UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Programa de Pós-Graduação em Computação

Tese

Uma Linguagem Visual e Metodologias para Resolução de Problemas Fundamentadas no Pensamento Computacional

Adriana Bordini

Pelotas, 2020

Adriana Bordini

Uma Linguagem Visual e Metodologias para Resolução de Problemas Fundamentadas no Pensamento Computacional

Tese apresentada ao Programa de Pós-Graduação em Computação do Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Doutor em Ciência da Computação.

Orientadora: Prof. Dr. Simone André da Costa Cavalheiro
Coorientadora: Prof. Dr. Luciana Foss

Pelotas, 2020

Universidade Federal de Pelotas / Sistema de Bibliotecas
Catalogação na Publicação

B729l Bordini, Adriana

Uma linguagem visual e metodologias para resolução de problemas fundamentadas no pensamento computacional / Adriana Bordini ; Simone André da Costa Cavalheiro, orientadora ; Luciana Foss, coorientadora. — Pelotas, 2020.

119 f. : il.

Tese (Doutorado) — Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2020.

1. Pensamento computacional. 2. Linguagem de especificação. 3. Resolução de problemas. 4. Habilidades do século xxi. I. Cavalheiro, Simone André da Costa, orient. II. Foss, Luciana, coorient. III. Título.

CDD : 005

Elaborada por Maria Inez Figueiredo Figas Machado CRB: 10/1612

Adriana Bordini

Uma Linguagem Visual e Metodologias para Resolução de Problemas Fundamentadas no Pensamento Computacional

Tese aprovada, como requisito parcial, para obtenção do grau de Doutor em Ciência da Computação, Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 26 de maio de 2020

Banca Examinadora:

Profa. Dra. Simone André da Costa Cavalheiro (orientador)
Doutora em Computação pela Universidade Federal do Rio Grande do Sul.

Profa. Dra. Renata Hax Sander Reiser
Doutora em Computação pela Universidade Federal do Rio Grande do Sul.

Profa. Dra. Regina Trilho Otero Xavier
Doutora em Informática na Educação pela Universidade Federal do Rio Grande do Sul.

Prof. Dr. Luis Otoni Meireles Ribeiro
Doutor em Informática na Educação pela Universidade Federal do Rio Grande do Sul.

Aos meus pais *Antônio* (in memorian) e *Catarina*.
E ao meu filho *Eduardo* e meu esposo *Luiz Eduardo*.

AGRADECIMENTOS

À Deus pela oportunidade.

Aos meus pais Antônio e Catarina pelo amor e por sempre acreditarem, apoiarem e incentivarem.

Ao meu filho Eduardo por aceitar as minhas ausências na metade da sua vida.

Ao meu marido Luiz Eduardo pelo companheirismo e mais uma vez tornar possível.

Aos meus irmãos Andréia e Antônio Júnior pelo apoio e incentivo.

Aos meus tios/dindos Nicéia e Gilberto pelo carinho e sempre acreditarem.

Às professoras Simone e Luciana por me acompanharem nesta caminhada com a constante orientação neste trabalho.

E a todos, que direta ou indiretamente, ajudaram neste sonho, de objetivo virar meta e se tornar realidade.

Muito Obrigada!

“A inquietude não deve ser negada, mas remetida para novos horizontes e se tornar nosso próprio horizonte.”
— EDGAR MORIN

RESUMO

BORDINI, Adriana. **Uma Linguagem Visual e Metodologias para Resolução de Problemas Fundamentadas no Pensamento Computacional**. Orientadora: Simone André da Costa Cavalheiro. 2020. 119 f. Tese (Doutorado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2020.

As linguagens que cercam os estudantes fora da escola, no dia a dia, por meio da TV, dos vídeos, das animações, dos jogos, dos aplicativos e dos dispositivos móveis, são muitas vezes visuais e multimídias, trazendo textos, imagens, sons, vídeos, *emojis* e animações. Além disso, as tecnologias inseridas em nossa vida cotidiana mudam as formas de pensar e de resolver problemas. Consequentemente essas mudanças também alteram as habilidades necessárias para a educação do século XXI. Diversos autores concordam que muitas dessas habilidades podem ser desenvolvidas por meio do Pensamento Computacional (PC). Jeannette Wing descreveu o PC como um processo de resolução de problemas, fundamentado na Ciência da Computação, capaz de promover as competências de abstração, decomposição, automação, análise, entre outras, e que podem ser aplicadas em qualquer área do conhecimento. Muitos trabalhos estão sendo feitos na área, e muitos propõem a resolução de problemas por meio da programação. Embora este método tenha se mostrado eficaz para o desenvolvimento de muitas das competências do PC, ele exige que a solução dos problemas seja dada, em geral, em um nível baixo de abstração. Muitas vezes o foco acaba sendo dado mais na descrição sintática e na aprendizagem de comandos da linguagem do que nas estratégias para a resolução dos problemas. O objetivo deste trabalho é propor uma linguagem de especificação visual com um alto nível de abstração e um conjunto de metodologias que se fundamentam nas técnicas propostas pelo PC, como refinamento, decomposição, composição, abstração e generalização, permitindo que essas estratégias de resolução de problemas possam ser aplicadas em vários níveis de ensino e em diversas áreas do conhecimento. Por fim, estudos de caso foram realizados para avaliar a linguagem e as metodologias. Em um dos estudos foi apresentado um problema para ser resolvido com a linguagem visual proposta e com a linguagem natural. O estudo mostrou que os alunos que utilizaram a linguagem visual para resolver o problema, chegaram a uma solução com um nível maior de detalhamento e exatidão.

Palavras-chave: Pensamento Computacional. Linguagem de Especificação. Resolução de problemas. Habilidades do século XXI.

ABSTRACT

BORDINI, Adriana. **A Visual Language and Methodologies for Problem Solving Based on Computational Thinking**. Advisor: Simone André da Costa Cavalheiro. 2020. 119 f. Thesis (Doctorate in Computer Science) – Technology Development Center, Federal University of Pelotas, Pelotas, 2020.

The languages that surround students outside of school, day to day life, through TV, videos, animations, games, applications and mobile devices, are often visual and multimedia languages, bringing texts, images, sounds, videos, emojis and animations. Besides, the technologies inserted in our daily life change the ways of thinking and solving problems. Consequently, these changes also modify the skills needed for 21st century education. Several authors agree that many of these skills can be developed through Computational Thinking (CT). Jeannette Wing described the CT as a problem solving process, based on Computer Science, capable of promoting the skills of abstraction, decomposition, automation, analysis, among others, and that can be applied in any area of knowledge. Many works are being done in the area, and many propose to solve problems through programming. Although this method has been shown to be effective for the development of many of the CT's skills, it requires that the solution of problems is, in general, given at a low level of abstraction. Often, the focus ends up being more on syntactic description and on learning language commands than on strategies for solving problems. The goal of this work is to propose a visual specification language with a high level of abstraction and a set of methodologies that are based on the techniques proposed by the CT, such as refinement, decomposition, composition, abstraction and generalization, allowing these problem solving strategies can be applied at various levels of education and in different areas of knowledge. Finally, case studies were carried out to evaluate language and methodologies. In one of the studies, a problem was presented to be solved with the proposed visual language and with natural language. The study showed that students who used visual language to solve the problem, arrived at a solution with a greater level of detail and accuracy.

Keywords: Computational Thinking. Visual Specification Language. Problem solving. 21st century skills.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BNCC	Base Nacional Comum Curricular
CC	Ciência da Computação
CD	Componente Decomposto
CE	Componente Elementar
CSTA	*Computer Science Teachers Association*
DFD	Diagrama de Fluxo de Dados
ISTE	*International Society for Technology in Education*
PC	Pensamento Computacional
RSL	Revisão Sistemática de Literatura
UML	*Unified Modeling Language*
VPL	*Visual Programming Language*

SUMÁRIO

1 INTRODUÇÃO

As tecnologias inseridas em nossa vida cotidiana e em qualquer área do conhecimento mudam as formas de pensar e de resolver problemas. Consequentemente essas mudanças também alteram as habilidades que devem ser trabalhadas nos diferentes níveis de educação do século XXI. Além do conhecimento de conteúdos a serem inseridos de forma interdisciplinar, tais como: matemática, geografia, idiomas, consciência global e cultural, alfabetização financeira, alfabetização ambiental, entre outros; os estudos concordam em algumas habilidades críticas a serem desenvolvidas, tais como: criatividade e imaginação, pensamento crítico, resolução de problemas, comunicação oral e escrita, colaboração e trabalho em equipe, alfabetização tecnológica, iniciativa, responsabilidade social e ética, entre outros; as quais são consideradas essenciais para que os alunos fiquem preparados para o futuro (Partnership for 21st Century Skills, 2019; NOURI et al., 2019; ENVISION EXPERIENCE, 2018).

Diversos autores concordam (TABESH, 2017; YADAV et al., 2016; MOHAGHEGH; MCCAULEY, 2016) que muitas das habilidades acima citadas podem ser desenvolvidas por meio do Pensamento Computacional (PC). Jeannette Wing (WING, 2006) descreveu o PC como um processo de resolução de problemas, fundamentado na Ciência da Computação (CC), capaz de promover as competências de abstração, decomposição, automação, análise, entre outras, e que podem ser aplicadas em qualquer área do conhecimento (WING, 2006). Em particular, a *Computer Science Teachers Association* (CSTA) e a *International Society for Technology in Education* (ISTE) junto com colaboradores propuseram a definição operacional do PC[1], delineando seu escopo e dimensões.

