compilados do SRILM, devem ser encontrados em /[caminho_para_SRILM]/bin/i686-m64/

Os passos para execução de um treinamento simples com o SRILM são:

1. Navegue até a pasta /[caminho_para_SRILM]/bin/i686-m64/
2. Execute o comando:
    a. *./ngram-count -text [corpus_texto] -order 3 -lm [arquivo_de_saida]*

O executável binário, *ngram-cout*, é a principal ferramenta do SRILM. Ele é responsável por treinar os modelos de linguagem e possuem uma série de parâmetros de configuração de treinamento.

- ***-text*** *[corpus_texto]*: Este parâmetro representa a entrada de texto para o treinamento do modelo de linguagem. Este arquivo deve respeitar os padrões discutidos na seção 5.4. para *corpus* de texto
- ***-order*** *n*: Este parâmetro representa a ordem de *n*-gram que deseja que o modelo seja treinado. No exemplo acima foi treinado o modelo Trigram para o *corpus* de texto de entrada
- ***-lm*** *[arquivo_de_saida]:* arquivo de saída do treinamento onde será descrito as probabilidades de cada gram treinado. Um exemplo de arquivo será apresentado nas próximas páginas.

Existem ainda outros parâmetros para extrair informações e melhorar o modelo. Entre eles podemos citar alguns mais importantes como:

- ***-write-vocab*** *file:* Escrever no arquivo *file* o vocabulário de um dado *corpus* de texto
    - Ex: *./ngram-count -text [corpus_texto] write-vocab [arquivo_de_saida]*
- ***-prune*** *threshold:* Efetuar um corte no modelo final, definido por um *threshold*
    - Ex: *./ngram-count -text [corpus_texto] -order 3 -prune 1e-10 -lm [arquivo_de_saida]*
- ***-wbdiscount****n:* Parâmetro para uso do método de desconto Witten-Bell. Onde *n* é o valor do gram onde se deseja aplicar o desconto
    - Ex: *./ngram-count -text [corpus_texto] -order 3 -wbdiscount3 -wbdiscount2 -lm [arquivo_de_saida]*
- ***-ndiscount****n:* Parâmetro para uso da lei de desconto natural de Ristad. Onde *n* é o valor do gram onde se deseja aplicar o desconto
    - Ex: *./ngram-count -text [corpus_texto] -order 3 -ndiscount3 -ndiscount2 -ndiscount1 -lm [arquivo_de_saida]*
- ***-kndiscount****n:* Parâmetro para uso do método de desconto Kneser-Ney modificado. Onde *n* é o valor do gram onde se deseja aplicar o desconto
    - Ex: *./ngram-count -text [corpus_texto] -order 3 -kndiscount2 -ndiscount1 -lm [arquivo_de_saida]*
- ***-ukndiscount****n:* Parâmetro para uso do método de desconto Kneser-Ney original. Onde *n* é o valor do gram onde se deseja aplicar o desconto
    - Ex: *./ngram-count -text [corpus_texto] -order 3 -ukndiscount2 -lm [arquivo_de_saida]*
- ***-interpolate****n:* Parâmetro para interpolação de gram de ordem inferior a *n*.
    - Ex: *./ngram-count -text [corpus_texto] -order 3 -interpolate3 -kndiscount2 -ndiscount1 -lm [arquivo_de_saida]*

O resultado final de um treinamento de modelo de linguagem é um arquivo do tipo ARPA que é criado no caminho que for indicado no parâmetro *-lm [arquivo_de_saida]*. A cara do arquivo final de treinamento de modelo é apresentado na Figura 5.19., onde o comando utilizado para geração foi: *./ngram-count -text [corpus_texto] -order 4 -lm [arquivo_de_saida]*.

```
\data\
ngram 1=70
ngram 2=98

\1-grams:
-1.22617        </s>
-99     <s>     -0.8230478
-1.305351       a       -0.756101
-2.004321       aprovada        -0.2967086
-2.004321       arroz   -0.2923438
-2.004321       bom     -0.2923438
-2.004321       caro    -0.2923438
-2.004321       centavos        -0.2744322
-2.004321       chuvoso -0.2967086
-2.004321       cidade  -0.2834803

\2-grams:
-0.90309        <s> </s>
-0.60206        <s> a
-0.90309        <s> bom
-0.90309        <s> maria
-0.60206        <s> o
-0.7781513      a dezoito
-0.7781513      a ideia
-0.7781513      a maior
-0.7781513      a realidade
-0.7781513      a temperatura
-0.30103        aprovada em

\end\
```

Figura 5.19. Parte do arquivo ARPA de saída do treinamento de modelo de linguagem treinado com SRILM. Para este modelo foi utilizado um *corpus* de texto com 6 frases e 70 palavras sem repetição. Um *corpus* deste tamanho é considerado muito pequeno e deve ser utilizado apenas para exemplificação e não para aplicação.

Outra ferramenta importante do SRILM é o executável binário *ngram*. Ele é responsável por avaliação do modelo, medição de perplexidade e interpolação entre dois ou mais modelos treinados. Um exemplo de utilização para medição da perplexidade pode ser feito com o comando: *./ngram -lm [arquivo_ARPA] -ppl [corpus_teste]*. Para este comando, o resultado é como ilustrado na Figura 5.20. O *corpus* de teste do modelo de linguagem tem o mesmo formato do *corpus* de texto apresentado na seção 5.4., porém com tamanho bem menor. Um valor de referência para o tamanho do *corpus* de teste é de 20% do tamanho no *corpus* de texto no qual o modelo foi treinado, porém não existe um valor fixo para isto. Além disso, é importante que as sentenças que estiverem presentes no *corpus* de texto não façam parte do *corpus* de teste pois a avaliação não reproduziria o que acontece de fato, numa situação real de utilização do modelo de linguagem. Portanto, é aconselhável que separe o *corpus* de texto e *corpus* de teste antes

do treinamento, caso a avaliação seja um ponto de interesse. Repare que medir a perplexidade do modelo de linguagem não é um ponto vital para o funcionamento, somente um indicativo para avaliar o quão bom é um modelo de linguagem.

```
file corpus_teste: 6 sentences, 96 words, 11 OOVs
1 zeroprobs, logprob= -183.195 ppl= 108.517 ppl1= 151.667
```

Figura 5.20. Medição de perplexidade com SRILM

Na Figura 5.20, pode-se observar alguns resultados para o arquivo de teste do modelo de linguagem. A primeira linha apresenta o nome do arquivo de teste, o número de sentenças que ele possui, quantidade de palavras e quantas dessas não fazem parte do vocabulário do modelo de linguagem que está sendo avaliado (OOV - *Out of Vocabulary).* Na segunda linha alguns valores numéricos são representados, porém o valor mais importante é o *ppl* que representa a perplexidade do modelo de linguagem para o *corpus* de teste fornecido. Neste caso a perplexidade é de 108.517. É importante ressaltar que a perplexidade é o inverso da probabilidade, com isso quanto menor o valor da perplexidade melhor é o modelo de linguagem.

O executável binário *ngram* também possui outros parâmetros que são ferramentas importantes para melhorias e experimentos com modelo de linguagem. Entre eles podemos citar:

- ***-mix-lm*** *file:* Parâmetro utilizado quando se deseja interpolar dois modelos. O *file* deve ser o arquivo ARPA que se deseja interpolar.
    - Ex: *./ngram -lm [lm_ARPA_1] -order 4 -mix-lm [lm_ARPA_2]*
- ***-lambda*** *weight:* O parâmetro lambda define o peso do modelo principal na interpolação de modelos. O valor *default* é 0.5
    - Ex: *./ngram -lm [lm_ARPA_1] -order 4 -mix-lm [lm_ARPA_2] -lambda 0.6*
- ***-mix-lm2*** *file:* Este parâmetro deve ser usado quando se deseja interpolar 2 modelos ou mais (*-mix-lm3, -mix-lm4 …).* O *file* deve ser o arquivo ARPA que se deseja interpolar.
    - Ex: *./ngram -lm [lm_ARPA_1] -order 4 -mix-lm [lm_ARPA_2] -mix-lm2 [lm_ARPA_3]*
- ***-mix-lambda2*** *weight:* Esse parâmetro deve ser usado quando se deseja interpolar 2 modelos ou mais (*-mix-lambda3, -mix-lambda4 …).* O valor *weight* representará o peso de cada modelo na interpolação.O valor *default* é 0.5
    - Ex: *./ngram -lm [lm_ARPA_1] -order 4 -mix-lm [lm_ARPA_2] -lambda 0.4 -mix-lm2 [lm_ARPA_3] -mix-lambda2 0.6*

Os modelos de linguagem são sempre dependentes do contexto de treinamento em que foram construídos, com isso não é possível afirmar qual o melhor método de treinamento (“qual método de desconto utilizar?”; “qual número de grams no treinamento?”; “é necessário interpolar os grams?”; “é ideal criar dois modelos e interpola-los?”; “qual peso definir para cada modelo na interpolação?”). Portanto, executar experimentos de treinamento avaliando os modelos a partir da perplexidade é sempre uma forma interessante de construir um bom modelo de linguagem para a aplicação desejada.

5.5. Aplicações

A viabilidade de extrair informações de arquivos multimídia com áudio e vídeo possibilita uma série de aplicações em diversos contextos.

Uma das aplicações mais conhecidas para reconhecimento automático de fala é o uso de transcrição simultânea na criação de legendas ocultas para sistemas de televisão. O AUDIMUS.MÉDIA [Meinedo et al., 2003] é um sistema que gera legendas ao vivo para emissoras de TV e de *streaming*. O objetivo do sistema de reconhecimento de fala, neste contexto, é gerar a legenda sem intervenção humana evitando custo com serviços de legendagem manual.

O reconhecimento de fala também é utilizado para questões de aprendizado e acessibilidade. O CineAD [Campos et al., 2014] é um sistema para geração automática de roteiros de audiodescrição (AD), que é um recurso essencial para pessoas cegas e com baixa visão para acesso ao cinema. Este trabalho utiliza, dentre outras técnicas, o reconhecimento de fala para geração de legendas que serão utilizadas na criação de AD. O reconhecimento de fala também pode ser utilizado para questões de aprendizado, como descrito em [Higgins and Raskind, 1999], onde foi avaliado o uso de um sistema ASR para auxiliar na educação de crianças e adolescentes com dificuldade de aprendizagem. O reconhecimento de fala também pode ser usado, na área da educação, para melhorar o desempenho de crianças e adolescentes como descrito em [Hämäläinen et al., 2013], onde é apresentado um jogo educacional que utiliza a técnica de reconhecimento de fala com o objetivo de melhorar a coordenação física e as habilidades em matemática e música dos estudantes.

Ainda, podemos citar aplicações práticas e cotidianas para sistemas de reconhecimento de fala. O uso de assistentes virtuais em *smartphones* está presente no sistema das principais empresas da área de telefonia móvel. Os assistentes são elaboradas para permitir que o usuário interaja com o dispositivo a partir da voz, utilizando assim um sistema de reconhecimento de fala e técnicas para processamento do resultado do reconhecimento. Estes sistemas de reconhecimento também podem ser encontrados em diversos dispositivos que interagem com usuários, como por exemplo carros, eletrodomésticos, sistemas de autenticação, casas inteligentes etc. No âmbito dos negócios, o CALO [Tur et al., 2008] é um assistente de reunião, que fornece informações automáticas de reuniões utilizando, dentre outras técnicas, o uso de reconhecimento de fala para legendagem, geração de ata e busca por conteúdos de conversas.

Também podemos citar aplicações diretas para sistemas de reconhecimento de fala, como os esforços de alguns trabalhos [Raimond and Lowis, 2012; Coelho and Souza, 2015; Yang and Meinel, 2014] que utilizam estes sistemas para auxiliar na anotação, busca e recomendação de vídeos. Este processo utiliza o texto de resultado da transcrição, como entrada para algoritmos de anotação semântica na intenção de identificar termos para classificar o conteúdo multimídia. Estes termos podem ser relacionados a bases de conhecimento e desta forma criar uma estrutura de dados ligados podendo assim, utilizar técnicas para recomendação de conteúdos. Assim, pode-se perceber que os cenários para aplicação de sistemas de reconhecimento de fala

são amplos e que este tipo de sistema pode auxiliar tanto na automatização de tarefas quanto na extração de informações para aplicações.

5.5. Conclusões

Neste trabalho, apresentamos a arquitetura básica de sistemas de reconhecimento automático de fala, descrevendo os principais elementos, funcionalidades e métodos para treinamento e criação de sistemas ASR. Também apresentamos aplicações diretas e indiretas para estes sistemas com o objetivo de demonstrar como estes sistemas se comportam nos mais diversos contextos.

O reconhecimento de fala é um tema recorrente na área de processamento de linguagem natural e conta com uma comunidade ativa e em constante processo de melhorias paras os diversos elementos desses sistemas. Assim, o reconhecimento de fala é um tema promissor e pode ser aplicado a diversos problemas, principalmente da área de multimídia.

Referências

Alumäe, Tanel. "Full-duplex Speech-to-text System for Estonian." Baltic HLT. 2014.

Arlindo Oliveira da Veiga. Treino não supervisionado de modelos acústicos para reconhecimento de fala. Ph.D. Dissertation. Universidade de Coimbra. 2013.

Bengio, Yoshua. "Markovian models for sequential data." Neural computing surveys 2.199 (1999): 129-162.

Campos, Virginia Pinto, T. M. U. Araujo, and G. L. Souza Filho. "CineAD: Um Sistema de Geração Automática de Roteiros de Audiodescrição." Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia) (2014).

Chan, Ho Yin, and Phil Woodland. "Improving broadcast news transcription by lightly supervised discriminative training." Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on. Vol. 1. IEEE, 2004.

Coelho, S. A. ; Souza, J. F. . Anotação semântica de transcritos para indexação e busca de vídeos. In: Conferência Ibero Americana WWW/INTERNET, 2014, Porto, Portugal. 12ª Conferência Ibero Americana WWW/INTERNET, 2014. v. 1. p. 51-58.

Cuadros, Carlos Daniel Riquelme. "Reconhecimento de voz e de locutor em ambientes ruidosos: comparação das técnicas MFCC e ZCPA." Dissertação de mestrado. Programa de Pós-Graduação em Engenharia de Telecomunicações da Escola de Engenharia da Universidade Federal Fluminense. (2007).

Dahl, George E., et al. "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition." IEEE Transactions on audio, speech, and language processing 20.1 (2012): 30-42.

Davis, Steven, and Paul Mermelstein. "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences." IEEE transactions on acoustics, speech, and signal processing 28.4 (1980): 357-366.

de Sá Pessoa, Luis A., Fábio Violaro, and Plínio A. Barbosa. "Modelos da Língua Baseados em Classes de Palavras para Sistema de Reconhecimento de Fala Contínua 10.14209/jcis. 1999.10." Journal of Communication and Information Systems 14.2 (1999).

Evermann, Gunnar, et al. "Training LVCSR systems on thousands of hours of data." Acoustics, Speech, and Signal Processing, 2005. Proceedings.(ICASSP'05). IEEE International Conference on. Vol. 1. IEEE, 2005.

Gonzalez, Marco, and Vera LS Lima. "Recuperação de informação e processamento da linguagem natural." XXIII Congresso da Sociedade Brasileira de Computação. Vol. 3. 2003.

Hämäläinen, Annika, et al. "A Multimodal Educational Game for 3-10-year-old Children: Collecting and Automatically Recognising European Portuguese Children's Speech." Speech and Language Technology in Education. 2013.

Higgins, Eleanor L., and Marshall H. Raskind. "Speaking to read: The effects of continuous vs. discrete speech recognition systems on the reading and spelling of children with learning disabilities." Journal of Special Education Technology 15.1 (1999): 19-30.

Hinton, Geoffrey, et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups." IEEE Signal Processing Magazine 29.6 (2012): 82-97.

Jurafsky, Daniel, and James H. Martin. "Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics". 2nd edition. Prentice-Hall. 2009.

Jurafsky, Dan. "Language Modeling". Natural Language Processing Group of Stanford University (2017). Disponível em https://web.stanford.edu/~jurafsky/ . Acesso em 3 de Setembro de 2017. Jurafsky

Katz, Slava. "Estimation of probabilities from sparse data for the language model component of a speech recognizer." IEEE transactions on acoustics, speech, and signal processing 35.3 (1987): 400-401.

Meinedo, Hugo, et al. "AUDIMUS. media: a Broadcast News speech recognition system for the European Portuguese language." Computational Processing of the Portuguese Language (2003): 196-196.