Na Base Nacional Comum Curricular (BNCC) (BRASIL, 2018) muitos destes conhecimentos e habilidades são listados e considerados essenciais para o estudante brasileiro, para que possa se desenvolver como pessoa, se preparar para o exercício da cidadania e se qualificar para o trabalho. Na BNCC, dez competências são listadas, incluindo, resolução de problemas e comunicação.

Diversos trabalhos propuseram a introdução de conceitos da Computação na edu-

[1] http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf

cação, bem como o desenvolvimento de habilidades do PC em seus diferentes níveis. Pesquisas já relatam as diferentes estratégias adotadas para este fim na educação (MIOTO et al., 2019; WANGENHEIM et al., 2019; MARQUES et al., 2017; BORDINI et al., 2016a; MARTINS et al., 2016; MOTA et al., 2014), bem como descrevem os desdobramentos do PC no Brasil (BORDINI et al., 2016b; OLIVEIRA et al., 2016; ANDRADE et al., 2013) e no exterior (BORDINI et al., 2017).

Diante dos recursos tecnológicos e a influência de diferentes estímulos multimídias, percebe-se que as habilidades destacadas como importantes para a educação do século XXI precisam ser trabalhadas de forma diferente do ensino tradicional (MORAN, 2019). Dado que mudando o contexto de aprendizagem, deve-se mudar a forma de ensinar (MORAN, 2018, 2007, 2000), sendo necessário que novas metodologias de ensino/aprendizagem sejam propostas.

Assim como as habilidades e competências necessárias para a resolução de problemas vêm se modificando ao longo do tempo, as linguagens utilizadas para a descrição das soluções também. Desde a pré-história até os dias atuais a linguagem é utilizada para a comunicação. E aos poucos os meios de comunicação, tais como: jornal, rádio, televisão, telefonia fixa e móvel tiveram uma influência universal sobre o caráter das línguas (CRYSTAL, 2005). Sendo a Internet o último desses meios, o seu impacto na língua tem sido o mais revolucionário de todos (CRYSTAL, 2005).

A Internet antes continha na sua maioria hipertextos e algumas imagens. Hoje ela possui muitas linguagens, com multimídias, trazendo textos, imagens, sons, vídeos, *emojis* e animações. Tornando-se um "...veículo eletrônico, global e interativo..." (CRYSTAL, 2005, p. 80), trazendo consequências para a linguagem, tanto a encontrada na rede, como a falada entre as pessoas na sociedade. A Internet explora linguagens visuais como forma de atrair e alcançar um maior número de pessoas. Linguagens estas que estão presentes no dia a dia dos estudantes, por meio da TV, dos vídeos, das animações, dos jogos, dos aplicativos, dos ícones e dos dispositivos móveis. Estas linguagens que cercam os estudantes fora da escola, nas mídias e redes sociais, linguagens visuais, muitas vezes são mais fáceis de se compreender, de se identificar os componentes e as relações entre estes, pois o potencial didático da imagem como facilitadora no processo de ensino-aprendizagem, colabora para que estas sejam "... mais facilmente lembradas do que suas correspondentes representações verbais" (MARTINS; GOUVÊA; PICCININI, 2005). Mas neste contexto de imagens, de multimídias, onde os alunos recebem estímulos visuais e sonoros, o sistema de ensino continua o mesmo de anos atrás. Pois quando o aluno chega na escola tudo é diferente do mundo que o cerca lá fora, sendo menos atrativo e voltado à educação do século passado, uma educação textual e oral, uma educação desconectada da sua realidade (DEMO, 2007).

Na computação existem as linguagens de programação e de especificação, tanto

as textuais quanto as visuais. As linguagens de programação especificam um conjunto de instruções e regras usadas para gerar programas de computador. As linguagens de especificação permitem descrever um sistema em alto nível de abstração, e através de um processo de refinamento, com a adição de detalhes de implementação, pode-se chegar a um algoritmo descrito em uma linguagem de programação. As linguagens visuais possuem um maior nível de abstração, facilitando a construção das soluções, em geral, em um ambiente mais lúdico e atraente, "...porque deixa evidente o fluxo e as ações envolvidas" (RIBEIRO; FOSS; CAVALHEIRO, 2017, p.7).

Existem várias linguagens visuais na computação com diversos níveis de abstração para solucionar problemas tais como: na engenharia de software, os diagramas de fluxo de dados (DFDs) e os diagramas da Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML); em banco de dados, os diagramas de entidade-relacionamento (ER); em algoritmos, os diagramas de blocos e os fluxogramas; em sistemas distribuídos, as redes de Petri e a gramática de grafos; na teoria da computação, as máquinas de estado, entre outros. No entanto, todas estas linguagens citadas, focam na área da Computação, sendo, em geral, utilizadas na modelagem e/ou no desenvolvimento de sistemas.

Neste trabalho, propõe-se uma linguagem de especificação visual voltada para a resolução de problemas das mais diversas áreas, usando um nível de abstração mais alto que os disponibilizados pelas linguagens de programação. Além disso, é apresentado um conjunto de metodologias, fundamentadas nas técnicas/estratégias do PC, que servem para guiar a resolução de problemas usando diferentes técnicas da área da Computação. Na próxima seção apresenta-se a justificativa deste trabalho.

1.1 Justificativa

Diversos trabalhos abordam a resolução de problemas, estando geralmente relacionados a programação, simulação, jogos e/ou robótica, muitas vezes em ambientes de programação visual (PELLAS; VOSINAKIS, 2018; REZENDE; BISPO, 2018; MAQUIL et al., 2018; MALIZIA et al., 2017; SU; WANG, 2017; GROVER; PEA; COOPER, 2016; LAKANEN; ISOMÖTTÖNEN, 2015; DORLING; WHITE, 2015; SULLIVAN et al., 2015; LEE et al., 2014; KAFAI; BURKE, 2013; KOH et al., 2010). Diversos focam na resolução em baixo nível de abstração e não em técnicas de resolução de problemas, tais como, decomposição, abstração, refinamento, generalização, entre outras.

Alguns autores (GARDELI; VOSINAKIS, 2017; KOH et al., 2010; HOWLAND; GOOD; NICHOLSON, 2009) questionam, tanto o uso da programação quanto os próprios ambientes visuais de programação no desenvolvimento de habilidades do PC. São questionados quais conceitos do PC são realmente aprendidos e até que ponto a aquisição de habilidades do PC tem sido um efeito colateral de aprender a programar.

Afirmam ainda que: o PC é mais amplo do que programação e deve ser trabalhado em um nível mais alto de abstração (HOWLAND; GOOD; NICHOLSON, 2009); a utilização de um ambiente computacional fácil de usar, por si só, não é suficiente para a aquisição efetiva de habilidades do PC (GARDELI; VOSINAKIS, 2017); e as linguagens de programação visual existentes, por serem mais lúdicas e intuitivas, acabam por "mascarar" os conceitos de computação que estão sendo trabalhados e, consequentemente, impedem o desenvolvimento de habilidades do PC (HOWLAND; GOOD; NICHOLSON, 2009).

Por meio de revisões sistemáticas, detalhadas no capítulo 2, observou-se a ausência de uma linguagem visual de alto nível de abstração com o foco no desenvolvimento das técnicas de resolução de problemas do PC. Assim o objetivo deste trabalho é estabelecer uma linguagem alternativa às linguagens de programação para introduzir as estratégias de solução de problemas do PC.

A ideia é desviar do foco da programação. Ou seja, é estabelecida uma linguagem visual de alto nível de abstração, que possa ser usada por pessoas de qualquer área do conhecimento. Além disso, espera-se que a linguagem facilite seu uso, visto que, por ser visual, possui uma representação mais próxima do dia a dia dos estudantes.

Assim, a linguagem visual proposta não é uma linguagem de programação, mas sim uma linguagem de especificação, que visa desenvolver técnicas/estratégias do Pensamento Computacional por meio de metodologias que se fundamentam nas técnicas/estratégias de refinamento, decomposição, composição, abstração e generalização. A linguagem proposta foi inspirada em três linguagens visuais da computação: DFD, Diagramas de atividades UML e Simulink. Na próxima seção apresenta-se os objetivos que nortearam este trabalho.

1.2 Objetivos

O **objetivo geral** deste trabalho é definir uma linguagem de especificação visual para resolução de problemas que permita especificar soluções desde um alto nível de abstração, focando nas técnicas/estratégias utilizadas pelo PC (como refinamento, decomposição, composição, abstração e generalização) e que seja aplicável a diversas áreas do conhecimento e diferentes níveis de ensino.

Para atingir o objetivo geral, alguns **objetivos específicos** foram definidos:

1. Estudar o estado da arte sobre o tema PC no escopo da Educação Básica. Identificando as principais abordagens utilizadas para a integração do PC no ensino, escrevendo seus objetivos, práticas pedagógicas, as ferramentas utilizadas e/ou criadas, bem como que conceitos e habilidades têm sido trabalhados;

2. Definir uma linguagem visual detalhando seus componentes e especificando sua sintaxe por meio de componentes visuais, bem como definindo formalmente sua sintaxe e semântica;

3. Identificar as técnicas do PC que vão ser abordadas;

4. Propor metodologias fundamentadas no PC que guiem a descrição de soluções por meio da linguagem proposta, conduzindo a aplicação das estratégias do PC na especificação de soluções;

5. Avaliar a linguagem e as metodologias propostas por meio de estudos de caso.

1.3 Etapas Metodológicas: Visão Geral

Nesta seção são listadas as etapas percorridas na realização deste trabalho:

1 **Referencial teórico**: levantamento do referencial teórico e do estado da arte.

2 **Revisão sistemática**: realização de revisões sistemáticas de literatura dos principais projetos na área do Pensamento Computacional, com foco no ensino fundamental e médio, que tiveram resultados publicados nos principais veículos de Informática na Educação no Brasil e no mundo.

3 **Linguagem visual**: determinação das construções da linguagem visual LiVE, estabelecendo suas representações visuais, bem como definindo formalmente sua sintaxe e semântica.

4 **Metodologias de uso da linguagem**: definição de metodologias de uso da linguagem visual LiVE, considerando as técnicas de refinamento, decomposição, composição, abstração e generalização do PC.

5 **Estudos de caso**: elaboração e aplicação de estudos de caso para avaliação do uso da linguagem.

1.4 Estrutura da Tese

A tese está organizada da seguinte maneira. No Capítulo 2 são apresentadas as revisões sistemáticas de literatura do desenvolvimento do PC na educação no Brasil e no mundo, o uso das linguagens visuais e das metodologias existentes para este fim. No Capítulo 3 são apresentados alguns conceitos e fundamentos do PC que são utilizados na linguagem proposta. Ainda neste capítulo são apresentadas algumas linguagens computacionais e metodologias. No Capítulo 4 é definida a linguagem visual LiVE. Esta definição inclui a sintaxe e a semântica da linguagem, além de exemplos

de aplicação. No Capítulo 5 são apresentadas quatro metodologias propostas para solução de problemas na linguagem: de refinamento, de abstração, de generalização, e de decomposição e composição. No Capítulo 6 são apresentados os estudos de caso realizados para avaliação da linguagem bem como das metodologias propostas. Ainda neste capítulo, são delineadas as mudanças que ocorreram nas propostas das metodologias durante a pesquisa. Por fim, no Capítulo 7 são apresentadas as conclusões deste trabalho, bem como os trabalhos futuros.

2 TRABALHOS RELACIONADOS

Para identificar o estado da arte sobre o tema deste trabalho, foram realizadas revisões sistemáticas de literatura (RSL) sobre o PC na educação do Brasil e do mundo, o desenvolvimento do PC por meio de linguagens visuais e por meio de linguagens de especificação, e as metodologias para o desenvolvimento do PC. Uma RSL tem por objetivo "...identificar, avaliar e interpretar todas as pesquisas disponíveis relevantes para um determinado tema" (KITCHENHAM, 2004). Cada RSL realizada dividiu-se basicamente em três fases: planejamento, realização e relato da revisão.

A seção 2.1 descreve três mapeamentos que foram realizados para a definição da proposta desta tese: os dois primeiros levantamentos abordaram, respectivamente, a inserção da Computação e do PC na educação no Brasil (BORDINI et al., 2016a,b); e o terceiro levantamento buscou caracterizar os principais objetivos e metodologias dos trabalhos sobre o tema no escopo mundial (BORDINI et al., 2017).

Os mapeamentos delineados nas demais seções foram feitos após a proposição dos objetivos deste trabalho e refeitos após sua realização. A seção 2.2 aborda o uso de linguagens de especificação (visuais) no desenvolvimento do PC. A seção 2.3 descreve a integração de metodologias na promoção de habilidades do PC.

2.1 Desenvolvimento do Pensamento Computacional na Educação

Esta seção traz o panorama encontrado sobre o desenvolvimento do PC na Educação no período de definição do objetivo desta tese. Constitui-se de levantamentos realizados com o propósito de identificar questões em aberto na área. A subseção 2.1.1 traz um panorama das estratégias que estavam sendo utilizadas para integração da Computação no ensino Básico. A subseção 2.1.2 investiga os trabalhos que abordavam PC no Brasil e a subseção 2.1.3 estende esta pesquisa para o âmbito mundial.

2.1.1 Computação na Educação do Brasil

Inicialmente foi realizado um levantamento dos principais projetos na área da computação, com foco no ensino fundamental e médio (BORDINI et al., 2016a), que tiveram resultados publicados no período de 2010 a 2015, em 7 principais veículos de Informática na Educação no Brasil: SBIE (Simpósio Brasileiro de Informática na Educação), WIE (Workshop de Informática na Escola), WAlgProg (Workshop de Ensino em Pensamento Computacional, Algoritmos e Programação), RBIE (Revista Brasileira de Informática na Educação), WEI (Workshop sobre Educação em Computação), WEIT (Workshop Escola de Informática Teórica) e CTD-IE (Concurso de TCC, Dissertações e Teses do Congresso Brasileiro de Informática na Educação). Com a *string* de busca selecionou-se artigos que continham as palavras "computação" e as palavras "educação" ou "escola" ou "ensino" ou "aprendizagem" em qualquer parte do texto.

A pesquisa direcionou-se principalmente a trabalhos que integrassem computação e educação básica, mas com alguma interface com o PC, seja pelas habilidades ou pelas atitudes trabalhadas. Dentre os trabalhos analisados, observou-se diferentes estratégias aplicadas na introdução da computação no ensino básico, das quais se destacam: algoritmos e programação, computação desplugada, jogos, robótica, dentre outras (teatro/música e ensino híbrido). O mapeamento do estudo realizado foi organizado a partir destas estratégias e, em cada uma delas, considerou-se 4 aspectos: os conceitos da ciência da computação abordados, as ferramentas utilizadas, a relação com outras disciplinas e a forma com que a estratégia tem desenvolvido a colaboração e/ou a comunicação (quando aplicável). Dentre os aspectos considerados nesta pesquisa, identificou-se pouco incentivo ao trabalho colaborativo e/ou interdisciplinar. A grande maioria, independente da abordagem (Algoritmos e Programação, Robótica, Jogos, Desplugada, entre outras) trabalhava apenas os conceitos e conteúdos da ciência da computação, e poucos integravam outras disciplinas que faziam parte do currículo básico escolar. A colaboração, também era outro aspecto pouco explorado, sendo abordada pela maioria dos trabalhos apenas como execução de tarefas em grupos.

2.1.2 Pensamento Computacional na Educação Brasileira

Concomitantemente com o levantamento descrito na subseção anterior, um segundo mapeamento dos trabalhos na área do PC na educação básica foi realizado (BORDINI et al., 2016b). O período pesquisado foi de 6 anos, de 2010 a 2015, em 8 principais veículos na área de Informática na Educação no Brasil: SBIE (Simpósio Brasileiro de Informática na Educação), WIE (Workshop de Informática na Escola), WAlgProg (Workshop de Ensino em Pensamento Computacional, Algoritmos e Programação), RBIE (Revista Brasileira de Informática na Educação), WEI (Workshop

sobre Educação em Computação), WEIT (Workshop Escola de Informática Teórica), RENOTE (Revista Novas Tecnologias na Educação) e DesafIE (Workshop de Desafios da Computação Aplicada à Educação). Com a *string* de busca selecionou-se artigos que continham as palavras "pensamento computacional" e as palavras "educação" ou "escola" ou "ensino" ou "aprendizagem" em qualquer parte do texto.

Este levantamento buscou responder as seguintes questões: Qual tem sido o objetivo de estudo dos trabalhos na área? Qual o público alvo e o número de participantes das pesquisas? Quais os resultados reportados? Quais ambientes ou ferramentas têm sido adotados? O objetivo deste levantamento foi o de caracterizar os desdobramentos no Brasil sobre o tema PC.

Dentre os trabalhos analisados, foi possível observar que a maioria dos projetos realizavam práticas de ensino, com o objetivo principal de introduzir conceitos do PC. Estas experiências de ensino eram na sua maioria aplicadas ao Ensino Fundamental, utilizando as ferramentas *Scratch* e "Computação Desplugada". Os resultados relataram: um aumento do interesse pela área da computação, por todos os públicos trabalhados, inclusive por meninas que foram foco de alguns estudos; maior esclarecimento e/ou conhecimento sobre o tema PC; habilidades comuns entre a Matemática e o PC; propostas de metodologias e formas de avaliar o PC; entre outros. A maioria dos trabalhos tinham a perspectiva de continuar os projetos, ampliando a quantidade de alunos e o público-alvo. Assim como incentivavam a busca de ferramentas, metodologias e avaliações para o ensino de computação no Brasil, mostrando que o tema PC no país ainda estava em expansão.

2.1.3 Pensamento Computacional na Educação Mundial

O objetivo desta revisão foi o de identificar, avaliar e interpretar pesquisas relevantes na área do PC no âmbito do Ensino Fundamental e Médio no mundo (BORDINI et al., 2017). Consideraram-se os artigos publicados nos 5 anos anteriores a 2017 em periódicos e anais de conferências internacionais, abrangendo o período de 2012 a agosto/2016, no idioma inglês. Optou-se por realizar a busca dos trabalhos nas principais bases de dados científicas digitais *on-line*: ACM (*Association for Computing Machinery*), IEEE (*Institute for Electrical and Electronics Engineers*), *Science Direct*, *Springer* e *Scopus*. Com a *string* de busca selecionou-se artigos que continham as palavras "*computational thinking*" no título ou no resumo ou nas palavras chaves do autor e as palavras *education* ou *school* ou *k-12* ou *teach* ou *learn* em qualquer parte do texto.