Miao, Yajie. "Kaldi+ PDNN: building DNN-based ASR systems with Kaldi and PDNN." arXiv preprint arXiv:1401.6984 (2014).

Neto, Nelson, Ênio Silva, and Erick Sousa. "Software usando reconhecimento e síntese de voz: o estado da arte para o Português brasileiro." Proceedings of the 2005 Latin American conference on Human-computer interaction. ACM, 2005.

Ney, Hermann, and Ute Essen. "On smoothing techniques for bigram-based natural language modelling." Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on. IEEE, 1991.

Panayotov, Vassil, et al. "Librispeech: an ASR corpus based on public domain audio books." Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on. IEEE, 2015.

Povey, Daniel, et al. "The Kaldi speech recognition toolkit." IEEE 2011 workshop on automatic speech recognition and understanding. No. EPFL-CONF-192584. IEEE Signal Processing Society, 2011.

Rabiner, Lawrence, and B. Juang. "An introduction to hidden Markov models." ieee assp magazine 3.1 (1986): 4-16.

Raimond, Yves, and Chris Lowis. "Automated interlinking of speech radio archives." LDOW 937 (2012).

Riccardi, G.; Hakkani-Tür, D. Z. Active and unsupervised learning for automatic speech recognition. In: INTERSPEECH, 2003.

Robert M Ochshorn and Max Hawkins. 2017. “Gentle forced aligner [computer program]. (2017). https://lowerquality.com/gentle/ Access date: 25 ago. 2017.

Rosenfeld, Roni, and Philip Clarkson. "Statistical language modeling using the CMU-Cambridge toolkit." (1997).

Silva, E., et al. "Modelos de linguagem n-grama para reconhecimento de voz com grande vocabulário." III workshop em tecnologia da informação e da linguagem humana. 2004.

Silva, E., et al. "Desenvolvimento de um sistema de reconhecimento automático de voz contínua com grande vocabulário para o Português Brasileiro." XXV congresso da sociedade Brasileira de computação. 2005.

Silva, Carlos Patrick Alves da. Um software de reconhecimento de voz para português brasileiro. MS thesis. Universidade Federal do Pará, 2010.

Siravenha, Ana, et al. "Uso de regras fonológicas com determinação de vogal tônica para conversão grafema-fone em português brasileiro". 7th International

Information and Telecommunication Technologies Symposium. 2008.

Stolcke, Andreas. "SRILM-an extensible language modeling toolkit." Interspeech. Vol. 2002. 2002.

Tevah, Rafael Teruszkin. Implementação de um sistema de reconhecimento de fala contínua com amplo vocabulário para o português brasileiro. Diss. Dissertação (mestrado). COPPE/UFRJ, M. Sc., Engenharia Elétrica, 2006.

Tur, Gokhan, et al. "The CALO meeting speech recognition and understanding system." Spoken Language Technology Workshop, 2008. SLT 2008. IEEE. IEEE, 2008.

Wessel, Frank, and Hermann Ney. "Unsupervised training of acoustic models for large vocabulary continuous speech recognition." IEEE Transactions on Speech and Audio Processing 13.1 (2005): 23-31.

Witten, Ian H., and Timothy C. Bell. "The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression." Ieee transactions on information theory 37.4 (1991): 1085-1094.

Yang, Haojin, and Christoph Meinel. "Content based lecture video retrieval using speech and video text information." IEEE Transactions on Learning Technologies 7.2 (2014): 142-154.

Ynoguti, Carlos Alberto. "Reconhecimento de fala contínua usando modelos ocultos de Markov.". Ph.D. Dissertation. Universidade Estadual de Campinas. 1999.

Young, S., et al. "The HTK book (v3. 4)." Cambridge University (2006).

Biografia Resumida dos Autores

Jairo Francisco de Souza é graduado em Ciência da Computação (UFJF), possui mestrado em Engenharia de Sistemas e Computação (UFRJ) e Doutorado em Informática (PUC-RJ). Atualmente é professor adjunto da Universidade Federal de Juiz de Fora e docente do programa de pós-graduação em Ciência da Computação da UFJF, onde atua em pesquisas nas áreas de recuperação de informação, processamento de linguagem natural e integração semântica.

Marcos Valadão Gualberto Ferreira é graduando do oitavo período do curso de Ciência da Computação na Universidade Federal de Juiz de Fora (UFJF) e estagiário da Rede Nacional de Ensino e Pesquisa (RNP). Atua há quase dois anos no projeto de Busca Avançada por Vídeos baseada em transcrição de áudio, metadados, anotação semântica e recomendação (GT-BAVi).

Chapter 6

Using Mobile Cloud Computing for Developing Context-Aware Multimedia Applications

- A Case Study of the CAOS Framework

Fernando A. M. Trinta[1], Paulo A. L. Rego[1], Francisco A. A. Gomes[1], Lincoln S. Rocha[1] and José N. de Souza[1]

[1]Federal University of Ceará. GREat Research Group

{trinta,almada,lincoln}@great.ufc.br {pauloalr,neuman}@ufc.br

Abstract

Mobile cloud computing (MCC) and Context-Aware Computing (CAC) are research topics in growing evidence. The former seeks to leverage cloud computing features to improve the performance of mobile applications and reduce the energy consumption of mobile devices, while the latter seeks effective ways to build applications that react to changes in its context environment. This short-course aims at presenting main concepts, solutions, and technologies related to the integration of MCC and context-aware applications. We will present different motivational scenarios, examples of applications, as well as a practical guide to the development of a context-aware multimedia Android application using the framework CAOS. In addition, we will highlight research challenges and opportunities that come with such integration.

Resumo

Mobile Cloud Computing (MCC) e a Computação Sensível ao Contexto são tópicos de pesquisa em crescente evidência. O primeiro procura utilizar recursos da computação em nuvem para melhorar o desempenho de aplicações móveis e reduzir o consumo de energia dos dispositivos, enquanto o último busca formas eficazes de criar aplicações que reajam às mudanças de contexto do ambiente. Este minicurso tem como objetivo apresentar os principais conceitos, soluções e tecnologias relacionadas à integração de MCC e sensibilidade ao contexto. Serão apresentados diferentes cenários motivacionais, exemplos de aplicações, bem como um guia prático de como desenvolver uma aplicação multimídia sensível ao contexto utilizando o framework CAOS. Além disso, serão discutidos desafios e oportunidades de pesquisa relacionados à integração entre os dois tópicos.

6.1. Contextualization

Mobile Devices such as smartphones and tablets have become important tools for daily activities in modern society. These devices have improved their processing power due to faster processors, increased storage resources and better network interfaces. Mobile devices have also been gradually equipped with a plethora of sensors that gather data from the user's environment (e.g., location and temperature). One challenge for mobile and distributed computing is to explore the changing environment where mobile devices are inserted with a new kind of mobile application that take benefits from features of their dynamic environment. These new types of systems are called context-aware applications.

Nowadays, context management and inferences are becoming complex processes, as the amount of data increases and new algorithms are being proposed [Gomes et al. 2016]. In this scenario, cloud services can offer an interesting option by taking on more intensive context management tasks and performing these tasks only once for multiple users. However, since most contextual information is sensed and captured by mobile devices themselves, it is not always clear whether it is wise to send all the sensor data to the cloud for a remote processing or perform the whole processing of contextual data locally by the mobile device. These trade-offs depend on factors such as the amount of data being transferred and the type of processing that is required. According to [Naqvi et al. 2013], the Mobile Cloud Computing paradigm, from a context-sensitive perspective, can be seen as a promising field of research that seeks to find effective ways of doing service in Cloud-aware applications and clients.

Offloading is the main research topic in MCC [Fernando et al. 2013] and represents the idea of moving data and computation from mobile devices with scarce resources to more powerful machines [Rego et al. 2016]. There are several opportunities where computation and data offloading can bring improvements to context-aware mobile applications as well as multimedia applications. Some scenarios include: (i) devices with low processing power can use the cloud to act on their behalf (e.g., rendering images or videos) [Costa et al. 2015], (ii) it would be possible to leverage cloud resources to save energy by delegating tasks away from mobile devices [Barbera et al. 2013], and (iii) to save data storage from mobile devices.

In order to understand different approaches, concepts, and challenges about the integration between Mobile Cloud Computing (MCC) and Context-Aware Computing (CAC), this short-course is proposed as an opportunity to disseminate and improve the understanding of the target audience regarding the union of these two themes. To the best of our knowledge, until the writing of this document, there were no similar initiatives on the subject in any major symposium or conference in Brazil. We will use a framework developed by the GREat research group[1], called CAOS - Context-Aware and Offloading System [Gomes et al. 2017], which is a software platform for the development of context-aware mobile applications based on the Android platform. CAOS supports both data and computing offloading (i.e., it enables the migration of computing and contextual data from mobile devices to cloud platforms in a transparent and an automatic way).

The integration of cloud services is an increasing trend in several areas. This short-course aims to present how this integration for the domain of context-aware applications

[1]GREat website: http://www.great.ufc.br

can be done, focusing mainly on multimedia applications.

6.2. Theoretical Background

6.2.1. Context-Aware Computing

Mobile devices, such as smartwatches, smartphones, tablets, and ultrabooks, have become part of our everyday lives. They allow users to access a vast range of applications on-the-go, from personal agendas to existing social networks, from standalone applications running on the device, to distributed applications interacting with the environment [Herrmann 2010]. An important benefit of using an application on a mobile device is the possibility to use it anywhere and anytime. In fact, it can be seen as a step toward to achieve the Mark Weiser's Ubiquitous Computing [Weiser 1991] vision, embedding the computation in the user's devices and turning the user-interaction more soft and natural.

To achieve the calm interaction between users and computers is a complex task. Applications running on mobile devices have to interact with a dynamic environment, where available users, devices, and resources change over time. Therefore, the software running on these devices, interacting with users and the environment, must be designed from scratch taking into account these changes and adapting itself in order to achieve the desired behavior. The ability to perceive changes in the environment and adapt its behavior to meet these changes is called context-awareness. [Preuveneers and Berbers 2007].

Before precisely define context-aware we must first understand what context is. In that sense, we introduce in the following the most spread definition of context and context-awareness given by Anind K. Dey [Dey 2001]. Next, we introduce the Viana's *et al.* [Viana et al. 2011] context-aware definition adopted in this document.

> "*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*" [Dey 2001]

> "*A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.*" [Dey 2001]

Viana *et al.* [Viana et al. 2011] extends Dey's definition taking into account the dynamic nature of the context structure and its acquisition. In that sense, not only the values of the context elements evolve over time, but also the number of context elements changes. According to Viana *et al.* [Viana et al. 2011], the elements composing the context depends on the system's interest and the chance to observe them. So, once the system execution evolves, the set of observed context elements also evolve. In a nutshell, the context can be determined by the intersection of two sets of information in an instant t. The first one is called Zone of Interest (ZoI), comprising the set of contextual elements relevant to the system. The second set is called Zone of Observation (ZoO) and it describes the set of contextual elements that can be collected by the system. Figure 6.1 shows a Venn diagram with these two sets of information. Their intersection is the Zone of Context (ZoC) that is composed of any information that can be described and observed. All

three zones may change over time. For instance, ZoO may change when a sensor becomes unavailable; the ZoI may change when the system changes its interests, not requiring specific contextual information any longer. At last, the ZoC changes when changes on ZoI and ZoO affect their intersection.

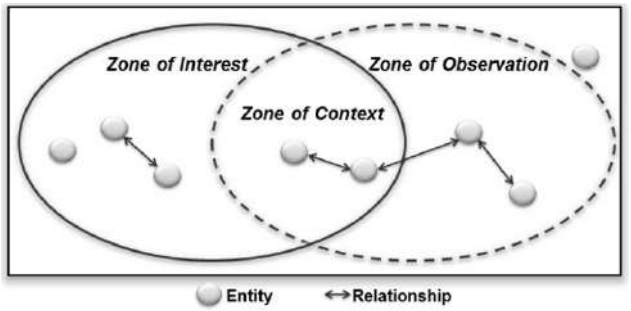


Figure 6.1. Context definition proposed by Viana *et al.* [Viana et al. 2011].

In short, context-awareness is the ability of the computer system to discover and react to its context changes. Mobile applications can benefit from context-awareness to provide personalized services to users and adapt their structure and behavior accordingly in order to save or optimize the mobile device resources usage. The mobile devices constrains (e.g., battery power, bandwidth, and storage capacity) and issues related to the context management (e.g., capture, process, and delivery), security, privacy, and trust turn the design and development of context-aware mobile applications a challenging task.

A considerable research effort has been done to provide solutions to support the development of context-aware systems. In particular, context management infrastructures to support the development and execution of context-aware applications received significant attention [Baldauf et al. 2007, Bettini et al. 2010]. These infrastructures are typically implemented as a middleware platform or a horizontal framework, providing support to capture, aggregate, store, infer, and delivery context information. Usually, such infrastructure follows a thin client/server architecture style, where the application running on the mobile device (client) performs no or few tasks related to the context management and all or most of the context management processing is performed on the server side. As a drawback, these infrastructures are not adaptable, in a sense they are not able to adapt its own context management mechanism [Da et al. 2011].

6.2.1.1. Solutions

The solutions found in the literature review were classified according to the following taxonomy divided into seven aspects: (i) **Research Subject** - this aspect concerns to the kind of software infrastructure is provided to support development of context-aware application (e.g., framework and middleware); (ii) **Target-Platform** - represents the mobile platform and development technology on which the solution was evaluated/implemented (e.g., Java, Android, Titanium, RESTful); (iii) **Interaction Paradigm** - this aspect captures the kind of interaction paradigm used between applications and the context management infrastructure (e.g., tuple-space, publish/subscribe, and request/response); (iv) **Context Type** - this aspect comprises the type of context each solution supports, following

the Yurur *et al.* [Yurur et al. 2016] classification: device context (e.g., available network interfaces, battery load, CPU and memory usage); physical context (e.g., temperature, noise level, light intensity and traffic conditions) user context (e.g., personal profiles, location and people surrounding them, social situation); and temporal context (e.g., time, day, week, month, season and year); (v) **Dependency** - this aspect indicates if the solution has any software dependence, such as an external framework or library; (vi) **Modularity** - this aspects indicates if the context management concerns implementation are clearly separated from the application business logic; and (vii) **Cloud Interaction** - this aspect concerns to the usage of Cloud Computing concepts to manage context information. Table 6.1 summarized the review and classification according proposed taxonomy.

All papers report solutions that were implemented as a framework (S02, S04, S05, S07, S10, and S12-S15) or as middleware platform (S01, S03, S08, S09, and S11) to manage context information. Only one solution (S06) combines both framework and middleware strategies. Most of them are implemented on top of Android platform (S01-S05, S08, S12-S15). The other three solutions focus on RESTful services (S07 and S09) and cross-platform technology (S10). These solutions implement different coordination models to perform interaction between context management infrastructure and the mobile applications. Most of them support both synchronous request-response interaction (S01, S02, S03, S07-S011) and asynchronous publish-subscribe interaction (S02-S07, S10, S11, S13-S15). Only three solutions provide support for tuple-space interaction model (S06, S08, and S12). All these solutions support different context data, which vary from raw sensor data to personal user information extracted from social networks. Most solutions do not have third-parties dependencies, except S06, S08, S11, and S13 solutions. These four solutions depend on specific implementations of OSGi specification[2]. Once most of the solutions are implemented as a framework or a middleware platform, the context management concerns are well modularized. Finally, regarding the cloud integration aspect, only a few solutions (S10, S12, and S15) use cloud services to manage context information, but none of them mention to have support for offloading techniques.

6.2.1.2. LoCCAM

The LoCCAM (Loosely Coupled Context Acquisition Middleware) [Maia et al. 2013] is a context management infrastructure that adopts the mobile device as the center point of context acquisition and decision. LoCCAM provides a transparent and self-adaptive context data acquisition approach, gathering context information both locally and remotely from the device. It also uses a novel mechanism to support adaptation in the way context information is collected and inferred, following the Viana *et al.*'s definition [Viana et al. 2011] (see Section 6.2.1). An overview of LoCCAM's software architecture is given in the Figure. 6.2. Basically, LoCCAM can be divided into two main parts: (1) the SysSU (System Support for Ubiquity) module; and (2) the CAM (Context Acquisition Manager) framework.

The SysSU [Lima et al. 2011] module provides a coordination mechanism based on tuple spaces and event-based notifications. Such mechanism enhances the coupling

[2]https://www.osgi.org/developer/specifications/

Table 6.1. The Context-Awareness Solutions Survey Summary.