Após a busca e a seleção por trabalhos completos publicados em conferências e periódicos com Qualis A1 a B1, excluindo outras revisões na área, obtiveram-se 80 artigos, a partir dos quais foi preenchido um formulário de extração de informações. Além das informações básicas (dados bibliográficos, data de publicação, entre

outros), esse formulário contém a síntese do trabalho, seu objetivo, categoria (criada pela autora desta tese), público alvo, ferramentas utilizadas, metodologia, resultados apontados e algumas reflexões pessoais. No que segue sumariza-se o relato desta revisão.

Este levantamento buscou caracterizar os principais objetivos e metodologias dos trabalhos sobre o tema. Do total dos trabalhos analisados, foram identificados trabalhos que: buscaram desenvolver habilidades do PC por meio da programação (6 artigos), da robótica (7 artigos), de jogos (2 artigos), do uso e/ou criação de ferramentas (5 artigos) ou da introdução de conceitos de computação (5 artigos); investigaram a inserção da computação no currículo (10 artigos); apresentaram metodologias para o desenvolvimento do PC (9 artigos); propuseram metodologias de avaliação do PC (17 artigos); buscaram a integração da computação em outras disciplinas (10 artigos); realizaram intervenções com professores da educação básica (7 artigos); e investigaram a percepção da comunidade escolar sobre PC (2 artigos). Para cada um dos grupos identificados, buscou-se responder as seguintes questões de pesquisa: Quais são os objetivos, ferramentas utilizadas e/ou desenvolvidas e as práticas pedagógicas adotadas em cada uma das abordagens? Quais são as habilidades do PC que têm sido trabalhadas e quais são os conceitos de computação abordados?

Dentro das diferentes abordagens encontradas, destacaram-se: na “avaliação do PC”, predominou o uso e/ou criação de ferramentas para a avaliação de habilidades do PC; na “integração da computação com outras disciplinas”, as disciplinas da área de ciências, matemática e física, foram as mais citadas; na “inserção da computação no currículo”, a maioria dos trabalhos propõe a computação como disciplina; nas “metodologias de implantação”, predominaram propostas associadas ao desenvolvimento de jogos.

Observou-se que aulas de programação eram as práticas pedagógicas mais utilizadas, onde geralmente eram propostas através da metodologia de projetos, o que facilitava e incentivava a programação em pares e/ou colaboração. A abordagem *scaffolding* também apareceu em muitos trabalhos auxiliando a construção do conhecimento.

O foco era dado nas habilidades de abstração e pensamento algorítmico, as quais foram trabalhadas através dos conceitos de algoritmos e programação. Dentre as linguagens de programação que predominavam, as visuais eram as mais adotadas. Quanto às metodologias de avaliação empregadas, observou-se que em geral avaliações qualitativas eram obtidas por meio de observações e entrevistas e as quantitativas eram obtidas pela aplicação de pré- e pós- testes.

Muitos objetivos de pesquisa encontrados neste levantamento foram semelhantes aos encontrados no Brasil (BORDINI et al., 2016b), tais como: atrair a atenção para a área da computação, alguns em especial das meninas; utilizar linguagens de progra-

mação visuais em experiências de ensino de programação; e buscar formas de avaliar o desenvolvimento de habilidades do PC. Embora muitos trabalhos apresentassem metodologias para o ensino de habilidades do PC, muitas eram propostas isoladas, carecendo de um modelo conceitual genérico. Igualmente, as propostas de inserção do PC no currículo, seja como disciplina ou interdisciplinarmente, ainda se constituíam de propostas independentes e desconectadas, de forma que investigações e fundamentações de quais habilidades do PC e/ou conceitos da computação seriam adequados a cada idade e/ou ano da educação básica deveriam ser estabelecidas. Metodologias de ensino para o melhor desenvolvimento de cada uma das dimensões e habilidades do PC também careciam de fundamentações.

2.2 Linguagens no Desenvolvimento do Pensamento Computacional

A partir das revisões de literatura realizadas, observou-se uma carência de linguagens e de metodologias que permitissem focar nas técnicas de resolução de problemas e no desenvolvimento de habilidades do PC, sem a necessidade de aprender e manipular conceitos da Computação de baixo nível de abstração. Diversos trabalhos utilizam linguagens de programação como meio de resolver problemas promovendo o PC. No entanto, muitas construções e detalhes envolvidos no uso destas linguagens não dizem respeito a resolução do problema em si, mas se relacionam com a forma de implementação da linguagem (envolvendo, por exemplo, o armazenamento de valores na memória, relações entre variáveis locais e globais, entre outros). Diante destes resultados, definiu-se o objetivo desta tese: definir uma linguagem de especificação visual de alto nível de abstração para a qual seja possível desenvolver metodologias que viabilizem a solução de problemas por meio de técnicas do PC.

As próximas seções buscam identificar o estado da arte sobre o desenvolvimento do PC por meio de linguagens de especificação visuais (subseções 2.2.1 e 2.2.2), bem como as metodologias propostas para a promoção do PC (seção 2.3).

2.2.1 Linguagem de Especificação Visual para o Desenvolvimento do Pensamento Computacional

Na primeira fase, do planejamento, identificou-se que não havia um levantamento já realizado neste escopo e definiu-se o protocolo da pesquisa. O objetivo do levantamento foi o de identificar, avaliar e interpretar pesquisas relevantes que integrassem PC e Linguagem de Especificação Visual. Então em novembro de 2018 e posteriormente em janeiro de 2020, foi realizada a RSL com a *string* de busca "*computational thinking*" e "*visual language*" no título ou no resumo ou nas palavras chaves do autor. A busca dos trabalhos foi realizada nas principais bases de dados científicas digitais

on-line: ACM (*Association for Computing Machinery*), IEEE (*Institute for Electrical and Electronics Engineers*), *Science Direct*, *Springer* e *Scopus*. Consideraram-se os artigos publicados em qualquer data e anais de conferências internacionais, no idioma inglês.

Na primeira fase, para fazer a seleção dos artigos que se enquadravam no foco da pesquisa, definiram-se critérios de inclusão e exclusão. O critério de inclusão considerado foi: I1 - trabalhos que integravam Pensamento Computacional e alguma linguagem visual. Os critérios de exclusão considerados foram: E1 - trabalhos que utilizavam uma linguagem de programação, e não uma de linguagem de especificação visual; ao invés de linguagem de especificação; E2 - trabalhos que utilizavam apenas um dos termos, PC ou linguagem visual; E3 - trabalhos duplicados; E4 - trabalhos publicados de um mesmo projeto, somente o mais recente foi considerado.

Na segunda fase, efetivamente realizou-se o levantamento dos trabalhos. Aplicando a *string* de busca no período descrito retornaram 90 artigos, vale ressaltar que a maioria destes artigos tratavam de linguagens de programação visuais, os quais foram desconsiderados pelo critério E1. Foram retirados artigos duplicados. Então o critério I1 foi considerado, resultando em nenhum artigo.

2.2.2 Linguagem de Especificação para o Desenvolvimento do Pensamento Computacional

Visto que a revisão anterior não trouxe resultados, partiu-se para outra pesquisa, para saber se existia algum tipo de linguagem de especificação, mesmo que não visual, sendo relacionada ao PC.

Na primeira fase, de planejamento, identificou-se que não havia um levantamento já realizado neste escopo e definiu-se o protocolo da pesquisa. Então em novembro de 2018 e posteriormente em janeiro de 2020, foi realizada a RSL com a *string* de busca "*computational thinking*" e "*specification language*" no título ou no resumo ou nas palavras chaves do autor. A busca dos trabalhos foi realizada nas principais bases de dados científicas digitais *on-line*: IEEE (*Institute for Electrical and Electronics Engineers*), ACM (*Association for Computing Machinery*), *Scopus*, *Science Direct* e *Springer*. Considerou-se os artigos publicados em qualquer data e anais de conferências internacionais, no idioma inglês. O objetivo do levantamento foi o de identificar, avaliar e interpretar pesquisas relevantes que integram PC e Linguagens de Especificação.

Ainda na primeira fase, para fazer a seleção dos artigos que se enquadravam no foco da pesquisa, definiu-se critérios de inclusão e exclusão. O critério de inclusão considerado foi: I1 - trabalhos que integram Pensamento Computacional e alguma linguagem de especificação. Os critérios de exclusão considerados foram: E1 - trabalhos que citam apenas um dos termos; E2 - trabalhos duplicados; E3 - trabalhos publicados de um mesmo projeto, somente o mais recente será considerado.

Na segunda fase, efetivamente realizou-se o levantamento dos trabalhos. Aplicando a *string* de busca no período descrito retornaram 6 artigos. Foram retirados artigos duplicados. Então o critério I1 foi considerado, resultando em 1 artigo. Na sequência, por meio de uma leitura e análise do título, do resumo e das palavras-chave aplicou-se os demais critérios de inclusão e exclusão, resultando em nenhum artigo.

Com o resultado desta revisão, não se encontrou nada no período analisado que integrasse PC e linguagem de especificação visual em alto nível de abstração.

2.3 Metodologias para o Desenvolvimento do Pensamento Computacional

Ainda, na busca de trabalhos relacionados ao foco desta tese, fez-se uma RSL com o objetivo de identificar pesquisas que integrassem PC e metodologias de um modo geral. Então em novembro de 2018 e posteriormente em janeiro de 2020, considerou-se os artigos publicados em qualquer data e anais de conferências internacionais, no idioma inglês. A busca dos trabalhos foi nas principais bases de dados científicas digitais *on-line*: ACM (*Association for Computing Machinery*), IEEE (*Institute for Electrical and Electronics Engineers*), *Science Direct* e *Scopus*. Com a *string* de busca selecionou-se artigos que continham as palavras "*computational thinking*" e "*methodology*" no título ou no resumo ou nas palavras chaves do trabalho.

Ainda na primeira fase, para fazer a seleção dos artigos que se enquadravam no foco da pesquisa, definiu-se critérios de inclusão e exclusão. O critério de inclusão considerado foi: I1 - trabalhos que integram PC e metodologias. Os critérios de exclusão considerados foram: E1 - trabalhos que utilizam apenas um dos termos, PC ou metodologia; E2 - trabalhos duplicados; E3 - trabalhos publicados de um mesmo projeto, somente o mais recente foi considerado; E4 - trabalhos curtos.

Na segunda fase, efetivamente realizou-se o levantamento dos trabalhos. Aplicando a *string* de busca retornaram 162 artigos (Tabela 1). Foi considerado o critério I1 e retirados artigos duplicados, resultando em 42 artigos. Com a leitura na íntegra, resultaram 21 artigos.

As metodologias encontradas nos artigos pesquisados, estão basicamente divididas em três abordagens, metodologia da pesquisa científica, de avaliação e de ensino-aprendizagem. Os artigos que tratam de metodologias de pesquisa, tais como estudo de caso, observação, exploratória, etc, não foram considerados pois não eram o foco deste levantamento, assim como as metodologias de avaliação.

Já os artigos que trazem as metodologias de ensino-aprendizagem geralmente são para desenvolver as habilidades do PC. Dentre estas destacaram-se as metodologias ativas e a programação. Nas metodologias ativas, teve: aprendizagem ba-

Tabela 1 – Classificação por evento dos artigos analisados: PC e Metodologia.

Bases	**Total Busca**	**Análise Título e Resumo**	**Artigos selecionados**
ACM	30	11	8
IEEE	31	17	6
Science Direct	10	4	3
Springer	81	6	1
Scopus	10	4	3
TOTAIS	**162**	**42**	**21**

seada em desafios, que foi utilizada para o desenvolvimento do PC com robôs móveis de transporte e de navegação (GONÇALVES et al., 2019); aprendizagem por pares (SENSKE, 2017); aprendizagem baseada em projetos (SÁEZ-LÓPEZ; ROMÁN-GONZÁLEZ; VÁZQUEZ-CANO, 2016); metodologias expositivas e interativas, centradas no aluno (CESAR et al., 2017); atividades educacionais mediadas pelo aprendizado colaborativo (GONZÁLEZ; MUÑOZ-REPISO, 2017). Diversos trabalhos utilizaram a programação como metodologia de ensino-aprendizagem (ALMEIDA; TEIXEIRA; ALMEIDA, 2019; ROJAS-LÓPEZ; GARCÍA-PEÑALVO, 2018; FIGUEIREDO; GOMES; GARCÍA-PEÑALVO, 2016; HUANG; DENG; RONGSHENG, 2009; SULLIVAN; BERS; PUGNALI, 2017; FLOS; VILAHUR, 2016), alguns focando na prática da programação paralela para a resolução de problemas (CHEN et al., 2018; RODRIGUES et al., 2018), outro no uso de metáforas para ensinar a programar (PÉREZ-MARÍN; HIJÓN-NEIRA; MARTÍN-LOPE, 2018) e também um projeto que usou a programação para notação musical (LUDOVICO; MALCHIODI; ZECCA, 2017).

Além destas, também foram encontradas outras metodologias, tais como: modelo pedagógico que visa unir a criatividade, a colaboração e novas habilidades de alfabetização midiática ao PC (TSORTANIDOU; DARADOUMIS; BARBERÁ, 2019); uma trajetória com treze objetivos de aprendizagem que pode ser usada para a prática da decomposição, e o desenvolvimento de materiais curriculares de avaliação e de desenvolvimento profissional (RICH et al., 2018); algoritmos baseados em cenários (HAREL; MARRON, 2018); agentes de conversação pedagógica que ensinam conversando (URRUTIA et al., 2017); metodologia generalizada para o projeto de construir objetos para pensar computacionalmente (WELLER; DO; GROSS, 2008); aprendizagem distribuída através de ambientes interativos (PAPADOPOULOS; TEGOS, 2012); e uso de pensamento algorítmico para a resolução de problemas em qualquer área, o *AlgoThink* (ALTAHER; FERCHICHI, 2018).

O *AlgoThink* (ALTAHER; FERCHICHI, 2018) propõe uma metodologia para modelar o ensino e a aprendizagem de forma algorítmica. Isto é, uma metodologia para representar qualquer atividade com algoritmos, onde o conceito de algoritmo consiste em uma abordagem, não em uma ferramenta. O autor ainda compara o PC e o *Al-*

goThink, que os dois métodos são duas maneiras distintas de resolver problemas, porém a metodologia proposta consegue atingir o conceito do pensamento algoritmo do PC.

No entanto, a metodologia *AlgoThink* foca em apenas uma das técnicas de resolução de problemas do PC, o pensamento algorítmico. Já neste trabalho busca-se uma abordagem mais ampla, que englobe as principais técnicas do PC: abstração, decomposição, composição, refinamento e generalização.

3 REFERENCIAL TEÓRICO

Este referencial teórico têm como objetivo apresentar uma síntese dos conceitos e técnicas que guiaram este trabalho. Este capítulo foi dividido basicamente em três partes. A seção 3.1 traz uma definição operacional, algumas dimensões e alguns dos conceitos do PC, assim como uma síntese das técnicas que norteiam este trabalho. A seção 3.2 caracteriza diferentes tipos de linguagens computacionais, bem como discute algumas abordagens que embasaram este trabalho. Por fim, a seção 3.3 discute as principais metodologias de ensino-aprendizagem que têm sido adotadas ou propostas para o desenvolvimento do PC.

3.1 Pensamento Computacional

O PC é definido como um processo de resolução de problemas, que inclui (mas não está limitado) as seguintes características/habilidades:

> formulação de problemas de forma que seja possível usar um computador e outras ferramentas para ajudar a resolvê-los; organização lógica e análise de dados; representação de dados por meio de abstrações, como modelos e simulações; automatizações de soluções através do pensamento algorítmico (uma série de passos ordenados); identificação, análise e implementação de soluções possíveis com o objetivo de alcançar a combinação mais eficiente e eficaz das medidas e recursos; generalização e transferência desse processo de resolução de problemas para uma grande variedade de problemas. (ISTE; CSTA, 2011, p. 13)

As dimensões essenciais do PC, incluem as seguintes disposições (ISTE; CSTA, 2011, p. 13): “confiança em lidar com a complexidade; persistência em trabalhar com problemas difíceis; tolerância para a ambiguidade; capacidade de lidar com os problemas em aberto; capacidade de comunicar e trabalhar com outros para atingir um objetivo comum”.

Além disso, foram estabelecidos 9 conceitos (estrutura e vocabulário), que estão relacionados com os conceitos da Ciência da Computação e são considerados impor-

tantes de serem trabalhados para o desenvolvimento do PC (ISTE; CSTA, 2011, p. 14):

> coleta de dados - o processo de reunir informação apropriadas; análise de dados - dar sentido a dados, encontrar padrões e tirar conclusões; representação de dados - descrever e organizar dados em gráficos, mapas, palavras ou imagens apropriadas; decomposição do problema - quebrar as tarefas em partes menores gerenciáveis; abstração - reduzir a complexidade para definir a ideia principal; algoritmos e procedimentos - resolver um problema por meio de uma série de passos ordenados; automação - usar máquinas para fazerem tarefas repetitivas ou tediosas; simulação - executar experimentos usando modelos; paralelização - organizar recursos para, simultaneamente, realizar tarefas para alcançar um objetivo comum.

Muitas destas habilidades destacadas por ISTE e CSTA (2011) também são consideradas importantes para o século XXI. Um desafio que se apresenta é o de trabalhar esses conceitos de forma atraente para o aluno. A linguagem visual LiVE fundamenta-se nas técnicas de refinamento, decomposição, composição, abstração e generalização.

O **refinamento** "...permite construir um modelo gradualmente, tornando-o cada vez mais preciso, mais próximo da realidade" (ABRIAL, 2010). Isto é, através do refinamento detalha-se o problema a ser resolvido, até chegar no nível desejado para sua solução.

A **decomposição** permite compreender os artefatos em termos de suas partes, as quais podem ser resolvidas e avaliadas separadamente. Com isso, problemas mais complexos tornam-se mais fáceis de resolver (CAS, 2015). Assim, na técnica de decomposição, os problemas são quebrados em problemas menores (mais simples de se resolver) que são resolvidos independentemente e cujas soluções são combinadas para se obter a solução do problema maior. Por exemplo, para fazer um bolo com cobertura pode-se dividir a tarefa em atividades menores como: fazer a massa, fazer a cobertura e montar o bolo, onde cada uma dessas atividades também podem ser divididas em atividades menores. "Através da decomposição da tarefa original, cada parte pode ser desenvolvida e integrada posteriormente no processo" (CAS, 2015). Na **composição** da solução como um todo, deve ser levado em conta as inter-relações entre as partes componentes. Além de juntar as várias partes para compor uma solução para o problema inicial, estas também podem ser combinadas de várias formas diferentes (sequencial, paralela, por dependências, etc) (RIBEIRO; FOSS; CAVALHEIRO, 2017).