ID	Paper	Research Subject	Target-Platform	Interaction Paradigm	Context Type	Dependency	Modularity	Cloud Interaction
S01	[Mane and Surve 2016]	Middleware	Android	Request/Response	Device Context Physical Context User Context Temporal Context	–	Yes	No
S02	[Da et al. 2014a]	Framework	Android Java SE	Request/Response Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	No
S03	[Da et al. 2014b]	Middleware	Android Java SE	Request/Response Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	No
S04	[Williams and Gray 2014]	Framework	Android	Publish/Subscribe	Device Context	–	Yes	No
S05	[Ferroni et al. 2014]	Framework	Android	Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	No
S06	[Maia et al. 2013]	Framework Middleware	Android	Publish/Subscribe Tuple-Space	Device Context Physical Context User Context Temporal Context	OSGi	Yes	No
S07	[Chihani et al. 2013]	Framework	RESTful	Request/Response Publish/Subscribe	Physical Context	–	Yes	No
S08	[Punjabi et al. 2013]	Middleware	Android	Request/Response Tuple-Space	Device Context Physical Context	OSGi JSON	Yes	No
S09	[Dogdu and Soyer 2013]	Middleware	Java ME RESTful	Request/Response	Physical Context	– –	Yes	No
S10	[Doukas and Antonelli 2013]	Framework	Titanium	Request/Response Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	Yes
S11	[Curiel and Lago 2012]	Middleware	RESTful	Request/Response Publish/Subscribe	Device Context Physical Context User Context	OSGi	Yes	No
S12	[Buthpitiya et al. 2012]	Framework	Android	Tuple-Space	Physical Context	–	Yes	Yes
S13	[Carlson and Schrader 2012]	Framework	Android	Publish/Subscribe	Device Context Physical Context User Context Temporal Context	OSGi	Yes	No
S14	[Lee et al. 2012]	Framework	Android	Publish/Subscribe	Physical Context	–	Yes	No
S15	[Mitchell et al. 2011]	Framework	Android	Publish/Subscribe	Device Context Physical Context User Context Temporal Context	–	Yes	Yes

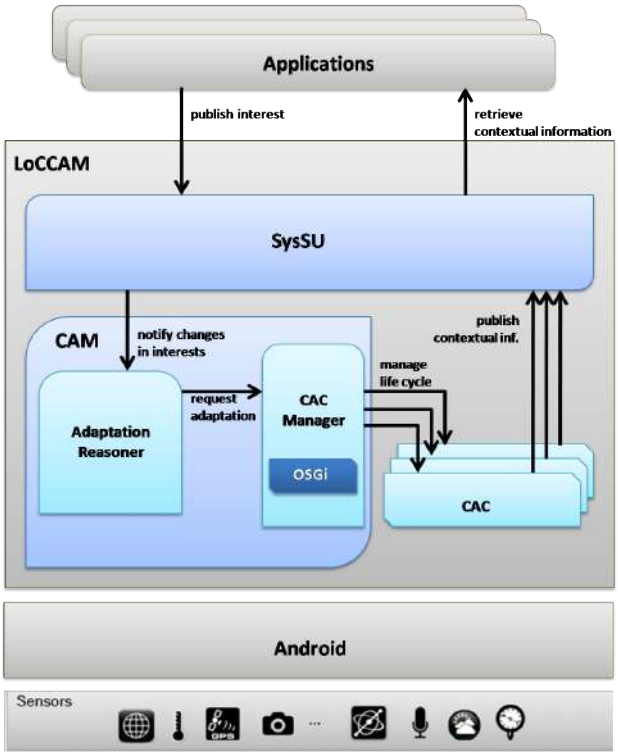


Figure 6.2. The LoCCAM Architecture [Duarte et al. 2015].

between applications and the context acquisition layer. This acquisition layer is based on software-based sensors called CAC (Context Acquisition Component) and can be classified into physical or logical ones. The physical CACs are those that only encapsulate the access to mobile device sensor information (e.g., accelerometer, temperature, and luminosity). On the other hand, the logical CACs are those that can use more than one mobile device sensor and information from other sources (e.g., social networks and weather service on the Internet) to provide high-level context information. Furthermore, LoCCAM offers a common vocabulary that allows CACs and applications exchange context information. Applications subscribe SysSU for the contextual information access by publishing their interests using its common vocabulary. Then, LoCCAM tries to find the more suitable CAC to provide such information. Such communication is based on the concept of Context Keys. They serve as a shared vocabulary to represent each type of contextual information that can be accessed. Each Context Key provides a unique name that must be used by CAC to determine the contextual information it publishes. This key is used by applications to make subscriptions on SysSU. For example, a Context Key that represents the ambient temperature in Celsius scale could be "`context.ambient.temperature.celsius`". To improve the context data selection, applications can use the concept of a filter, which defines a set of fine-grained selection criteria to precisely select the desired context information.

The CAM framework is divided into two modules: Adaptation Reasoner and CAC (Context Acquisition Component) Manager. The Adaptation Reasoner is responsible for maintain the applications subscription list of interest and keep it updated. When any change occurs in this list, the Adaptation Reasoner builds a reconfiguration plan to adapt

the context acquisition layer and sends it to the CAC Manager responsible for performing it. The CAC Manager controls the CAC's lifecycle using a specific implementation of OSGi framework[3]. The CAC Manager can install, uninstall, activate, and deactivate CACs at runtime, according to the application's interest regarding a contextual information access and the reconfiguration plan. A CAC is only activated when an application subscribes SysSU asking for specific contextual information provided by such CAC.

6.2.2. Mobile Cloud Computing

Mobile cloud computing is a new paradigm that incorporates three heterogeneous technologies (mobile computing, cloud computing, and networking) and aims to reduce the limitations of mobile devices by taking advantage of ubiquitous wireless access to local and public cloud resources. Such resources are used to augment mobile devices computing capabilities, conserve local resources, extend storage capacity, and enrich the computing experience of mobile users.

Several authors have also defined mobile cloud computing:

> "*Mobile cloud computing is an integration of cloud computing technology with mobile devices to make mobile devices resource-full in terms of computational power, memory, storage, energy, and context awareness. Mobile cloud computing is the outcome of interdisciplinary approaches comprising mobile computing and cloud computing.*" [Khan et al. 2014]

> "*Mobile cloud computing is a rich mobile computing technology that leverages unified elastic resources of varied clouds and network technologies toward unrestricted functionality, storage, and mobility to serve a multitude of mobile devices anywhere, anytime through the channel of Ethernet or Internet regardless of heterogeneous environments and platforms based on the pay-as-you-use principle.*" [Sanaei et al. 2014]

According to these definitions, the mobile cloud computing model is composed of mobile devices, wireless networks, and remote servers, where mobile devices use wireless technologies to leverage remote servers to execute their compute-intensive tasks or storage data.

Computation offloading is a popular technique to increase performance and reduce the energy consumption of mobile devices by migrating processing or data from mobile devices to other infrastructure, with greater computing power and storage. The concept of offloading, also referred as cyber foraging, appeared in 2001 [Satyanarayanan 2001] and was improved in 2002 [Balan et al. 2002], in order to allow mobile devices to use available computing resources opportunistically.

Migrating computation to another machine is not a new idea. The traditional client-server model is also widely used for this purpose. In fact, the ideas behind the

[3]Apache Felix distribution - http://http://felix.apache.org/

concept of computation offloading date back to the era of dumb terminals that used mainframes for processing. With the adoption of personal computers (e.g., desktops and notebooks), the need to migrate computation has decreased. Nevertheless, with the advent of portable devices, a new need for remote computing power has emerged [Dinh et al. 2011].

According to [Verbelen et al. 2012], offloading can be executed on a remote virtual machine-based environment (e.g., public clouds) or on any machine that is in the same WLAN in which mobile devices are connected to. In the latter case, the remote execution environment is known as cloudlet [Satyanarayanan et al. 2009], and its primary goal is to deliver a better quality of service since Wi-Fi networks are less congested (i.e., they have higher speeds and lower latency) than mobile networks.

It is important to highlight that offloading is different from the traditional client-server model, in which a thin client *always* migrates computation to a remote server. Depending on the availability and the condition of the network, which is highly influenced by users' mobility, offloading may not be possible or advantageous. Therefore, it is important to state that, in mobile cloud computing, the concept of computation offloading is usually implemented by a special program structure, or a design pattern, that enables a piece of code to execute locally on the mobile device or remotely, without impacting on the correctness of the application [Zhang et al. 2012].

In the next subsections we present a further discussion about the types of mobile applications and explore the concept of offloading, by addressing the questions *How?*, *When?*, *Where?* and *Why?* to perform computation offloading.

6.2.2.1. Types of Mobile Applications

According to [Kovachev et al. 2011], applications for mobile devices can be classified as offline, online and hybrid. Offline applications, which are also called native applications, act as fat clients that process the presentation and business logic layer locally on mobile devices, usually with data downloaded from remote servers. Also, they may periodically synchronize data with a remote server, but most resources are available locally, rather than distributed over the network.

Some advantages of native applications are: good integration with features of the device, optimized performance for specific hardware and multitasking, always available capabilities, even without Internet access. On the other hand, the main disadvantages are: they are not portable to other platforms and dependent exclusively on storage and processing power of the device, which may not be enough to execute certain types of computation.

Online applications assume that the connection between mobile devices and remote servers is available most of the time. They are usually based on Web technologies and present a few advantages when compared to offline applications, such as the fact that they are multiplatform and are accessible from anywhere. Nevertheless, they also present some disadvantages: the typical Internet latency can be a problem for some types of applications (e.g., real-time applications), difficulty in dealing with scenarios that require keeping the communication session opened for extended periods of time, and no access

to device's sensors such as camera or GPS.

Hybrid applications are targets of mobile cloud computing and offloading researchers. They can execute locally (as native applications) when there is no connection to remote servers. However, when there is connectivity between mobile devices and remote servers, they can migrate some of the computation to be performed out of the mobile device or access web services (as online applications). The idea is to combine the advantages of online and offline applications.

In the rest of this short-course, we consider only hybrid applications, which are the ones that can benefit from offloading techniques.

6.2.2.2. Where to Perform Offloading?

Mobile devices use remote resources[4] to improve application performance by leveraging offloading techniques. These remote resources are public cloud, cloudlets or another mobile device.

Public Cloud

The execution of services in the public cloud is common among mobile application developers since they can leverage features such as elasticity and connectivity to social networks to improve services. Applications such as Gmail and GoogleDocs are examples of online applications that require smartphones to be connected to the Internet all the time to be able to access data.

In order to connect to the public cloud, mobile devices can use mobile networks (e.g., 2G, 3G e 4G) or a Wi-Fi hotspot. Figure 6.3 illustrates a mobile device accessing an application that relies on Internet connection.

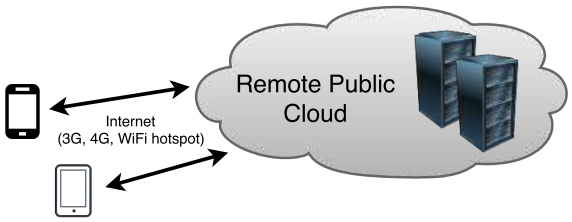


Figure 6.3. Public cloud as remote execution environment.

Cloudlet

The idea of using nearby servers to reduce the connection latency and improve users' quality of experience is being used since the emergence of the term cyber foraging, introduced by [Satyanarayanan 2001]. On that time, the authors used servers in the vicinity to handle the limited capabilities of mobile devices.

The concept of cloudlet is newer and was proposed in [Satyanarayanan et al. 2009]

[4]In this short-course, when we use terms such as remote servers, remote resources or remote execution environments, we refer to any real or virtual remote equipment where the computation of mobile devices can be migrated to.

also to use resources of servers that are close to mobile devices. The difference is that the authors have used virtual machines, in a trusted environment, as remote servers. Today, several studies have used the terms offloading or cyber foraging to indicate that there is a migration of data and/or computation of mobile devices to another location; as well as the terms cloudlet and surrogate that have been used indiscriminately to indicate a computer or cluster of computers directly connected to the same WLAN of mobile devices. Figure 6.4 illustrates the concept of cloudlets by showing mobile devices and remote servers connected to the same wireless network.

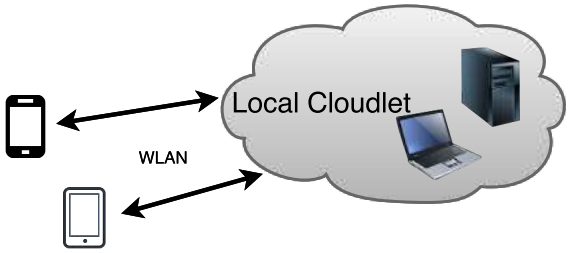


Figure 6.4. Cloudlet as remote execution environment.

The vision of [Satyanarayanan et al. 2009] is that cloudlets would be conventional equipment deployed such as access points, which would be located in public areas (e.g., cafes, pubs, and restaurants) so that mobile devices could connect via Wi-Fi networks and perform offloading without facing high latency and the typical variation of Internet bandwidth.

Other Mobile Device

There is another approach in the literature, commonly referred as "mobile cloud", that considers using other mobile devices as the source of resources, especially to perform a computation. Figure 6.5 illustrates this approach, in which mobile devices are usually connected using a peer-to-peer network and create a cluster (or cloud) of devices.

The idea behind this solution is to enable people, that are in the same place and share the same interests, to create a cloud of mobile devices and share their resources aiming to compute tasks more quickly or reduce energy consumption.

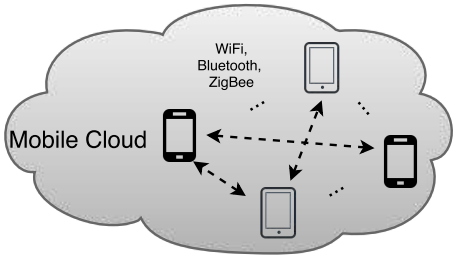


Figure 6.5. Cluster of mobile devices as remote execution environment.

Hybrid Environment

A hybrid environment is composed of two or more of the environments mentioned above. In Figure 6.6, we can see an example of this type of environment, where a mobile

device is part of a "mobile cloud" and can perform offloading by leveraging cloudlets and public cloud as remote execution environments.

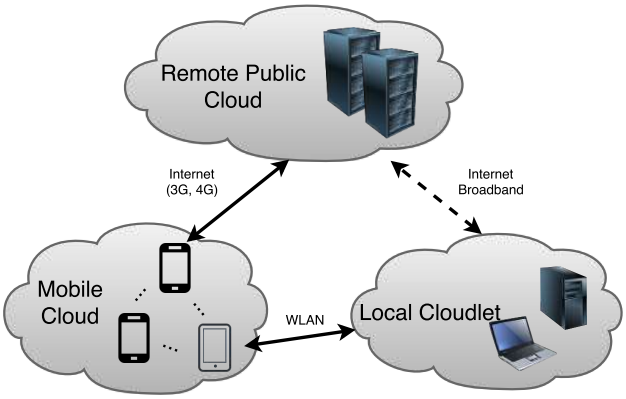


Figure 6.6. Hybrid remote execution environment composed of cloudlets, public cloud and a cluster of mobile devices.

6.2.2.3. Why to Perform Offloading?

Given the limited resources of mobile devices, researchers have mainly used computation offloading to enhance applications' performance, save battery power, and execute applications that are unable to run due to insufficient resources. Therefore, the main reasons for performing offloading are:

- **Improve Performance:** when performance improvement (i.e., reduce the execution time) is the main goal;
- **Save Energy:** when energy efficiency (i.e., reduce energy consumption) is the main goal;
- **Other:** when energy and performance are not the main reasons for performing offloading. Instead, the main reason may be to improve collaboration, extend storage capacity or reduce monetary costs.

6.2.2.4. When to Perform Offloading?

The reasons for performing offloading presented in the previous section are directly related to the decision of when to offload.

[Kumar et al. 2013] present an analytical model to answer the question "When the offloading technique improves the performance of mobile devices?". The model compares the time to process an application task on a mobile device ($\frac{W}{P_m}$) and the time to transfer the data and perform the computation out of the device ($\frac{D_u}{T_u} + \frac{W}{P_c} + \frac{D_d}{T_d}$), either on a public cloud instance or on a cloudlet. The model considers the following parameters: W is the total computation to be performed, which may be expressed in MI (million instructions); P_m and P_c are, respectively, the processing power of the mobile device and the cloud

(or cloudlet), which may be expressed in MIPS (million instructions per second); D_u is the amount of data sent from the device to the cloud (bytes), while D_d is the amount of data received by the device (bytes); and T_u and T_d are the upload and download rate (bytes/second) respectively.

$$\frac{W}{P_m} > \frac{D_u}{T_u} + \frac{W}{P_c} + \frac{D_d}{T_d} \tag{1}$$

Analyzing the model presented in (1), we can see that, to improve performance, the computation should be heavy (high value to W) and the communication between mobile device and the cloud should be brief (low value to $\frac{D_u}{T_u} + \frac{D_d}{T_d}$), either by transferring few data, or by having a high throughput.