Abstração é um processo que permite decidir quais detalhes iremos destacar e

quais podemos ignorar (WING, 2008). Por exemplo, tendo um problema para solucionar, foca-se em alguns pontos que são considerados mais importantes, isolando as etapas que levam a sua solução em partes menores, de forma a simplificá-lo (PAPERT, 1994). Para Wing (2008) “abstrações são as ferramentas ‘mentais’ da computação”, que ajudam na resolução de problemas. A abstração da Ciência da Computação (CC) é mais rica e complexa que a da matemática e das ciências físicas. Pois as abstrações da CC “... são extremamente gerais, porque elas são simbólicas, onde abstrações numéricas são apenas um caso especial” (WING, 2008). Na computação, o processo de abstração é feito em camadas, num trabalho “... simultâneo com pelo menos duas camadas de abstração: a camada de interesse e a camada inferior; ou a camada de interesse e a camada superior” (WING, 2008).

Para CAS (2015) a **generalização** está associada à identificação de padrões, onde identifica características semelhantes de soluções de diferentes problemas e reaplica a mesma técnica para solução de novos problemas.

3.2 Linguagens Computacionais

Nesta seção são caracterizados diferentes tipos de linguagens computacionais como linguagens textuais e visuais, de programação e de especificação. Além disso, são apresentadas algumas abordagens que embasaram este trabalho.

Uma linguagem textual, no contexto da Computação, é formada por um conjunto de instruções básicas e pré-definidas da linguagem (RIBEIRO; FOSS; CAVALHEIRO, 2017), com regras sintáticas e semânticas, para a escrita de algoritmos (sequência de passos para se atingir um objetivo) e geração de código fonte ou para a escrita da especificação formal na análise dos requisitos necessários para construção do sistema. Já uma linguagem visual é composta por um conjunto de diagramas que formam uma sentença que facilita a comunicação (MARRIOTT; MEYER, 1998). Linguagens visuais já vêm sendo usadas desde a pré-história até os dias de hoje, abrangendo desde as artes plásticas até a comunicação técnica, como a música, a matemática e mapas (MARRIOTT; MEYER, 1998). Com o avanço das tecnologias a sua importância na área da Computação aumentou, tornando-se “um componente chave na interação entre homem-computador” (MARRIOTT; MEYER, 1998, p. 1), e presente nas diferentes mídias de hoje. As linguagens visuais podem ser classificadas basicamente em 3 tipos (GOLIN; REISS, 1990; INTRODUCTION: VISUAL LANGUAGES AND ICONIC LANGUAGES, 1986; SHU, 1986): linguagens visuais para programar a realidade com expressões visuais; linguagens visuais para o processamento de informações visuais; e linguagens visuais para apoiar interações visuais.

Uma linguagem de programação descreve um conjunto de instruções que podem ser usadas para construir um programa e que dizem ao computador o que deve ser

feito e como. As Linguagens de Programação Visual (*Visual Programming Language* - VPL) também podem ser classificadas em três grupos de acordo com a extensão da expressão visual usada (SHU, 1986): linguagens baseadas em ícones, linguagens baseadas em formulários e linguagens de diagramas. Nas linguagens de programação tradicionais, as estruturas das linguagens são baseadas em representações unidimensionais e textuais, já nas VPLs são usadas mais de uma dimensão para descrever tais estruturas (BURNETT, 1999). A autora também considera que uma linguagem de programação é uma VPL quando a sua sintaxe inclui expressões visuais multidimensionais (diagramas, esboços a mão livre, ícones ou demonstrações de ações executadas por objetos gráficos).

Uma linguagem de especificação é uma linguagem formal, isto é, uma linguagem com sintaxe e semântica definidas matematicamente, que permite descrever a solução de um problema em um alto nível de abstração. Uma especificação descreve o que deve ser feito e não como, o que a diferencia de um programa que deve considerar todos os detalhes da implementação, descrevendo como resolver o problema.

Na computação existem várias linguagens de especificação visual para a solução de problemas em um alto nível de abstração. Aqui serão apresentadas brevemente as três linguagens que serviram como base para a definição da linguagem proposta neste trabalho: diagrama de fluxo de dados (DFD), diagrama de atividades da UML e Simulink.

O DFD é uma “linguagem de modelagem de processos usada amplamente na fase de análise de requisitos estruturados” (MENG; CHU; ZHAN, 2010, p. 1). Um DFD mapeia o fluxo de informações de um processo ou sistema. Ele possui representações gráficas para identificar entradas e saídas de dados, processos que transformam os dados, pontos de armazenamento e o limite entre o que pertence ao sistema e o que está fora dele. Um DFD é composto por quatro componentes: processo (funções, atividade de transformação da informação), fluxo de dados, entidade externa (entidade de origem ou destino da informação) e depósito de dados (armazena dados).

Um diagrama de atividades da UML, “é uma linguagem de modelagem de processos usada amplamente em várias fases do método de desenvolvimento orientado a objetos” (MENG; CHU; ZHAN, 2010, p. 1). O diagrama de atividades do UML ilustra uma sequência de atividades que são modeladas para representar os aspectos dinâmicos de um processo computacional. Este modelo detalha o fluxo de controle e o fluxo de dados de uma determinada atividade, mostrando as ramificações de controle e as situações onde existem processamento paralelo (OMG, 2017). As atividades são comportamentos que modelam uma execução não-atômica, já uma ação representa um processamento atômico (ação que deve ser executada completamente em caso de sucesso, ou ser abortada completamente em caso de erro) (BOCK, 2003). As atividades coordenam as ações. Esta coordenação é capturada como um grafo, onde os

nós representam as ações e são conectados por arestas. O fluxo de dados é representado por nós de objetos e por arestas de fluxo de dados, que são subclasses dos nós de atividade e das arestas, respectivamente (BOCK, 2003).

Simulink é um ambiente gráfico de modelagem e simulação para sistemas dinâmicos, através da criação de diagramas de blocos, em que os blocos representam partes de um sistema, tais como um componente físico, um pequeno sistema ou uma função (MATHWORKS, 2020). O Simulink suporta modelos de fluxo de dados e de tempo contínuo (FOSS et al., 2013, p. 103). Um modelo Simulink consiste em três componentes: as fontes que são as entradas do sistema; os diagrama de blocos que é a modelagem por meio de blocos; e os dispositivos de saídas que são os blocos que permitem verificar o comportamento do sistema. Os dados têm três categorias (MATHWORKS, 2020): sinais, que bloqueiam entradas e saídas; estado, que são valores internos representando a dinâmica do bloco; e parâmetros, que são valores que afetam o comportamento de um bloco. Os sinais e o estado são computados durante a simulação, já os parâmetros são controlados pelo usuário, que os definem no momento de criação do modelo e pode alterá-los durante a execução da simulação.

A Tabela 2 sintetiza algumas das características presentes nestas três linguagens. Todas elas possuem descrições para processos, fluxo de dados e regras de decisão, bem como, representam as estruturas de repetição por meio de ciclos. Além disso, enquanto em DFD e UML o fluxo de controle é descrito de forma explícita, no Simulink ele é inferido pela dependência dos sinais. Diferente do UML e do Simulink, o DFD possui uma representação para o armazenamento de dados. Já o Simulink é o único que possui a representação de composição hierárquica e tipos de dados (restritos aos tipos do MatLab).

Tabela 2 – Características das linguagens selecionadas.

Características/Linguagens	**DFD**	**UML**	**Simulink**
Processos/ações	X	X	X
Fluxo de dados	X	X	X
Armazenamento de dados	X	-	-
Regras de decisão	X	X	X
Estrutura de repetição	ciclos	ciclos	ciclos
Fluxo de controle	explícito	explícito	implícito
Composição hierárquica	-	-	X
Tipos de dados	-	-	restritos ao MatLab

3.3 Metodologias de Ensino-aprendizagem

Metodologia, conforme o dicionário, é o estudo de um conjunto de métodos, regras ou caminhos para se chegar a um determinado objetivo. As metodologias de ensino-aprendizagem estudam a aplicação de métodos no processo de ensino para pesquisar

a melhor forma de aprendizagem.

Dentre as metodologias de ensino-aprendizagem, as mais frequentes na revisão de literatura (seção 2.3) são as metodologias ativas, que são centradas nos alunos, onde estes se tornam protagonistas do processo de construção do conhecimento (VALENTE; ALMEIDA; GERALDINI, 2017). Estas metodologias envolvem a combinação do ensino com projetos, desafios, resolução de problemas, trabalhos em grupo, jogos, entre outros; os estudantes aprendem fazendo e também aprendem juntos e no próprio ritmo, tornando a construção do conhecimento mais significativa e contextualizada (MORAN, 2015). Entre as metodologias ativas, temos: Aprendizagem Baseada em Projetos (*Project-Based Learning*), Aprendizagem Baseada em Problemas (*Problem-Based Learning*), Metodologias Baseada em Desafios (*Challenge Based Learning* - CBL), Aprendizagem por Pares (*Peer Instruction* - PI), entre outras. Aqui será feita uma breve introdução das metodologias citadas, que vão ao encontro deste trabalho, embora existam outras.

A metodologia de Aprendizagem Baseada em Projetos "é um modelo de ensino que consiste em permitir que os alunos confrontem as questões e os problemas do mundo real que consideram significativos, determinando como abordá-los e, então, agindo de forma cooperativa em busca de soluções" (BENDER, 2014, p. 9) gerando assim um produto final. O aluno é um sujeito ativo, cooperativo e integrado no processo de ensino-aprendizagem, onde aplica o conhecimento de forma crítica e reflexiva na resolução de problemas tanto na vida acadêmica quanto profissional. Ele têm o controle de como o projeto terminará e também sobre o produto final (SAVERY, 2015). Bender (2014) ainda afirma que a Aprendizagem Baseada em Projetos, pelas características anteriormente citadas, "é uma técnica de ensino do século XXI" (BENDER, 2014, p. 16).

Na metodologia de Aprendizagem Baseada em Problemas os alunos são estimulados a resolução de problemas que fazem parte do tema de estudo e de sua realidade, e que frequentemente não há uma resposta correta. Esta aprendizagem "capacita os alunos a conduzir pesquisas, integrar teoria e prática e aplicar conhecimentos e habilidades para desenvolver uma solução viável para um problema definido" (SAVERY, 2015), gerando uma solução possível.

Nas metodologias de Aprendizagem Baseadas em Projetos e Problemas ambas têm o aluno como protagonista, são guiadas pelo professor e conectam as soluções com o mundo real. Na Aprendizagem Baseada em Projetos o professor define um problema e estabelece as etapas para a sua solução. As atividades podem ser realizadas individualmente ou em grupos, gerando um produto final. Já na Aprendizagem Baseada em Problemas os estudantes geralmente trabalham em grupos para resolver um problema, buscando uma das possíveis soluções (SAVERY, 2015).

A metodologia de Aprendizagem Baseada em Desafios (CBL) foi inspirada na

Aprendizagem Baseada em Problemas, pois motiva os alunos a resolverem problemas reais da sociedade, mas através do uso de tecnologias (MORESI et al., 2019). A CBL é dividida em três etapas: “envolver, investigar e agir, que pode variar em duração e intensidade e ser incorporada ou adaptada para a maioria dos ambientes de aprendizagem” (MORESI et al., 2019, p. 65). Na primeira etapa (envolver) os alunos passam por questionamentos e a partir de uma grande ideia, chega-se a um desafio concreto e executável (NICHOLS; CATOR; TORRES, 2016). Na segunda etapa (investigar) os alunos planejam e criam bases para a solução do problema, que devem atender aos requisitos acadêmicos pré-estabelecidos (NICHOLS; CATOR; TORRES, 2016). Na terceira etapa (agir) são desenvolvidas soluções e implementadas com um público real e depois avaliadas (NICHOLS; CATOR; TORRES, 2016).

A Aprendizagem por Pares é uma metodologia em que a aprendizagem acontece na interação entre alunos de forma colaborativa, onde todos os envolvidos trabalham na mesma questão para atingir um objetivo comum. Os alunos estudam previamente em fontes primárias o conteúdo a ser abordado em sala de aula e se envolvem em seu próprio aprendizado. Os conceitos são trabalhados por meio de perguntas e interação com colegas (MAZUR, 2015).

Este trabalho não abordou uma das metodologias ativas em específico, mas seguiu algumas de suas características, tanto na proposta das metodologias (capítulo 5), quanto na aplicação do estudo de caso (capítulo 6). As principais características consideradas foram o foco no aluno e na comunicação entre eles.

4 LINGUAGEM DE ESPECIFICAÇÃO VISUAL LiVE

Dentre os diversos trabalhos encontrados nas RSLs do capítulo 2, a maioria utilizam VPLs. O *Scratch* foi a mais utilizada. Giordano (GIORDANO; MAIORANA, 2015, p. 1) acredita que estes ambientes de programação visual têm "... a vantagem de evitar problemas sintáticos e permitir que os alunos se concentrem nas atividades de resolução de problemas".

Shu (SHU, 1986, p. 12) também concorda, pois cita alguns fatores que estimulam o ensino com VPLs, que são: "as pessoas, em geral, preferem figuras a palavras; as imagens são mais poderosas que as palavras como meio de comunicação; e as imagens não possuem as barreiras linguísticas das linguagens naturais (elas são entendidas pelas pessoas, independentemente do idioma que falam)". A partir disso, acredita-se que a introdução do PC é estimulado por estas linguagens, por facilitar o entendimento, principalmente de quem está começando a solucionar problemas utilizando técnicas da computação.

Em geral, por ser uma linguagem de programação, uma VPL ajuda a resolver problemas em baixo nível de abstração, o que dificulta a identificação de quais habilidades do PC o aluno realmente desenvolveu, se é que desenvolveu alguma, e se ele conseguirá fazer uso destas habilidades para resolver diferentes problemas. Assim, este trabalho propõe uma linguagem visual não para programar, mas sim para especificar uma solução em um nível de abstração mais alto. A linguagem denominada LiVE (Linguagem Visual de Especificação), aqui proposta, permite usar de forma explícita diferentes técnicas da computação para resolver problemas, sem a necessidade de preocupar-se com detalhes específicos de implementação. Além disso, a linguagem LiVE é simples e concisa, e pode ser usada por não especialistas e em qualquer área do conhecimento.

LiVE tem uma descrição para as ações e para o fluxo de dados, que gera uma dependência entre estas ações. Ela utiliza o conceito de parâmetros, representado por portas. As portas são similares a endereços de memória que não precisam ser mantidos de forma direta, pois são tratadas apenas como um meio de passar a informação. A linguagem LiVE acaba mantendo o essencial para representar uma sequência de

passos e as relações entre estas dependências, descrevendo de forma abstrata a solução do problema. Ela também facilita o uso das metodologias propostas, pois é simples o suficiente para representar a solução em si, sem sobrecarregar com outros detalhes que dizem respeito só a implementação.

Tanto linguagens de programação quanto de especificação podem ser formalmente definidas, o que garante definições exatas tanto da sintaxe quanto da semântica de cada uma. Isso é importante para que não haja dúvida a respeito do que um programa ou uma especificação está descrevendo. A seção 4.1 apresenta a sintaxe da linguagem LiVE e a seção 4.2 define a sua semântica.

4.1 Sintaxe

A linguagem LiVE foi inspirada nas linguagens de especificação visual citadas na seção 3.2, DFDs (MENG; CHU; ZHAN, 2010; WARD, 1986), Diagramas de Atividades da UML (PRESSMAN, 2011) e Simulink (MATHWORKS, 2020).

A linguagem proposta possui o conceito de processos (ações) e de fluxo de dados como no DFD, mas difere nos conceitos de armazenagem, a qual é abstraída nesta proposta. Além disso, a linguagem LiVE permite a representação de loops por meio da recursão, diferentemente dos DFDs. Em relação aos diagramas de atividades da UML, a linguagem LiVE se assemelha na caracterização das ações e do fluxo de dados, diferindo por não apresentar o fluxo de controle de forma explícita (em LiVE o fluxo de controle é inferido pelo fluxo de dados). Também há uma diferença na representação de repetições, as quais são descritas em UML por ciclos dentro do fluxo de controle ao invés de utilizar a recursão, como em LiVE. LiVE é fortemente inspirado no Simulink, onde blocos representam funções que recebem dados por meio de portas e são interconectados através de links. O Simulink, assim como LiVE, possui uma característica de composição hierárquica, onde um componente pode ser decomposto em outros componentes interconectados, o que permite a representação de sistemas complexos em diferentes níveis de abstração. Por sua vez, os laços de repetição no Simulink também são representados por ciclos ao invés de recursão. A principal diferença incorporada na linguagem LiVE é permitir os mais diversos tipos de dados, enquanto o Simulink está restrita aos tipos de dados do Matlab (MATHWORKS, 2020).

A principal diferença, entre todas estas linguagens, é que LiVE não tem o foco no desenvolvimento de sistemas de computação, como os DFDs, os diagramas de atividades da UML e o Simulink. Ela propõe a resolução de problemas em qualquer área do conhecimento e sem os detalhes da implementação. Ademais, em LiVE foi adicionada a possibilidade de abstração e refinamento dos dados, ausentes nestas outras linguagens.

Uma especificação na linguagem LiVE descreve a solução de um problema. Ela

inclui um conjunto de componentes elementares que descrevem de forma abstrata todas as ações que podem ser realizadas. Um destes componentes é o principal, o qual descreve a ação que soluciona o problema como um todo.

Um Componente Elementar (CE), ilustrado na Figura 1, descreve uma ação a ser executada sem detalhá-la, apenas especificando os tipos das informações de entrada necessárias para sua execução, bem como os tipos dos resultados obtidos. Um CE (representado por um retângulo) é definido por um nome, descrevendo a ação, uma lista de portas de entrada e uma lista de portas de saída (quadrados pretos na borda superior e inferior dos componentes, respectivamente). As portas de entrada/saída são os meios pelos quais as informações/resultados são recebidos/enviados pelo CE. Por sua vez, cada porta possui um tipo que restringe o tipo de elementos que pode ser associado a ela.

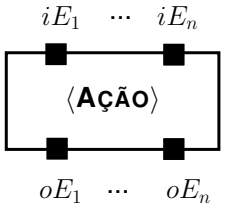


Figura 1 – Componente Elementar

Caso sejam necessários mais detalhes, um componente elementar pode ser associado a um componente decomposto que define tais detalhes. O Componente Decomposto (CD), ilustrado na Figura 2, é detalhado por conjuntos de instâncias de CE, nós condicionais, nós de refinamento e nós de abstração. Estes elementos são ligados por conexões entre suas portas, estabelecendo caminhos por onde os dados circulam. Cabe salientar que quando referenciam-se as portas de entrada/saída de um CD, na verdade se estão referenciando as respectivas portas do CE associado.