[Kumar et al. 2013] highlight that increasing the difference between the processing power of mobile devices and clouds does not bring great impact to the decision. This fact can be observed in (2), when we consider a cloud K times faster than a smartphone ($P_c = K \cdot P_m$). Clearly, for large values of K, Equation (1) can be simplified to $\frac{W}{P_m} > \frac{D_u}{T_u} + \frac{D_d}{T_d}$, which means that the time required to transfer the data between mobile device and cloud has a key role in deciding when to perform offloading. Figure 6.7 depicts the trade-off between the amount of communication and computation for deciding whether or not to perform offloading.

$$\begin{aligned} \frac{W}{P_m} - \frac{W}{K \cdot P_m} > \frac{D_u}{T_u} + \frac{D_d}{T_d} &\Rightarrow \frac{W}{P_m} \cdot \left(\frac{K-1}{K}\right) > \frac{D_u}{T_u} + \frac{D_d}{T_d} \\ &\Rightarrow \frac{W}{P_m} > \frac{D_u}{T_u} + \frac{D_d}{T_d} \end{aligned} \tag{2}$$

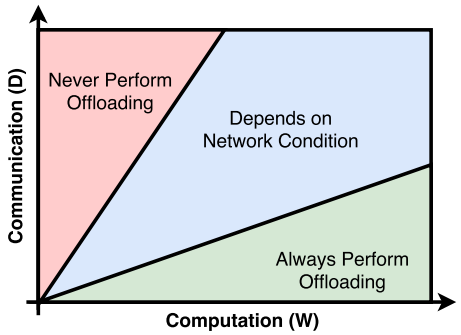


Figure 6.7. Offloading decision trade-off. [Kumar and Lu 2010]

Other researchers focus on different objectives such as throughput maximization [Xia et al. 2013], energy saving [Kharbanda et al. 2012], and cost reduction [Kosta et al. 2012].

Regarding energy savings, [Kumar and Lu 2010] list four basic approaches to save energy and extend the battery lifetime of mobile devices:

1. **Adopt a new generation of semiconductor technology.** Unfortunately, as transistors become smaller, more transistors are needed to provide more functionalities and better performance; as a result, the power consumption actually increases;

2. **Avoid wasting energy.** Whole systems or individual components may enter standby or sleep modes to save power;

3. **Execute programs slowly.** When a processor's clock speed doubles, the power consumption nearly octuples. If the clock speed is reduced by half, the execution time doubles, but only one-quarter of the energy is consumed. Therefore, executing applications more slowly is a good option to save energy. That is indeed done in some smartphone with DVFS to put devices in energy-saving mode. However, it is important to assess the trade-off between performance and energy saving.

4. **Eliminate computation altogether.** A mobile device does not perform computations; instead, the computation is performed somewhere else, thereby extending the battery lifetime of mobile devices.

The last approach can be realized by using offloading techniques to migrate computations from mobile devices to remote servers. We refer to [Magurawalage et al. 2014] for an energy savings analytical model for answering the question "When the offloading technique saves energy of mobile devices?".

Regardless of the reasons that motivate the use of offloading mechanisms, the solutions also differ regarding the **Offloading Decision**, which is related to how an offloading solution decides when to perform offloading. In short, the offloading is called:

1. **Static:** when the developer or system defines prior to execution (at design or installation time) what parts of the application should be offloaded and to where;

2. **Dynamic:** when the framework/system decides at runtime which parts of the application should be offloaded and where to offload, based on metrics related to the current condition of the network, mobile devices, and remote server.

6.2.2.5. What to Offload?

When the application is not available on a remote server, there is a need to migrate parts of (or the whole) application to the server along to the computation request and input data. In order to separate the intensive mobile application components that operate independently in the distributed environment, a partitioning procedure can be used to partition the application at different levels of granularity [Liu et al. 2015].

Partitioning of an application can be done automatically by the offloading system, or it can be provided by developers using a code markup (e.g., annotations on Java programming language). In the latter case, developers add some kind of syntactic metadata to the application source code to identify the components that are candidates to be offloaded. This markup process is usually done by applications' developers at the design phase and involves examining the complexity and dependency of methods.

Several strategies to perform offloading were proposed in the literature, and they differ in relation to which parts of the application are sent to be executed out of mobile devices. Thus, the parts of applications that are most commonly offloaded are as follows:

- **Methods:** when methods are used to partition the application (e.g., remote procedure calls);
- **Components/Modules:** when entire modules or components of an application are executed on another resource execution environment. It involves using specific frameworks for designing and developing modular applications (e.g., OSGi) or just a group of classes that may or may not be coupled;
- **Threads:** when threads of an application are migrated between mobile devices and remote execution environments. It usually involves performing changes on the Android virtual machine (DalvikVM);
- **Whole Application:** when virtualization techniques are used to run clones of mobile devices and the entire state of the application process is migrated and executed out of mobile devices (e.g., in a virtualized clone device). In this case, a synchronization module is usually required for keeping the applications running on mobile device and clone updated, by replicating all changes.

It is common for mobile applications to interact with sensors embedded in mobile devices (e.g., GPS and camera), thus, offloading the whole application may be impractical. That is why [Cuervo et al. 2010] defend that a fine-grain strategy leads to large energy savings as only the parts that benefit from remote executions are offloaded.

6.2.2.6. How to Perform Offloading?

There is no unique answer to the question "How to perform offloading?". In fact, several offloading frameworks/systems/middlewares have been proposed to address such question, and they usually differ regarding *What?*, *Where?*, *When?* and *Why?* to perform offloading.

Existing offloading solutions have applied various strategies and mechanisms to handle steps of the offloading process, such as cloudlet discovery, resources profiling, application partitioning and offloading decision [Sharifi et al. 2012]. Since the strategies used are quite varied, we have performed a literature review that helped us to identify the common approaches used for performing offloading and also have allowed us to create a taxonomy to assist in the classification of related works.

The categories of the taxonomy related to the question "How to perform offloading?" are discussed below:

Method Annotation: when an offloading solution supports granularity of methods and some type of syntax for marking methods is used to identify the methods prone to be offloaded:

1. **Yes:** when syntactic metadata is added to the application source code. The annotation is usually done by developers at design phase, and involves examining the complexity and dependency of methods;

2. **No:** when there is no annotation or when the framework uses a profiler component to collect information and annotate the relevant methods in an automatic fashion.

Decision Module Location: the module responsible for decision-making usually executes compute-intensive operations to decide when and where to perform offloading, which inevitably consumes resources of the mobile device when the module is executed locally. Despite the computation cost reduction when executing the decision module out of a mobile device, this solution imposes more communication costs, which is a clear trade-off that must be considered.

1. **Mobile Device:** when compute-intensive operations of the decision module are executed on mobile devices;
2. **Remote Execution Environment:** when compute-intensive operations of the decision module are executed out of mobile devices.

Decision Module Features: regards approaches and techniques used by a decision module.

1. **Online Profiling:** when the decision module uses measurements (collected at runtime) of different metrics of the environment, network, and application to improve the offloading decision;
2. **Historical Data:** when the solution collects and uses historical data to improve the decision module.

Metrics Used for Decision: regards the metrics considered by a decision module for deciding when and where to offload.

1. **Based on hardware:** when the solution uses metrics related to the hardware of mobile devices and remote servers (e.g., memory, CPU, battery);
2. **Based on software:** when the solution uses metrics related to the software of mobile devices and remote servers (e.g., size of data that will be transferred, execution time, code size or complexity, interdependence between modules or components);
3. **Based on network:** when the solution uses metrics related to network conditions (e.g., connection type, latency, jitter, packet loss, Wi-Fi signal strength, throughput).

Supported Platform/Programming Language: regards mobile platforms and programming languages supported by a solution.

1. **Android**;
2. **Windows Phone**;
3. **iOS**.

Discovery Mechanism: considers whether a solution uses any mechanism for discovering remote execution environments (usually cloudlets or other mobile devices).

1. **Yes:** when a solution automates the remote execution environment discovery;
2. **No:** when the address of a remote execution environment is somehow provided in advance by developers or when a DNS-based service discovery is used.

6.2.2.7. Taxonomy

During the literature review that we have performed, we identified several approaches used in offloading solutions. Then we categorized such approaches in groups based on the questions *What?*, *Why?*, *When?*, *Where?* and *How?* to perform offloading. Inspired by those questions and also in [Sharifi et al. 2012] and [Liu et al. 2015] taxonomies, we propose a taxonomy for offloading solutions.

Figure 6.8 depicts the proposed taxonomy, which is based on the aforementioned questions about offloading. Therefore, *What?* is related to the offloading granularity discussed in Section 6.2.2.5, *Why?* is related to the offloading objective discussed in Section 6.2.2.3, *When?* is related to the offloading decision discussed in Section 6.2.2.4, while *Where?* is related to the remote execution environments discussed in Section 6.2.2.2 and *How?* is related to all categories discussed in Section 6.2.2.6.

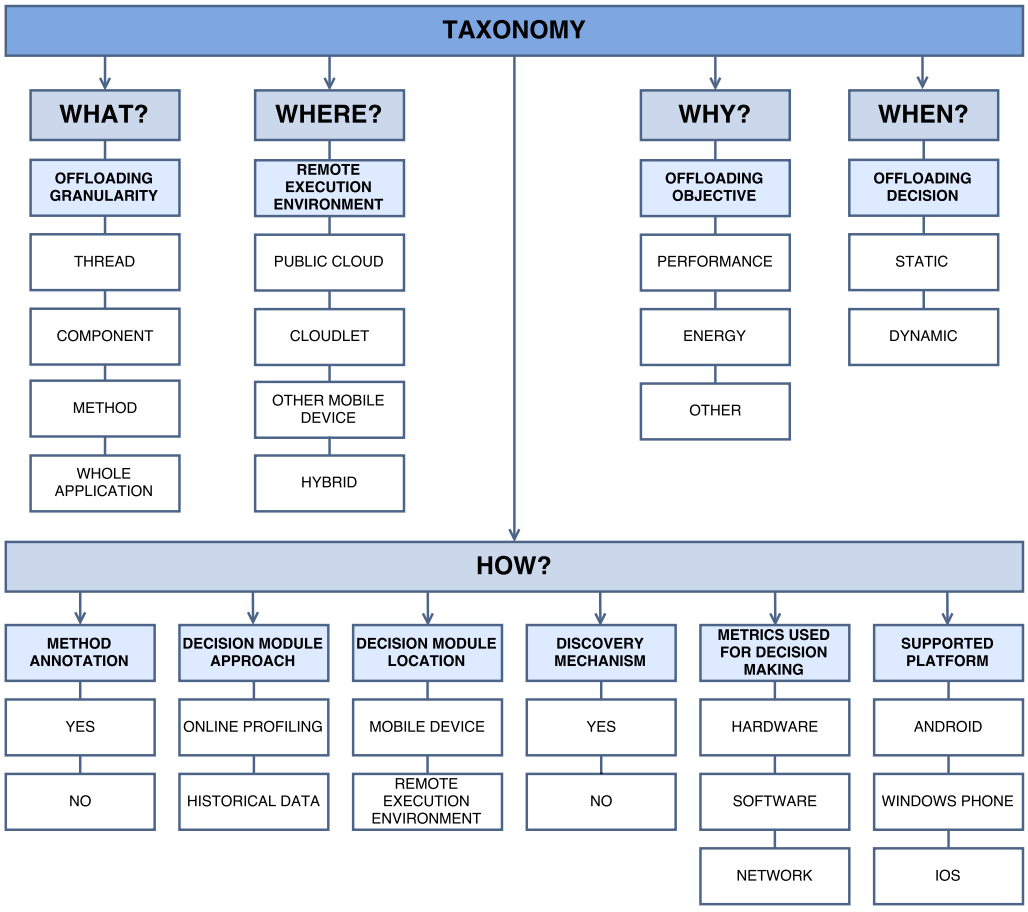


Figure 6.8. Taxonomy for offloading solutions.

6.2.2.8. Solutions

Several solutions have been developed in response to the challenges offered by MCC. In this section, we present some of these solutions, and we use the proposed taxonomy to classify them.

Regarding the offloading granularity, solutions like eXCloud [Ma et al. 2011] and MECCA [Kakadia et al. 2013] perform offloading of the whole application by leveraging virtualization techniques to run a clone of the mobile device. eXCloud uses a compact version of the Java virtual machine, called JamVM, and migrates the state of the application by transferring the Java virtual machine stack from the mobile device to its clone, which is running on the cloud. MECCA supports the Android platform and migrates an entire application process to a clone and allows the user to access the screen of the device using VNC (Virtual Network Computing).

Other solutions like μCloud [March et al. 2011] and MACS [Kovachev et al. 2012] perform offloading of modules/components of the application. Such solutions usually require the use of specific frameworks for designing and developing modular applications (e.g., OSGi and Android Services). For instance, to use MACS, developers have to design their applications using standard Android services, where the compute-intensive components must be implemented as services. Since Android services use inter-process communication channels to perform remote procedure call, MACS intercepts the requests sent to the services and decide whether the request must be executed on local or cloud services.

CloneCloud [Chun et al. 2011] and COMET [Gordon et al. 2012] are examples of frameworks that perform offloading of threads, which usually involves modifying the Android virtual machine to allow the seamless transfer of application's threads. Both solutions require a modified version of the Android operation system and also a clone of the mobile device running on the cloud.

The works that implement offloading of methods usually leverage techniques like Java Annotations to allow developers to identify the methods that are candidates to be offloaded. MAUI [Cuervo et al. 2010] is a framework developed in .NET for Windows Phone 6.5 that uses remote procedure calls to execute methods outside of mobile devices. MAUI uses an approach based on a proxy to intercept methods and redirect them to a server running on the cloud. On the other hand, Scavenger [Kristensen and Bouvin 2010] is a framework developed in Python that uses annotation of methods (using the concept of Python Decorators), and its decision module uses online profiling and historical data analysis to decide whether a method must be executed on mobile device or cloud.

The solutions AIOLOS [Verbelen et al. 2012], ThinkAir [Kosta et al. 2012], and ARC [Ferrari et al. 2016] use methods as offloading units, and they were developed for the Android platform. AIOLOS relies on code refactoring to generate OSGi components for a mobile application at building time. Despite executing OSGi components on the cloud, the decision of when to offload is based on methods provided by the components. On the other hand, ThinkAir relies on Java annotations to identify the methods that can be offloaded and allow such methods to be executed in parallel on multiple servers; while ARC is a framework where a method can be opportunistically offloaded to any available device that can be accessed through the wireless LAN (i.e., other mobile device or a dedi-

cated server, such as in the cloudlet concept). MpOS [Costa et al. 2015][Rego et al. 2016] is a framework that also uses methods as offloading units, but it can be used to develop Android and Windows Phone applications. The framework was developed in our research group and is discussed in more detail in the next section.

Regarding the remote execution environment, most works perform offloading to cloudlets or cloud environments (e.g., eXCloud, MpOS, ThinkAir, μCloud, CloneCloud, AIOLOS, MAUI, Scavenger). ARC and DroidCloudlet [El-Derini et al. 2014] are some of the solutions that use mobile devices as remote execution environments. In DroidCloudlet, the authors propose the creation of a mobile cloud of Android devices that can be used to improve performance and save energy of a device by offloading methods to other devices.

Regarding the offloading decision, μCloud uses static decision, while the other mentioned solutions support dynamic decision. Besides that, the solutions have different offloading objectives. While MpOS, COMET and eXCloud only focus on improving the performance of mobile applications; and Scavenger and μCloud only care about saving energy, ThinkAir, CloneCloud, DroidCloudlet, and AIOLOS consider both objectives (e.g., by solving multi-objective optimization problems).

6.2.2.9. MpOS

MpOS (Multiplatform Offloading System) is a framework for developing mobile applications that support offloading of methods for Android and Windows Phone platforms. MpOS was developed to address the lack of an offloading solution for multiple platforms, and its main goal is to improve performance.

To use MpOS, developers must mark the methods that can be executed out of the mobile device, and they can choose whether the offloading decision will be static or dynamic. If the dynamic offloading is defined, the Offloading System (illustrated in the Figure 6.9) checks if it is worthwhile to perform the offloading operation before sending the method to the remote server. Otherwise, when the static offloading is defined, MpOS verifies only if the remote server is available before performing offloading. In order to decide when to offloading, the decision module uses metrics (e.g., latency, download, and upload rate) that are online collected by the Network Profiler module.

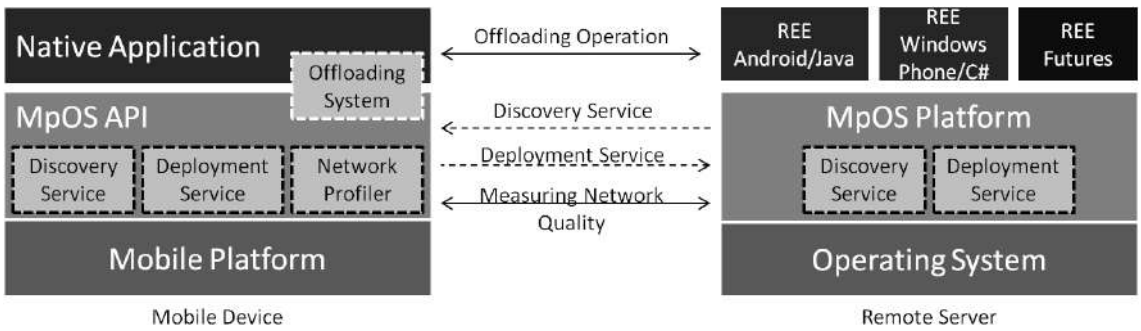


Figure 6.9. Overview of MpOS main components

MpOS has a Discovery Service that uses a multicast-based discovery mechanism

to identify nearby servers (cloudlets) and applications already running on the remote server. If the application is not running on the server, MpOS automatically sends the application and all dependencies to the remote server by means of the Deployment Service. All deployed applications are handled by the Remote Execution Environment (REE) module, which instantiates the applications in different endpoints. In order to support multiple platforms, MpOS provides Android and Windows Phone REEs, which run Java and C# code respectively. Once an application is instantiated on the REE, the mobile device can directly access the application's endpoint and perform offloading. More details about MpOS can be found in [Costa et al. 2015] and [Rego et al. 2016].

6.2.3. Context-Aware and MCC Integration

The advances in mobile computing have made the use of contextual information increasingly present in today's mobile applications. The adaptation of the applications according to contextual situations will improve the users' quality of experience while running the most diverse applications. However, as mentioned earlier in this document, the inference process of the user's contextual situation requires processing and data resources that may sometimes be unsuitable for processing locally on mobile devices.

This issue also occurs when the multimedia data is required to infer users' contextual situation. The use of complex data such as video, audio or images for determining the current context is pointed out as a future trend for mobile applications, but also brings processing requirements that become a problem for mobile computing environments with uncontrolled characteristics due to resource scarcity, low storage capabilities, intermittent connectivity, and power constraints.

In this sense, computing and data offloading fits like a glove to allow the integration of more advanced techniques for contextual inference (such as the use of machine learning algorithms, among others), while also allowing performance gains and energy savings for mobile devices.

It is important to notice that many research projects and reports that address the integration between Context-aware Mobile Computing and Mobile Cloud Computing use "context" as the situation in which a mobile application is running (network condition, remaining battery and/or size of parameters data) to decide whether is worthy to migrate data or processing to the cloud. This document presents a different point of view. We focus on how the management of contextual information may be enhanced (for instance, better or faster inference procedures) with the help of cloud services.

According to [Naqvi et al. 2013], the use of cloud services by context-aware mobile applications is a future trend with no turning back, even with the lack of reports about how this integration should be made. Once adaptation processes are becoming more and more complex, context analysis requires appropriate and robust services.

According to a brief literature review, a few solutions seek to integrate the management of contextual information with concepts of Mobile Cloud Computing, or even just cloud computing.

One of these is the CARMiCLOC [Aguilar et al. 2015], a reflexive middleware architecture based on autonomic computing, which uses cloud services to provide scal-

ability, self-adaptability, integration, and interoperability of context-aware applications. CARMiCLOC uses reflection to inspect its state and ensures a dynamic behavior of its operation. However, the use of cloud resources is restricted only to the storage of the contextual data obtained by sensors. Despite its merit, the cloud potential is underutilized in the proposal, because cloud resources are not used for data processing remotely in the Cloud.

[OSullivan and Grigoras 2016] presents CAMCS (Context Aware Mobile Cloud Services), a mobile cloud middleware solution that has been designed to deliver cloud-based services to mobile users while respecting the goal of providing an integrated user experience of mobile cloud applications and services. CAMCS supports the application partitioning, where some functionalities run on cloud services. Integration with contextual management services uses a component called Context Processor. This component can promote the customization of actions to be taken by the applications, which is performed by other component called Cloud Personal Assistant (CPA). CPA can perform task processing in the cloud on behalf of the user, asynchronously. CAMCS allows the use of historical data and ontologies, which are still prohibitive on mobile devices, due to their intensive processing.

6.2.3.1. Motivating Scenarios

In order to present some examples where context-aware mobile computing and MCC may improve the user experience, this subsection presents some possible scenarios for such integration.

Crowdsensing. Mobile crowdsensing refers to an approach where a large group of people uses mobile devices capable of sensing and computing (such as smartphones, tablet computers, wearables) to collectively share data and extract information to measure, map, analyze, estimate or infer (predict) processes of common interest[Liu et al. 2016]. This situation may enable to detect some contextual situations based on shared data from multiple users. MCC can improve this aggregation and processing of contextual situations based on hierarchies, for example, by aggregating contextual data from users connected to the same cloudlet, and infer some contextual situations in a collaborative way[Xiao et al. 2013][Gomes et al. 2016].

Healthcare and Well-being. With the advancements and increasing deployment of microsensors and low-power wireless communication technologies, the studies conducted on healthcare domain have grown, particularly the studies regarding the recognition of human activities. In this case, information corresponding to human postures (e.g., lying, sitting, standing, etc.) and movements (e.g., walking, running, etc.) can be inferred in order to provide useful feedbacks to the caregiver about a patient's behavior analysis [Yurur et al. 2016]. Recognizing such activities depends on monitoring and analyzing contextual data such as vital sign (e.g., heart rate, pressure level, and respiration rate), which might be aggregated by mobile devices. MCC can improve the execution and save energy of mobile devices when offloading the compute intensive operations required to recognize complex events.

Augmented Reality. A typical mobile augmented reality system comprises mo-

bile computing platforms, software frameworks, detection and tracking support, display, wireless communication, and data management [Chatzopoulos et al. 2017]. Contextual data is important for providing information about the user's environment, which might be used to present personalized content and improve user's experience. The problem is that vision-based applications are almost impossible to run on wearables and very difficult on smartphones since they require capable GPUs [Pulli et al. 2012]. MCC can, therefore, be used to offload the execution of heavy computations to a powerful remote device.

6.3. Context-Aware and Offloading System (CAOS)

CAOS is a software platform that assists developers to create context-aware applications for the Android platform and provides offloading mechanisms to delegate the migration of methods and contextual data from mobile devices into cloud platforms. CAOS is based on two solutions: (1) LoCCAM (Loosely Coupled Context Acquisition Middleware) [Maia et al. 2013], a middleware that helps context-aware applications developers to get information from sensors, as well as to separate questions related to contextual information acquisition from the applications business logic; and (2) MPoS (Multiplatform Offloading System) [Costa et al. 2015][Rego et al. 2016], a service-oriented architecture that enables developers to mark methods on their applications using annotations, in order to identify which methods can be transferred to cloud servers instead of being executed on the mobile device.

In order to conceive CAOS, we surveyed frameworks and middlewares that support both context-aware and offloading features. This survey provides us an insight of good design decisions that were used to build CAOS. According to these design principles, we designed a software architecture that supports both method and data offloading into cloud infrastructures. We implemented a prototype of CAOS, and we conducted experiments to evaluate its impact on performance and energy consumption of Android applications [Gomes et al. 2017].

Before showing how to use CAOS to develop context-aware Android applications, the next section introduces CAOS architecture and main components.

6.3.1. Architecture and Components

CAOS rests on a traditional client/server architecture, where mobile devices act as clients for services running on the top of cloud/cloudlets infrastructures. Figure 6.10 shows the CAOS architecture where its main components are divided into two groups: mobile side and cloud side components.

6.3.1.1. The Mobile Side

CAOS mobile side is composed by 10 (ten) components: CAOS (which is responsible for synchronizing the startup of other components of the mobile side), *Offloading Monitor*, *Offloading Client*, *Discovery Client*, *Authentication Client*, *Profile Monitor*, *Offloading Reasoner Client*, *Context-Acquisition Middleware*, *Tuple-Space Based Context-Bus* and *Context Synchronizer*. Each one of these components is detailed as follows.

The *Discovery Client* component uses a mechanism based on UDP/Multicast com-

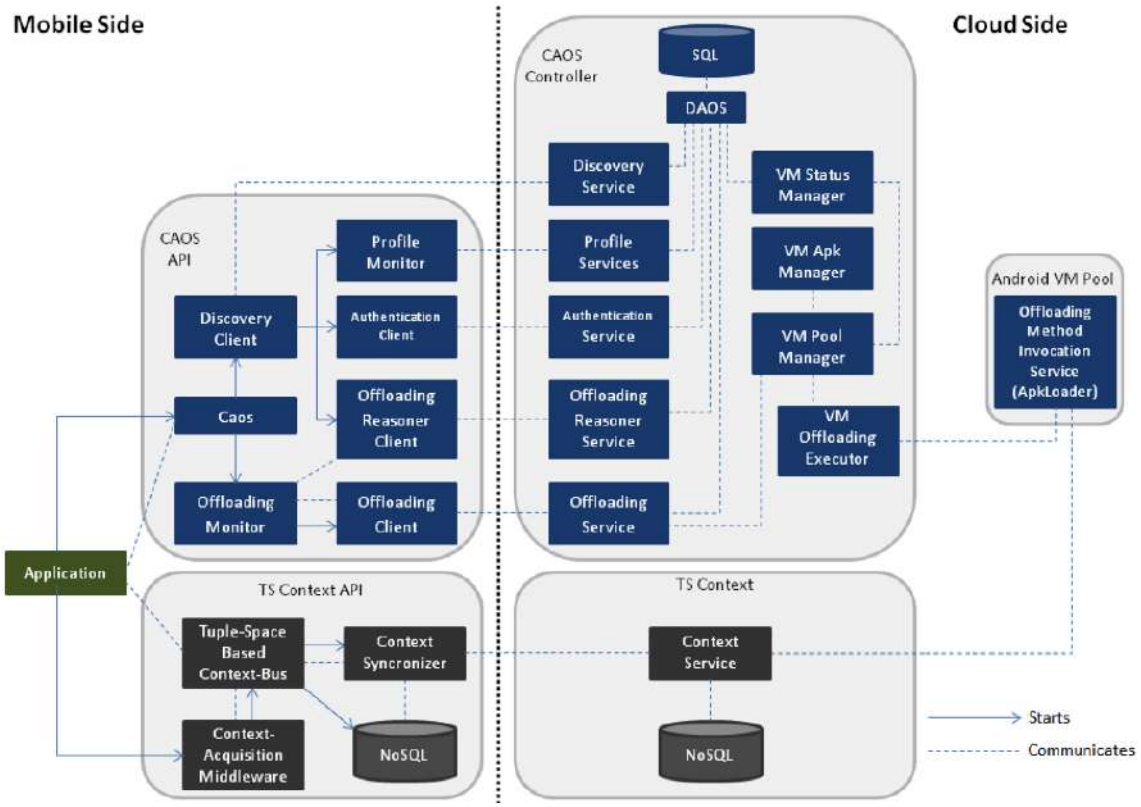


Figure 6.10. Overview of CAOS main components

munication to discover CAOS Controllers running in the user's local network (i.e., cloudlets). Once the *Discovery Client* detects a CAOS Controller, the *Authentication Client* authenticates the mobile application and sends device data to the cloud side to keep the list of devices attached to a specific CAOS Controller.

In CAOS, application's methods can be marked by developers with a Java annotation @*Offloadable*, which denotes that those methods can be executed out of the mobile device. The *Offloading Monitor* is the components responsible for monitoring the application execution and intercepting the execution flow whenever an annotated method is called. After intercepting the method call, the *Offloading Monitor* asks the *Offloading Reasoner Client* module whether it is possible to start the offloading process or not.

The *Offloading Reasoner Client* assists the offloading decision using a decision data-structure that is asynchronously received from the *Offloading Reasoner Service*. In CAOS, the decision whether is worthy or not to offload a method is performed in two steps: one at the cloud side, and another on the mobile side. The cloud side keeps receiving profiling data from each mobile device connected to its infrastructure and creates a two-class decision tree [Rego et al. 2017] with the parameters that should be considered to decide if it is worth to offload a method, such as latency, parameters types, and so on. Once the decision tree is created or updated, it is sent to the mobile device, so the *Offloading Reasoner Client* only has to enforce the decision based on current values of the monitored data and the decision tree structure.

Figure 6.11 presents an example of offloading decision tree for a dummy applica-

tion. In this example, if the upload rate is equal to 200 Kbps and RTT is equal to 100 ms, the method `Bar` must be executed locally. Moreover, in all cases the method `Foo` must be executed on the remote server.

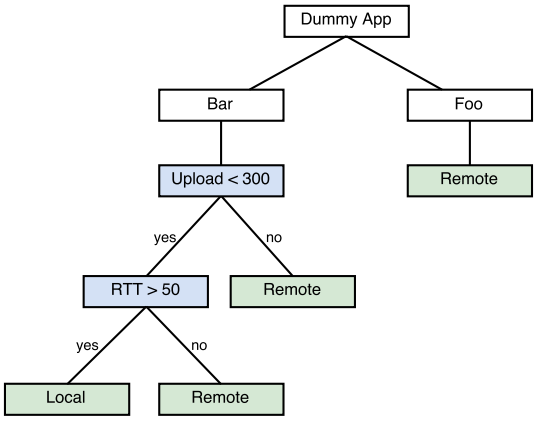


Figure 6.11. Example of offloading decision tree. [Rego et al. 2017]

By using the received data structure, the *Offloading Monitor* can decide locally when it is worth performing offloading. If the answer is negative, the method execution flow is resumed, and its execution is performed locally. Otherwise, when the answer is positive, the *Offloading Monitor* requests the *Offloading Client* to start the method offloading process, which in turn transfers the method and its parameters to the *Offloading Service* in the cloud side.

The *Profile Monitor* component is responsible for monitoring the mobile device environment (e.g., network bandwidth and latency, power and memory status) and sends such information periodically to the *Profile Services*, which will be used in the *Offloading Reasoner Service* (cloud side), that generates the offloading decision tree based on the mobile device information and then sends it back to the *Offloading Reasoner Client*.

The *Context-Acquisition Middleware* component is a new version of the former CAM component in the LoCCAM framework, which has been adapted for this project. The original component has been extended to provide a better integration with cloud/-cloudlets. A new optimized manager was built from the scratch to control CAC's lifecycle, removing OSGi dependency.

This component manages the software-based sensors lifecycle (i.e., search, deploy, start and stop) that encapsulates how context information is acquired. These sensors are called *Context Acquisition Component - CAC* and can be classified into physical or logical ones. The physical CACs are those that only encapsulate the access to mobile device sensor information (e.g., accelerometer, temperature, and luminosity). On the other hand, the logical CACs are those that can use more than one mobile device sensor and information from other sources (e.g., social networks and weather service on the Internet) to provide high-level context information. This middleware also provides an API to build new CACs and incorporate them into new and existing applications.

All context information acquired by CACs is stored in the *Tuple-Space Based Context-Bus module*, which is a new version of the former SysSU module. The *Tuple-*

Space Based Context-Bus stores the context information in a tuple-based format and deliver such information to applications using event notification via a contextual event-bus. This new version has a new feature, the Tuple-Space Based Context-Bus that stores more than one tuple for a particular sensor, creating a list of samples. The initial version of SysSU stored only the last collected sample, overlapping the previous one. This feature may improve inferences using a larger amount of contextual data.

All context information of each mobile device connected to a CAOS Controller is sent to the Context Service (cloud side) to keep a database of contextual information history. The idea is to explore the global context (i.e., the context of all mobile devices) to provide more accurate and rich context information. The *Context Synchronizer* exchanges contextual data between the mobile and the cloud sides. This data migration is important because, in CAOS, filters can be performed in both local context information repository (*Tuple-Space Based Context-Bus*) and global context repository (*Context Service*). If an application has an @*Offloadable* marked method that accesses context information using the filter concept; it can benefit itself from the offloading process to access the global context repository. This can be done using two filters: one to be executed locally (on the mobile device) and another that runs in the global context repository when the method is offloaded to the cloud. The decision of which filter will be executed is performed automatically by CAOS.

6.3.1.2. The Cloud Side

The CAOS cloud side is composed by 11 (eleven) components: *Discovery Service*, *Profile Services*, *Authentication Service*, *Offloading Reasoner Service*, *Offloading Service*, *VM Pool Manager*, *VM Apk Manager*, *VM Status Manager*, *VM Offloading Executor*, *Context Service*, and *Offloading Method Invocation Service*.

The *Discovery Service* provides correct endpoints information so that clients can access CAOS services. The *Authentication Service* is responsible for saving device information and controlling which devices are currently connected to the CAOS Controller. The *Profile Services* is a set of services which receives device monitored data related to the network quality and local execution time of offloadable methods, in order to keep historical records of executed methods. These records will be used during the offloading decision tree creation process.

The *Offloading Reasoner Service* is responsible for processing the mobile device profile data and creating the decision tree that will be used by the *Offloading Reasoner Client* component (on the mobile side) to decide about the offloading process execution. The *Offloading Service* receives offloading requests directly from *Offloading Client* and redirects them to the *VM Pool Manager*. When the offloading process finishes, the *Offloading Service* returns the result to the *Offloading Client* and persists offloading information.

In CAOS, offloaded methods are executed on Android Virtual Machines running on traditional x86 machines. The *VM Pool Manager* component is responsible for providing an environment that redirects offloading requests to a proper Android Virtual Machine where the offloading execution happens. In order to run a method from a specific applica-

tion, CAOS requires that the corresponding deployment packages (a.k.a. APK) of CAOS compliant applications must be stored in a special folder on the CAOS platform. The *VM Apk manager* pushes all APK files to all reachable Android Virtual Machines listed in this special folder. The *VM Status Manager* is responsible for monitoring and maintain information (e.g., the current number of offloading executions) about each virtual machine managed by the VM Pool Manager in a repository. The *VM Offloading Executor* component is responsible for requesting the offloading execution in an Android Virtual Machines calling the *Offloading Method Invocation Service*, which runs on the virtual machine and performs the offloading method execution.

The *Context Service* acts like a global context repository, and stores all context information data sent from all mobile devices connected to the CAOS Services. It maintains this context information in a NoSQL database and provides a proper interface that can be used by logical CACs to access this information to generate high-level context information for an application running on the mobile device.

Related to our architecture, it is possible to store the private contextual data in special cloudlet, called *Gateway*. In short, a Gateway is a cloudlet with privacy policies where its data are chosen by each user to be sent to Cloud or not. This approach filters all information that the user does not want to share with other users, but wants to use for personal purposes. This information is stored in a gateway, but not sent to the cloud.

6.3.1.3. Implementation Details

The technologies employed in the CAOS reference implementation are described in the following. In the mobile side, the CAOS modules were implemented using the Android SDK on Android Studio IDE. The Couchbase (a NoSQL database) was adopted to maintain a local cache of context information history to be offload to a remote side following the CAOS context data offloading strategy. The Bouncy Castle encryption API was chosen to improve the CAOS wireless communication security. The JCoAP (Java Constrained Application Protocol) was adopted as a protocol to achieve the standardization level requested by the project sponsor.

In the cloud/cloudlet side, the CAOS Controller modules were implemented using Java SE/EE and deployed on Apache TomCat web container. All server side services were implemented as RESTful Web services using the Jersey framework. All data in CAOS Controller are stored in a PostgreSQL database using the Hibernate framework. The Context Service stores the context tuples on a MongoDB (a NoSQL database). The VirtualBox hypervisor technology was employed in the Android environment virtualization in the server side using an ISO Android x86 version (4.4.2) Kitkat.

6.3.2. CAOS Study Case

This section presents the steps required to develop a context-aware multimedia Android application with offloading capability, called MyPhotos. Such application allows users to apply filters to their photos and share them on social networks along with *hashtags*, to make it easier to search for them. Also, photos can be tagged with contextual information, such as location, date and time at which they were captured. MyPhotos uses CAOS to

offload methods related to image filters execution and contextual data into CAOS servers. The CAOS server will then enable MyPhotos to recommend *hashtags* for photos with similar context, by leveraging contextual data offloaded by other users.

MyPhotos has two image filters that can be applied to a photo: Simple and Complex. The former applies a RedTone filter. The latter is more compute intensive than Simple because produces a cartoonized version of the image, where the new photo looks like a pencil sketch, as depicted in Figure 6.21. MyPhotos's main screen also shows the time elapsed for the last filter execution.

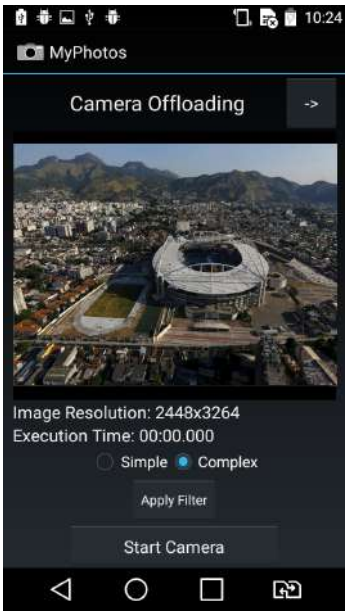


Figure 6.12. MyPhotos main screen

6.3.2.1. Preparing CACs for MyPhotos

The first step in the MyPhotos development process it to implement the CACs that provide the contextual data needed for the application. Two CACs need to be developed: the first CAC provides the location of the mobile device, while the second encapsulates a logical sensor. This second CAC uses the contextual data from the first CAC and provides two values: the hashtags produced by the user and the location/time at which they were produced.

Figure 6.13 illustrates the steps required to create a CAC.

Firstly, we need to configure the development environment in a Linux operating system, by downloading a *shell script* that was developed to create the main structure of a CAC project. This *script* generates a Maven [5] project, which can be edited in any development environment, and has the required settings for a CAC and all methods that need to be implemented.

[5]https://maven.apache.org

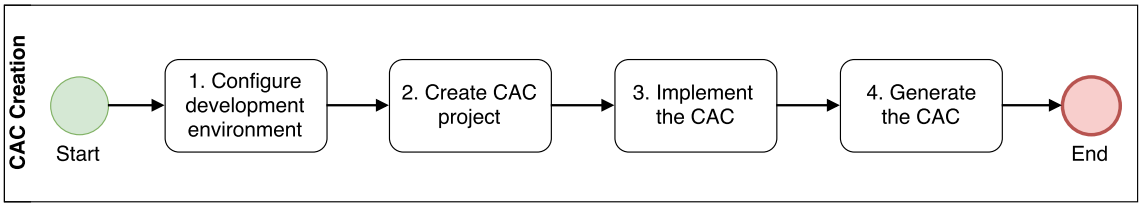


Figure 6.13. Overview of the CAC creation process

The *script* is called `mavenbuild.sh`. Ideally, it should be in the *PATH* setting of the operating system, so that developers can create CACs easily in any folder. The *script* requires two parameters: the CAC name and the project package.

The second step is to run the *script* and create the CAC project. Figure 6.14 shows an example of how to use the *script* to create the first CAC.

Figure 6.14. Example of how to create a CAC project.

Figure 6.15 shows a tree view of the CAC project folder. We can see that there is a file called `pom.xml`, which is where Maven project developers should place the project dependencies and other parameters (e.g., the project version). Two folders, `main` and `test`, have been created, and a package with the name that was passed as a parameter when launching the *script*. In the `main` folder, the CAC main class has the same name as the project, and it is in such file that the CAC must be configured and its required methods must be implemented. In the `test` folder, the main class is used to perform unit tests in the project.

In CAOS, once the CAC project is created, the developer must write the necessary codes (step 3). To configure a CAC, it is necessary to configure some parameters that will be used by CAOS. The main parameter of the CAC is the *ContextKey* (CK), which informs the type of contextual data that it provides. For the first CAC, CK was defined as "context.device.location". This CAC provides contextual data about latitude and longitude of the mobile device's location. For the second CAC, CK was defined as "context.ambient.hashtags".

The hashtags CAC is designed so that whenever the user of the MyPhotos application enters with *hashtags*, they produce contextual data containing the location of the mobile device and *hashtags*. Thus, the *hashtags* CAC performs a subscript of the produced *hashtags*, reads the device's location and provides a new contextual data with both information.

The last step is to generate the CAC, which consists of generating a JAR file containing all class files. This JAR file must be sent to a specific folder on the mobile device so that the CAC begins to provide contextual data according to applications' interest (see

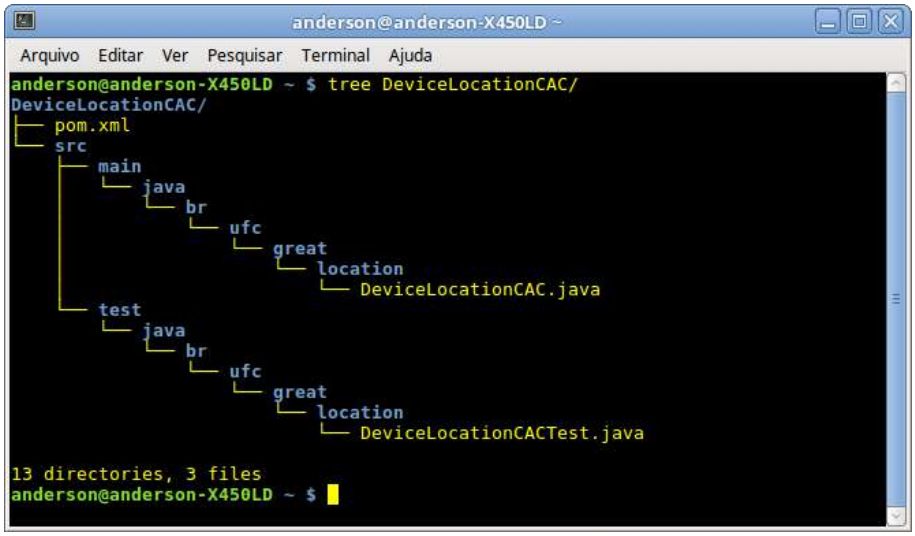


Figure 6.15. Tree view of the CAC project folder.

Figure 6.16).

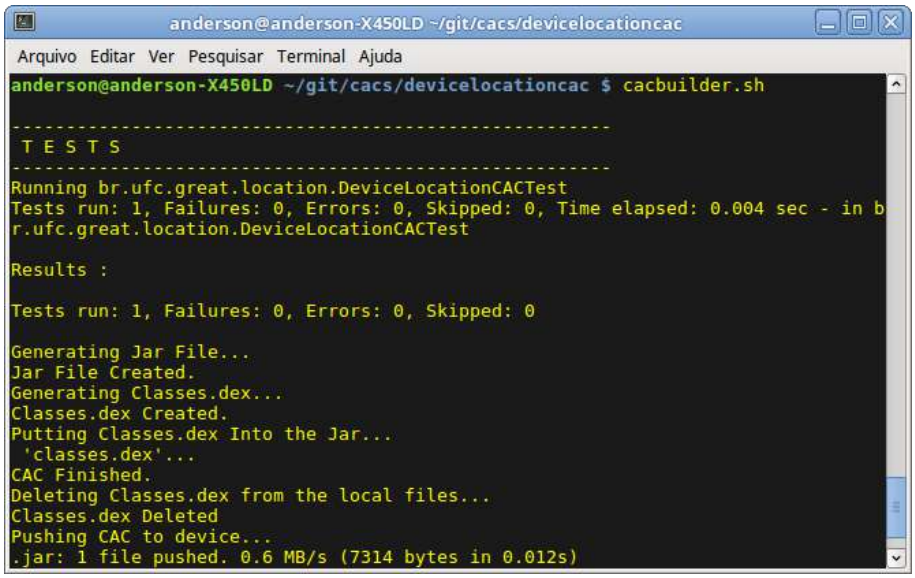


Figure 6.16. Generate the CAC

6.3.2.2. Developing MyPhotos

Once the CACs are built, we can implement MyPhotos. However, due to space constraints, we cannot show all steps and codes required for developing the application. Thus, we will consider that the basic code is already developed[6] and that we still need to configure MyPhotos, so it is able to perform offloading using CAOS. Figure 6.17 illustrates all steps a developer must follow to configure MyPhotos (or any application) to use CAOS.

[6]MyPhotos source code is available at: http://caos.great.ufc.br.

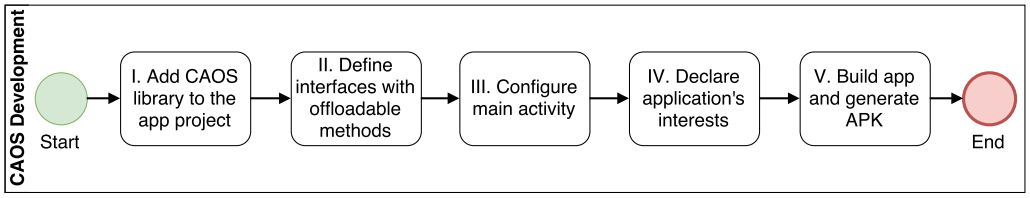


Figure 6.17. Overview of the CAOS development process

The first step (I) is to add the CAOS library to the set of libraries of the Android application project. The library contains CAOS API and TS Context API, which will be used at build time.

After including the CAOS library to the set of libraries of the MyPhotos project, we need to mark the methods that can be offloaded with `@Offloadable` annotation (step II). The `Effect` interface has two methods to apply different image effects to pictures. The methods `simple` and `complex` (respectively, lines 3 and 6 in Listing 6.1) are marked with the offloading markup (respectively, lines 2 and 5 in Listing 6.1), so the Offloading Monitor component will intercept the execution flow of these methods and decide whether the methods must be executed locally or out of the mobile device.

```
1  public interface Effect {
2      @Offloadable
3      public byte[] simple(byte source[]);
4
5      @Offloadable
6      public byte[] complex(byte source[]);
7      ...
8  }
9  public class ImageEffect implements Effect {
10     public byte[] simple(byte source[]){ ... }
11     public byte[] complex(byte source[]){ ... }
12 }
```

Listing 6.1. MyPhotos Interface Effect Markup.

The markup process also has to be performed for the interface responsible for offloading contextual data. The `Read` interface contains the method that retrieves data, and the concrete class `ReadHashtags` implements this interface and all processing related to the data.

```
1  public interface Read {
2      @Offloadable
3      public String readHashTags(double latitude, double longitude,
           double intervalTime, double intervalLocation);
4  }
5
6  public class ReadHashtags implements Read {
7      public String readHashTags(double latitude, double longitude,
           double intervalTime, double intervalLocation){ ... }
8  }
```

Listing 6.2. MyPhotos Interface Read Markup.

In the next step (III), we need to mark the main class of the MyPhotos application (`MyPhotosActivity.java`) with the `@Caos` markup (e.g., Listing 6.3, line 1) and we configure CAOS services to start and stop along with the application. We start

CAOS in the `onCreate` method and stop it in the `onDestroy` method, as illustrated in Listing 6.3, lines 16 and 24 respectively.

It is also necessary to declare the interests of the application regarding contextual data (step IV). Thus, we define MyPhotos interest in Location in Listing 6.3 line 31, while in line 32 we define the interest in Hashtags.

```
1  @Caos
2  public class MyPhotosMainActivity extends Activity implements CaosContextListener {
3
4      @Inject(ImageEffect.class)
5      private Effect effect;
6      @Inject(ReadHashtags.class)
7      private Read read;
8      ...
9      String CK_TAGS = "context.ambient.noise";
10     String CK_LOCATION = "context.device.location";
11     ...
12     @Override
13     protected void onCreate(Bundle bundle) {
14         ...
15         Caos caos = Caos.getConfig().getInstance();
16         caos.start(this, this);
17         ...
18     }
19     ...
20     @Override
21     protected void onDestroy(Bundle bundle) {
22         ...
23         Caos caos = Caos.getConfig().getInstance();
24         caos.stop(this);
25         ...
26     }
27     ...
28     @Override
29     public void onServiceConnected(ISysSUService service) {
30         try {
31             caos.putInterest(CK_LOCATION);
32             caos.putInterest(CK_TAGS);
33         } catch (RemoteException e) {
34             e.printStackTrace();
35         }
36     }
37 }
```

Listing 6.3. Application initialization code for Android.

Finally, when all steps are done, we can build the MyPhotos project and generate the APK file (V). Then, the APK can be installed in the mobile device and also sent to the CAOS server, in order to support offloading of MyPhotos methods (such step will be explained in the next section).

6.3.2.3. Configuring CAOS Services

In this section, we present how to configure CAOS Services on mobile and cloud sides.

On mobile side

In order to run CAOS on the mobile side, we need to the install the application CAOS. This application acts as an service, in which contextual data synchronization policies are configured as well as the privacy settings of such data. Figure 6.18 shows screen-

shots of the CAOS application.

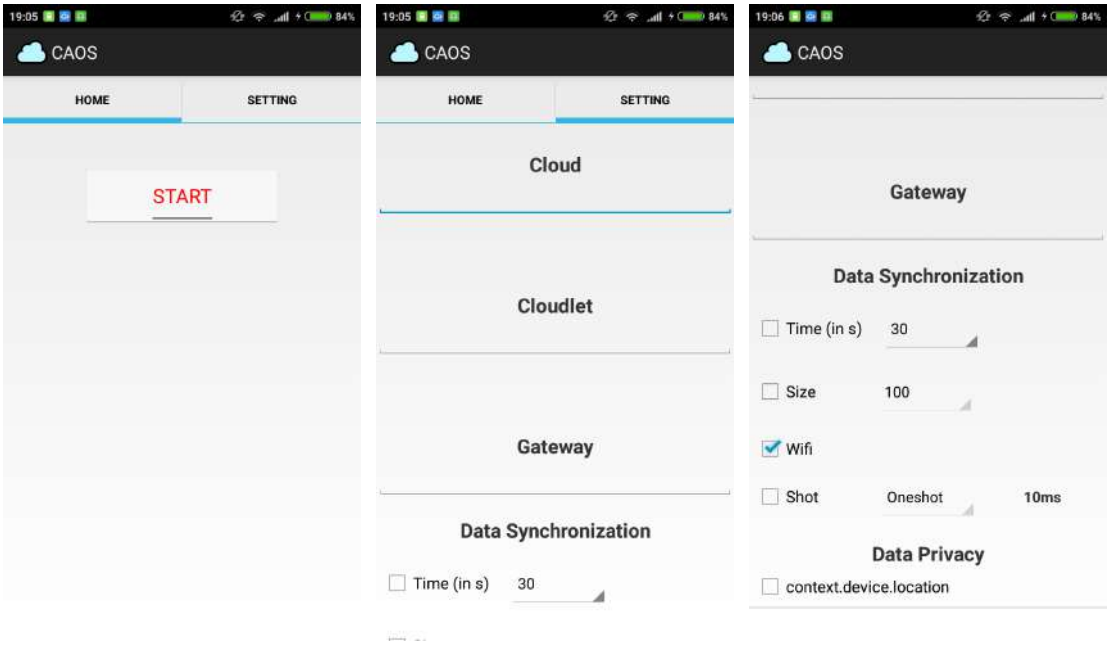


Figure 6.18. Screenshots of the CAOS application running on mobile device

The JAR file of each CAC must be sent to a specific mobile device folder, so the CAOS application can manage the CACs and control their use. In sum, all CACs stored in the `/storage/emulated/0/Android/data/br.ufc.great.caos/files/components` folder of the mobile device are loaded and their privacy setting are managed by CAOS. The user can then use the application to configure the privacy level of each CAC.

On cloud side

In order to run CAOS Services, we need to the install Android SDK, MongoDB, TomCat server, PostgreSQL server, and VirtualBox in a machine running any operating system. After installing the Android SDK, we need to include the ADB (Android Debug Bridge) tool into the PATH variable of the system. Thus, the CAOS controller can manage the pool of Android x86 virtual machines using ADB.

After installing the mentioned applications, the next step is to install the Android x86 operating system in a virtual machine with at least 512 Mb of RAM and 1 GB hard disk. Android x86 is a project that aims to port the Android open source project to the x86 platform. In their website, we can find ISO images for different Android versions, but we recommend to use the 4.4.3 version. Once the Android x86 is installed in a virtual machine, we need to install the ApkLoader application on it, in order to run the Offloading Method Invocation Service.

We need to configure a few properties before executing the CAOS controller. First, the database credentials have to be included in the `persistence.xml` file (as illustrated in Listing 6.4). We need to set IP, database name, username and password in the file according to the PostgreSQL configuration.

```
<persistence ... version="2.0">
  <persistence-unit name="caos" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
      <properties>
        <property name="javax.persistence.jdbc.url" value="jdbc:postgresql://<IP>/<DB>"/>
        <property name="javax.persistence.jdbc.driver" value="org.postgresql.Driver"/>
        <property name="javax.persistence.jdbc.user" value="<USER>"/>
        <property name="javax.persistence.jdbc.password" value="<PASS>"/>
        ...
      </properties>
  </persistence-unit>
</persistence>
```

Listing 6.4. Example of the persistence.xml file.

After that, we need to configure the `vms` file, which is located in the properties folder. In this file, we have to include the endpoint of the ADB server running on the Android x86 virtual machines. As by default ADB listens to the port 5555, the vms file should look like the Listing 6.5. In the example, we are setting two Android x86 virtual machines with IPs 192.168.56.110 and 192.168.56.120 to our VMs pool.

```
192.168.56.110:5555
192.168.56.120:5555
```

Listing 6.5. Example of the vms file.

The last file we need to edit is the `net.properties`, which is also located in the properties folder. In this file, we have to set the network interface of the machine that will listen for discovery messages, as well as the ports that will be used by several services. We recommend to use all default ports and change only the network interface name. The Listing 6.6 presents an example of the `net.properties` file. In the example, we use the wireless interface (`wlan0`) to listen for mobile devices discovery requests.

```
prop.server.networkInterfaceName=wlan0
prop.server.port.authentication=8300
...
prop.server.port.discovery.reply=31002
```

Listing 6.6. Example of the net.properties file.

Before launching the CAOS Controller, we need to put in the `apks` folder a copy of all applications (APK files) that we want to support offloading. Figure 6.19 shows the `apks` folder with several APK files.

When running, the CAOS Controller will distribute the files to the Android x86 virtual machines that will run the offloading requests. Figure 6.20 shows how to run the CAOS controller.

6.3.2.4. Running MyPhotos

MyPhotos allows users to apply filters to their photos, use *hashtags*, share and tag them with contextual information. Users can use two image filters to apply effects to photos: Simple and Complex. The Figure 6.21(a) shows the cartoonized version of the Maracanã Stadium image after applying the filter Complex.

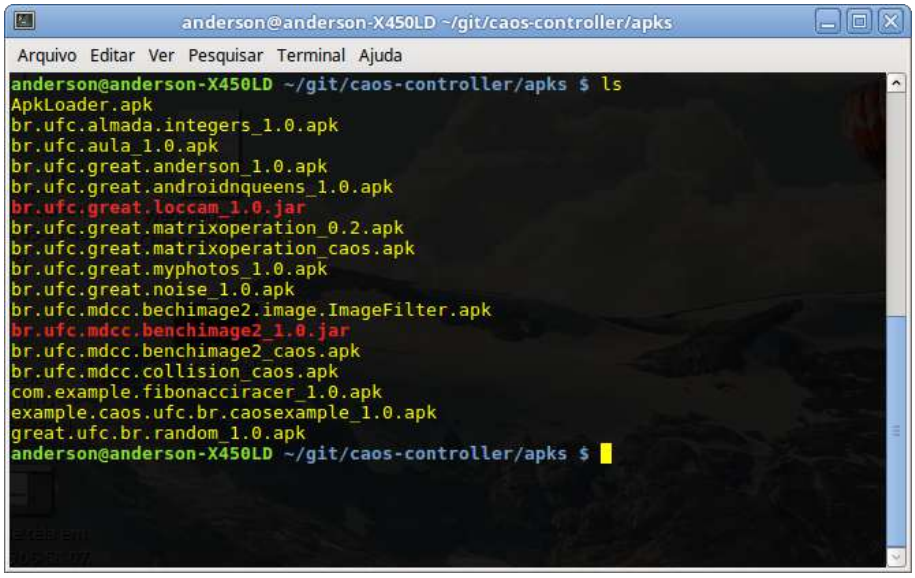


Figure 6.19. APKs Folder

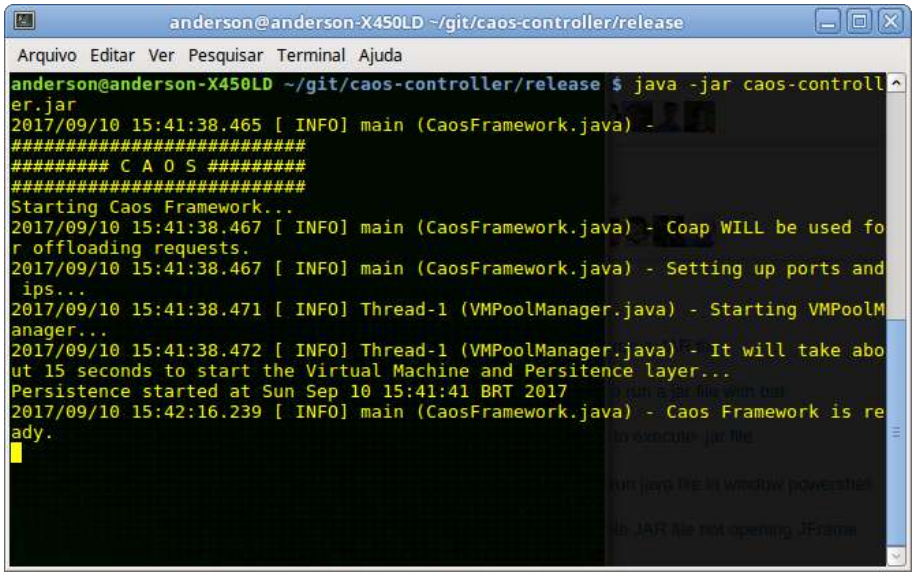
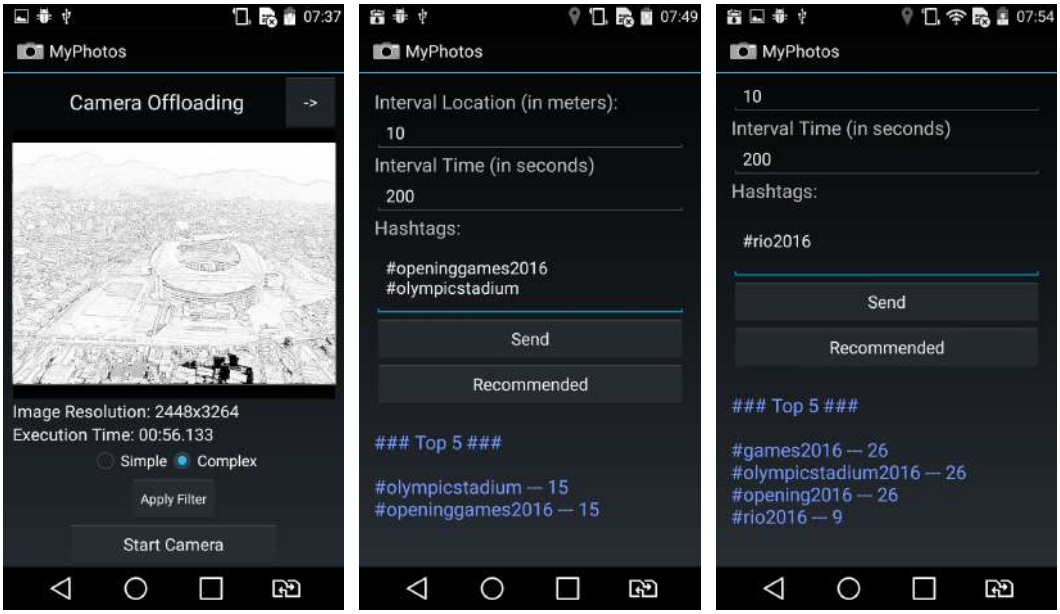


Figure 6.20. Run the CAOS Controller

In Figure 6.21(b), the settings available for the use of *hashtags* are presented. Two parameters are required: the location range (in meters) and the time interval (in seconds). Thus, the user defines how far and how late these data are according to his/her geographical position and time. The "Recommended" button retrieves *hashtags* used previously on images with the same context of the current photo. The "Send" button publishes the image.

First, MyPhotos was executed with the device disconnected from the network, which forces the image filter and tags recommendation to run locally. Figure 6.21(a) shows that it takes more than 56 seconds to apply the filter Complex.



(a) Applying the filter Complex (b) Recommended *hashtags* when running locally (c) Recommended *hashtags* when connected to the CAOS server

Figure 6.21. MyPhotos Screenshots

The Figure 6.21(b) shows that when the mobile device is not connected to the CAOS server, it only retrieves *hashtags* stored in the local Tuple-Space Based Context Bus. Since the user labeled the photo with two *hashtags*, #olympicstadium and #openinggames2016, these two tags are also the only *hashtags* that could be recommended to the user. In Figure 6.21(c), the device is connected to the CAOS server, and when the *hashtags* recommendation is requested, the application retrieves up to the five most popular *hashtags* with a context similar to the current user.

6.4. Trends and Research Challenges

Although the combination of context-aware Computing and MCC has great potential for future mobile applications, it is also important to understand that there are issues that raise new challenges to enable such integration. Some of these issues are mentioned in this section.

6.4.1. Costs for raw data transfer to compute the context inference on cloud services

For some scenarios, the amount of data required to process a contextual situation may be a barrier for the use of MCC. Sometimes, multiple sources of information are needed to derive a Context information, i.e., a multi-sensory process. This is a common scenario for data-intensive applications, in which some cases of context sensing may fit. [Naqvi et al. 2013] cites that step counting or falling detection algorithms from sensor samples such as accelerometer and gyroscope may require a reasonable amount of data to produce good results. However, transferring a huge amount of data very frequently may bring side effects related to energy consumption or charging fees while using mobile operator network.

6.4.2. High Latency between mobile devices and cloud resources

According to [Naqvi et al. 2013], certain applications such as live gaming, augmented reality do not perform efficiently on most mobile devices nowadays. Although the use of cloud resources can improve the processing of these applications, they are also sensitive to latency in communication between mobile devices and the cloud. Some context-aware mobile applications may suffer from the delay in processing contextual information, make it even unpractical. As pointed out in [Khan et al. 2012], "by the time the data makes its way to the cloud system for analysis, the opportunity to act on it might be gone". Studies conducted by our research group show how current mobile infrastructures in Brazil do not guarantee adequate QoS to transfer processing between mobile devices and the cloud [Costa et al. 2014]. This is mainly due to the distance between mobile devices and the centralized resources in cloud provider data centers. In these situations, cloud-based approaches or fog computing appear as strong candidates for meeting low latency requirements, since they typically involve one-hop communication. However, the large-scale dissemination of cloudlets or similar infrastructures is still in its infancy. There is also the fact that there are no formed business models that indicate how to encourage the popularization of these devices.

6.4.3. Security and Privacy

Cloud computing is often criticized for its centralized model of information storage on third-party machines. When thinking about the integration of context-aware computing with mobile cloud computing, this problem also rises, since the inference of contextual situations typically makes use of sensitive information from users (such as their location), which for many people, it can be seen as a breach of privacy. Traditional techniques such as information cryptography or data anonymization may be employed, but they also impose additional processing, and consequently delays and additional power consumption.

6.4.4. Power Consumption

Extending the device autonomy time is one of the biggest (if not the biggest) concerns for mobile application applications. While there has been a significant increase in the processing capacity of mobile devices each year, the same advancement is not seen in relation to the device's batteries. As a result, there are still barriers to processing or even intense data exchange, since radio transmission is one of the biggest villains for the power consumption of a mobile device. With this, the decision to process the contextual situation locally or remotely becomes even more difficult, making it a trade off between performance and energy savings. It is also possible that according to the decision, the inference of the contextual situation can be carried out in different ways, according to the context of the device. If the current situation allows to offload the inference of context to the cloud, more complex processing with more accurate results may be performed, based on larger datasets. If it is not possible, a simpler treatment can be offered, but decreasing the quality of the result of the inference. Finally, some scenarios can lead to situations where context management tasks can be divided between mobile device and cloud. This decision balances a choice between improving the accuracy of contextual inference or preserving device features.

6.4.5. Large-scale availability of MCC infrastructures

Despite the benefits that infrastructures such as cloudlets offer, having such services spread across multiple locations globally requires considerable investment, which is prohibitive for application providers. There is currently no commercial deployments that use MCC technology [Balan and Flinn 2017], probably due to the lack of appropriate business models. Even if we think of specialized companies that support MCC, or even users who want to offer resources spontaneously, aspects such as security and privacy can be obstacles to popularizing these initiatives.

6.4.6. Missing Killer applications

Experiments reports with MCC include topics such as image processing, recommendation systems, face recognition and language translation. However, according to [Balan and Flinn 2017], the popularization of the use of MCC faces two main problems. First, more advanced deep learning techniques typically used in these applications require large data sets, which impacts on the overall response time on these applications. Second, advances in the processing capability of mobile devices suppress the need of offloading tasks out of the mobile device. While this statement is true, our particular view is that only a category of mobile devices (high-end devices) meets processing capabilities for these applications. [Balan and Flinn 2017] claims that real-time video processing applications have a good potential to become a killer application in MCC field, since processing video streams does not require a huge amount of data, but demands an intensive processing to obtain data from a scene (such as subtitles on certain locations).

6.5. Conclusion

Contextual-Aware features are increasingly present in current mobile applications. Even with advances made over the past few years in mobile devices' capabilities, scenarios envisioned for context-aware mobile applications indicate the use of large datasets and machine learning techniques that will require resources not found (at least on most) in current mobile devices. This mini-course focuses on this problem, by presenting how Mobile Cloud Computing (MCC), especially offloading techniques, may improve the support for future context-aware mobile applications.

The use of "context" is already widely used in MCC integration. However, previous research works found on a literature review use this term as the situation for decision making if data or tasks offloading between mobile devices and cloud infrastructures are worthy or not, regarding performance or energy savings. In this mini-course, our approach focus on how context management services can be improved with MCC concepts. This view is presented in a practical way through the framework CAOS, an Android-based platform that supports both data and computing offloading. A case study illustrating how CAOS can be used to implement a context-aware multimedia application is described. To the best of our knowledge, this is one of the few initiatives where context management integration with cloud services is supported.

As future work, we plan to keep evolving our framework. We currently study a complete refactoring of CAOS architecture, where its services will be designed using MicroServices approach. We aim at improving CAOS' scalability and flexibility by mod-

eling its services as independent units.

References

Aguilar, J., Jerez, M., Exposito, E., and Villemur, T. (2015). Carmicloc: Context awareness middleware in cloud computing. In *2015 Latin American Computing Conference (CLEI)*, pages 1–10.

Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., and Yang, H.-I. (2002). The case for cyber foraging. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, EW 10, pages 87–92, New York, NY, USA. ACM.

Balan, R. K. and Flinn, J. (2017). Cyber foraging: Fifteen years later. *IEEE Pervasive Computing*, 16(3):24–30.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277.

Barbera, M. V., Kosta, S., Mei, A., and Stefa, J. (2013). To offload or not to offload? the bandwidth and energy costs of mobile cloud computing. In *Proc. of IEEE INFOCOM*, volume 2013.

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A., and Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive Mob. Comput.*, 6(2):161–180.

Buthpitiya, S., Luqman, F., Griss, M., Xing, B., and Dey, A. (2012). Hermes – a context-aware application development framework and toolkit for the mobile environment. In *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*, pages 663–670.

Carlson, D. and Schrader, A. (2012). Dynamix: An open plug-and-play context framework for android. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 151–158.

Chatzopoulos, D., Bermejo, C., Huang, Z., and Hui, P. (2017). Mobile augmented reality survey: From where we are to where we go. *IEEE Access*, 5:6917–6950.

Chihani, B., Bertin, E., and Crespi, N. (2013). Decoupling context management and application logic: A new framework. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*, pages 1–6.

Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. (2011). Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA. ACM.

Costa, P. B., Rego, P. A. L., Coutinho, E. F., Trinta, F. A. M., and d. Souza, J. N. (2014). An analysis of the impact of the quality of mobile networks on the use of cloudlets. In *2014 Brazilian Symposium on Computer Networks and Distributed Systems*, pages 113–121.

Costa, P. B., Rego, P. A. L., Rocha, L. S., Trinta, F. A. M., and de Souza, J. N. (2015). Mpos: A multiplatform offloading system. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, SAC '15, page 577–584, New York, NY, USA. ACM.

Cuervo, E., Balasubramanian, A., Cho, D.-k., Wolman, A., Saroiu, S., Chandra, R., and Bahl, P. (2010). Maui: Making smartphones last longer with code offload. In *MobiSys 2010, Proceedings ACM*, pages 49–62, New York, NY, USA. ACM.

Curiel, P. and Lago, A. (2012). Context management infrastructure for intelligent-mobile-services execution environments. In *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, pages 1–6.

Da, K., Dalmau, M., and Roose, P. (2011). A survey of adaptation systems. *International Journal on Internet and Distributed Computing Systems*, 2(1):1–18.

Da, K., Dalmau, M., and Roose, P. (2014a). Kalimucho: Middleware for mobile applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 413–419, New York, NY, USA. ACM.

Da, K., Roose, P., Dalmau, M., Nevado, J., and Karchoud, R. (2014b). Kali2much: A context middleware for autonomic adaptation-driven platform. In *Proceedings of the 1st ACM Workshop on Middleware for Context-Aware Applications in the IoT*, M4IOT '14, pages 25–30, New York, NY, USA. ACM.

Dey, A. K. (2001). Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2011). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*.

Dogdu, E. and Soyer, O. (2013). Morecon: A mobile restful context-aware middleware. In *Proceedings of the 51st ACM Southeast Conference*, ACMSE '13, pages 37:1–37:6, New York, NY, USA. ACM.

Doukas, C. and Antonelli, F. (2013). Compose: Building smart amp; context-aware mobile applications utilizing iot technologies. In *Global Information Infrastructure Symposium, 2013*, pages 1–6.

Duarte, P. A., Silva, L. F. M., Gomes, F. A., Viana, W., and Trinta, F. M. (2015). Dynamic deployment for context-aware multimedia environments. In *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web*, WebMedia '15, pages 197–204, New York, NY, USA. ACM.

El-Derini, M., Aly, H., El-Barbary, A.-H., and El-Sayed, L. (2014). Droidcloudlet: Towards cloudlet-based computing using mobile devices. In *Information and Communication Systems (ICICS), 2014 5th International Conference on*, pages 1–6.

Fernando, N., Loke, S. W., and Rahayu, W. (2013). Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84 – 106.

Ferrari, A., Giordano, S., and Puccinelli, D. (2016). Reducing your local footprint with anyrun computing. *Computer Communications*, 81:1 – 11.

Ferroni, M., Damiani, A., Nacci, A. A., Sciuto, D., and Santambrogio, M. D. (2014). coda: An open-source framework to easily design context-aware android apps. In *Proceedings of the 2014 12th IEEE International Conference on Embedded and Ubiquitous Computing*, EUC'14, pages 33–38, Washington, DC, USA. IEEE Computer Society.

Gomes, F. A., Viana, W., Rocha, L. S., and Trinta, F. (2016). A contextual data offloading service with privacy support. In *Proceedings of the 22nd Brazilian Symposium on Multimedia and the Web*, pages 23–30. ACM.

Gomes, F. A. A., Rego, P. A. L., Rocha, L., de Souza, J. N., and Trinta, F. (2017). Caos: A context acquisition and offloading system. *COMPSAC*.

Gordon, M. S., Jamshidi, D. A., Mahlke, S., Mao, Z. M., and Chen, X. (2012). Comet: Code offload by migrating execution transparently. In *USENIX 2012, Proceedings ACM*, pages 93–106, Berkeley, CA, USA. USENIX Association.

Herrmann, K. (2010). Self-organized service placement in ambient intelligence environments. *ACM Trans. Auton. Adapt. Syst.*, 5(2):6:1–6:39.

Kakadia, D., Saripalli, P., and Varma, V. (2013). Mecca: Mobile, efficient cloud computing workload adoption framework using scheduler customization and workload migration decisions. In *Proceedings of the First International Workshop on Mobile Cloud Computing & Networking*, MobileCloud '13, pages 41–46, New York, NY, USA. ACM.

Khan, A., Othman, M., Madani, S., and Khan, S. (2014). A survey of mobile cloud computing application models. *Communications Surveys Tutorials, IEEE*, 16(1):393–413.

Khan, R., Khan, S. U., Zaheer, R., and Khan, S. (2012). Future internet: The internet of things architecture, possible applications and key challenges. In *2012 10th International Conference on Frontiers of Information Technology*, pages 257–260.

Kharbanda, H., Krishnan, M., and Campbell, R. (2012). Synergy: A middleware for energy conservation in mobile devices. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 54–62.

Kosta, S., Aucinas, A., Hui, P., Mortier, R., and Zhang, X. (2012). Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953.

Kovachev, D., Cao, Y., and Klamma, R. (2011). Mobile cloud computing: A comparison of application models. *CoRR*, abs/1107.4940.

Kovachev, D., Yu, T., and Klamma, R. (2012). Adaptive computation offloading from mobile devices into the cloud. In *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*, pages 784–791.

Kristensen, M. D. and Bouvin, N. O. (2010). Scheduling and development support in the scavenger cyber foraging system. *Pervasive Mob. Comput.*, 6(6):677–692.

Kumar, K., Liu, J., Lu, Y.-H., and Bhargava, B. (2013). A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140.

Kumar, K. and Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56.

Lee, Y., Iyengar, S. S., Min, C., Ju, Y., Kang, S., Park, T., Lee, J., Rhee, Y., and Song, J. (2012). Mobicon: A mobile context-monitoring platform. *Commun. ACM*, 55(3):54–65.

Lima, F. F. P., Rocha, L. S., Maia, P. H. M., and Andrade, R. M. C. (2011). A decoupled and interoperable architecture for coordination in ubiquitous systems. In *Proceedings of the 2011 Fifth Brazilian Symposium on Software Components, Architectures and Reuse*, SBCARS'11, pages 31–40, Washington, DC, USA. IEEE Computer Society.

Liu, J., Ahmed, E., Shiraz, M., Gani, A., Buyya, R., and Qureshi, A. (2015). Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions. *Journal of Network and Computer Applications*, 48:99–117.

Liu, J., Shen, H., and Zhang, X. (2016). A survey of mobile crowdsensing techniques: A critical component for the internet of things. In *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6.

Ma, R. K. K., Lam, K. T., and Wang, C.-L. (2011). excloud: Transparent runtime support for scaling mobile applications in cloud. In *Proceedings of the 2011 International Conference on Cloud and Service Computing*, CSC '11, pages 103–110, Washington, DC, USA. IEEE Computer Society.

Magurawalage, C. M. S., Yang, K., Hu, L., and Zhang, J. (2014). Energy-efficient and network-aware offloading algorithm for mobile cloud computing. *Computer Networks*, 74, Part B:22 – 33. Special Issue on Mobile Computing for Content/Service-Oriented Networking Architecture.

Maia, M. E. F., Fonteles, A., Neto, B., Gadelha, R., Viana, W., and Andrade, R. M. C. (2013). Loccam - loosely coupled context acquisition middleware. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, SAC '13, pages 534–541, New York, NY, USA. ACM.

Mane, Y. V. and Surve, A. R. (2016). Capm: Context aware provisioning middleware for human activity recognition. In *2016 International Conference on Advanced Communication Control and Computing Technologies (ICACCCT)*, pages 661–665.

March, V., Gu, Y., Leonardi, E., Goh, G., Kirchberg, M., and Lee, B. S. (2011). μcloud: Towards a new paradigm of rich mobile applications. *Procedia Computer Science*, 5(0):618 – 624. The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT-2011) / The 8th International Conference on Mobile Web Information Systems (MobiWIS 2011).

Mitchell, M., Meyers, C., Wang, A.-I., and Tyson, G. (2011). Contextprovider: Context awareness for medical monitoring applications. In *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*, pages 5244–5247.

Naqvi, N. Z., Preuveneers, D., and Berbers, Y. (2013). *Cloud Computing: A Mobile Context-Awareness Perspective*, pages 155–175. Springer London, London.

OSullivan, M. J. and Grigoras, D. (2016). Context aware mobile cloud services: A user experience oriented middleware for mobile cloud computing. In *2016 4th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 67–72.

Preuveneers, D. and Berbers, Y. (2007). Towards context-aware and resource-driven self-adaptation for mobile handheld applications. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, SAC '07, pages 1165–1170, New York, NY, USA. ACM.

Pulli, K., Baksheev, A., Kornyakov, K., and Eruhimov, V. (2012). Real-time computer vision with opencv. *Commun. ACM*, 55(6):61–69.

Punjabi, J., Parkhi, S., Taneja, G., and Giri, N. (2013). Relaxed context-aware machine learning midddleware (rcamm) for android. In *Intelligent Computational Systems (RAICS), 2013 IEEE Recent Advances in*, pages 92–97.

Rego, P. A. L., Cheong, E., Coutinho, E. F., Trinta, F. A., Hasan, M. Z., and de Souza, J. N. (2017). Decision tree-based approaches for handling offloading decisions and performing adaptive monitoring in MCC systems. In *2017 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*.

Rego, P. A. L., Costa, P. B., Coutinho, E. F., Rocha, L. S., Trinta, F. A., and de Souza, J. N. (2016). Performing computation offloading on multiple platforms. *Computer Communications*.

Sanaei, Z., Abolfazli, S., Gani, A., and Buyya, R. (2014). Heterogeneity in mobile cloud computing: Taxonomy and open challenges. *Communications Surveys Tutorials, IEEE*, 16(1):369–392.

Satyanarayanan, M. (2001). Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10–17.

Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23.

Sharifi, M., Kafaie, S., and Kashefi, O. (2012). A Survey and Taxonomy of Cyber Foraging of Mobile Devices. *IEEE Communications Surveys & Tutorials*, 14(4):1232–1243.

Verbelen, T., Simoens, P., De Turck, F., and Dhoedt, B. (2012). Aiolos: Middleware for improving mobile application performance through cyber foraging. *J. Syst. Softw.*, 85(11):2629–2639.

Viana, W., Miron, A. D., Moisuc, B., Gensel, J., Villanova-Oliver, M., and Martin, H. (2011). Towards the semantic and context-aware management of mobile multimedia. *Multimedia Tools Appl.*, 53(2):391–429.

Weiser, M. (1991). The computer for the 21st century. *Scientific American*, 265(3):94–104.

Williams, E. and Gray, J. (2014). Contextion: A framework for developing context-aware mobile applications. In *Proceedings of the 2Nd International Workshop on Mobile Development Lifecycle*, MobileDeLi '14, pages 27–31, New York, NY, USA. ACM.

Xia, Q., Liang, W., and Xu, W. (2013). Throughput maximization for online request admissions in mobile cloudlets. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 589–596.

Xiao, Y., Simoens, P., Pillai, P., Ha, K., and Satyanarayanan, M. (2013). Lowering the barriers to large-scale mobile crowdsensing. In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications*, HotMobile '13, pages 9:1–9:6, New York, NY, USA. ACM.

Yurur, O., Liu, C. H., Sheng, Z., Leung, V. C. M., Moreno, W., and Leung, K. K. (2016). Context-awareness for mobile sensing: A survey and future directions. *IEEE Communications Surveys Tutorials*, 18(1):68–93.

Zhang, Y., Huang, G., Liu, X., Zhang, W., Mei, H., and Yang, S. (2012). Refactoring android java code for on-demand computation offloading. *SIGPLAN Not.*, 47(10):233–248.

Authors' Biographies

Fernando A. M. Trinta holds a PhD degree from Federal University of Pernambuco (2007). He is an adjunct professor at the Computer Science Department - Federal University of Ceará since 2011. He works mainly in the areas of Multimedia, Software Engineering and Distributed Systems. Currently, his research is focused on Mobile Cloud Computing, Context-Aware Computing and Fog Computing.

Paulo A. L. Rego holds a PhD in Computer Science from the Federal University of Ceará (UFC) since 2016. He is currently an adjunct professor at UFC - Campus Quixadá. He works in the area of Computer Science, with emphasis on Computer Networks and Distributed Systems. His main research interests include Mobile and Cloud Computing, Edge and Fog Computing.

Francisco A. A. Gomes holds a Master degree from the Federal University of Ceará. He is currently a PhD student. He works in the area of Software Engineering applied to Ubiquitous Computing. His main research interests are: Mobile Computing and MCC.

Lincoln S. Rocha holds a PhD in Computer Science from Federal University of Ceará (2013). He is currently a professor at the same university and has experience in the area of Software Engineering. His man research interests include Mobile and Ubiquitous Computing, Context-Aware Computing and Internet of Things.

José N. de Souza holds a PhD degree at Pierre and Marie Curie University (PARIS VI/MASI Laboratory), France, since 1994. He is currently working as a researcher full professor at the Federal University of Ceará in the Computer Science Department, and is IEEE senior member. Since 1999, he has been the Brazilian representative at the IFIP TC6. His main research interests are Cloud Computing and Network Management.

Índice de Autores