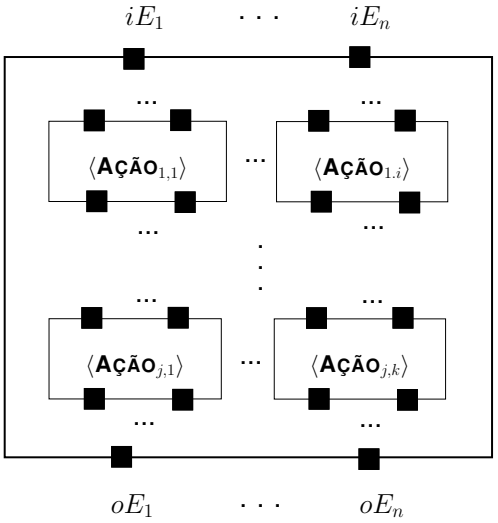


Figura 2 – Componente Decomposto.

Um nó condicional descreve uma escolha entre dois possíveis caminhos, pelos quais os dados podem seguir. Estes caminhos são delimitados por um nó de decisão (no início) e um nó de junção (no final), conforme as Figuras 3 e 4, respectivamente. Ao chegar em um nó de decisão (através das portas $\langle i_1, \ldots, i_n \rangle$), os dados são enviados por um dos caminhos (através das portas $\langle oT_1, \ldots, oT_n \rangle$ ou $\langle oF_1, \ldots, oF_n \rangle$), determinado por um valor booleano ($true$ ou $false$, respectivamente), recebido através da porta i_b. Os dados chegam em um nó de junção por um dos dois caminhos (através das portas $\langle iT_1, \ldots, iT_n \rangle$ ou $\langle iF_1, \ldots, iF_n \rangle$) e seguirão o seu fluxo (através das portas $\langle o_1, \ldots, o_n \rangle$). Deve existir uma equivalência de tipos entre todas as portas de entrada e todas as portas de saída, tanto para o nó de decisão quanto para o de junção do mesmo condicional.

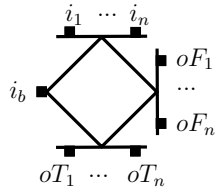


Figura 3 – Nó de decisão.

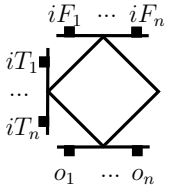


Figura 4 – Nó de junção.

As informações de entrada ou saída de um componente decomposto também podem ser mais detalhadas ou abstraídas, respectivamente. Tais detalhes são descritos por **nós de refinamento e de abstração** (representadas por barras horizontais, conforme as Figuras 5 e 6), os quais têm a função de detalhar e abstrair os dados, respectivamente. Um nó de refinamento permite que um tipo de dado mais abstrato (i) seja detalhado em uma lista de tipos mais concretos ($\langle oR_1 \cdots oR_n \rangle$). Um nó de abstração permite que uma lista de tipos mais concretos ($\langle iA_1 \cdots iA_n \rangle$) sejam abstraídos num tipo mais abstrato (o).

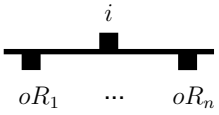


Figura 5 – Nó de refinamento.

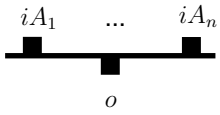


Figura 6 – Nó de abstração.

Os componentes até aqui apresentados são definidos formalmente no restante desta seção. As definições são apresentadas de forma incremental, partindo dos componentes mais básicos até a definição de uma especificação na linguagem LiVE.

Notação 1. *Seja l uma lista. O j-ésimo elemento de l é denotado por $l[j]$ e o tamanho de l é denotado por $|l|$. Se $x \in l$, então $x = l[i]$ para algum $i \in \{1, \ldots, |l|\}$. A função $lSet : Lista \to Set$ recebe uma lista e retorna um conjunto contendo todos os elementos desta lista. A operação $\bullet$ denota concatenação de listas e quando se*

diz que uma lista é vazia, denotada por ε, significa que todas suas posições estão indefinidas.

Uma porta é definida por um nome e um tipo, o qual restringe o tipo de dados que pode passar por ela. Um tipo é constituído por um conjunto de valores que o determinam.

Definição 1 (Porta)**.** *Seja $\mathcal{T}$ um conjunto de tipos e P um conjunto de nomes, uma **porta** de entrada/saída é uma dupla $p = \langle n, T \rangle$ onde:*

- $n \in P$ *é o nome da porta.*
- $T \in \mathcal{T}$ *é o tipo da porta.*

□

Cabe observar que, nas definições que seguem, quando há referência a "uma" porta, na verdade está sendo considerado uma lista com uma única porta.

Exemplo 1. ***(Porta)****. A Figura 7 mostra um exemplo de CE que descreve a ação* **Fazer Bolo com Cobertura***. Neste componente há três portas de entrada:*

$\langle tm, string \rangle$
$\langle ig, ingMa \times ingCo \rangle$
$\langle ut, utMa \times utMo \times utCo \rangle$

e duas portas de saída:

$\langle bolo, boloCoberto \rangle$
$\langle ut', utMa \times utMo \times utCo \rangle$

*Os primeiros elementos dos pares descrevem os nomes das portas e os segundos elementos descrevem os tipos de dados que podem passar por elas. O tipo da primeira porta de entrada define que os dados que chegarão nela são strings (sequência de caracteres delimitados por " "). Já o tipo da segunda porta de entrada restringe os dados a elementos do tipo (conjunto) $ingMa \times ingCo$[1]**, tal que os tipos $ingMa$ e $ingCo$ podem ser definidos por:*

$ingMa = \{IngM1, IngM2, IngM3\}$**, onde**
$IngM1 = \{acúcar, ovo, leite, farinha, óleo, chocolate\}$

[1]Este tipo é definido por todas as possibilidades de pares de valores de ingredientes de massa ($ingMa$) e de cobertura ($ingCo$).

$IngM2 = \{acúcar, ovo, leite, farinha, margarina, chocolate, baunilha\}$
$IngM3 = \{acúcar, ovo, leite, farinha, manteiga, baunilha\}$

$ingCo = \{IngC1, IngC2\}$, *onde*
$IngC1 = \{leite, chocolate, acúcar\}$
$IngC2 = \{leite\ condensado, morango\}$

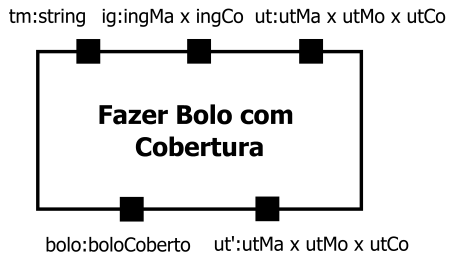


Figura 7 – CE Fazer Bolo com Cobertura.

Um nó de decisão é definido por uma lista de n portas de entrada ($\langle i_1, \cdots, i_n \rangle$), que recebem os dados de entrada, uma porta de entrada booleana (i_b), uma lista de n portas de saída associadas ao valor *true* ($\langle oT_1, \cdots, oT_n \rangle$) e uma lista de n portas de saída associadas ao valor *false* ($\langle oF_1, \cdots, oF_n \rangle$). Um nó de junção é definido por duas listas de n portas de entrada, ($\langle iT_1, \cdots, iT_n \rangle$ e $\langle iF_1, \cdots, iF_n \rangle$) e uma lista de n portas de saída ($\langle o_1, \cdots, o_n \rangle$). Um nó condicional, é composto por uma identificação, um nó de decisão e um nó de junção. Desta forma, em um condicional, cada nó de decisão está associado a um nó de junção. Além disso, deve existir uma equivalência de tipos entre todas as portas de entrada e todas as portas de saída, tanto para o nó de decisão quanto para o de junção, isto é, as portas i_j, oT_j e oF_j, $j = 1..n$, devem ter o mesmo tipo, bem como as portas iT_k, iF_k e o_k, $k = 1..n$.

Definição 2 (Nós de decisão, de junção e condicional)**.** *Um nó de* ***decisão*** *é definido pela tupla* $\langle i_b, i, oT, oF \rangle$*, onde* i_b *é uma porta de entrada, com* $i_b[2] = \{true, false\}$*;* i *é uma lista de portas de entrada;* oT *e* oF *listas de portas de saída associadas aos valores* $true$ *e* $false$*, respectivamente, tal que* $|i| = |oT| = |oF|$ *e* $i[j][2] = oT[j][2] = oF[j][2]$*, com* $j = 1..|i|$*.*

Um ***nó de junção*** *é definido pela tupla* $\langle iT, iF, o \rangle$*, onde* iT *e* iF *são listas de portas de entrada e* o *é uma lista de portas de saída, tal que* $|iT| = |iF| = |o|$ *e* $iT[j][2] = iF[j][2] = o[j][2]$*, com* $j = 1..|o|$*.*

Um ***nó condicional*** *é uma tupla* $\langle k, d, j \rangle$*, onde* $k \in \mathbb{N}$*,* d *é um nó de decisão e* j *é um nó de junção.*

□

Exemplo 2. ***(Nós de decisão, de junção e condicional)****. A Figura 8 mostra um exemplo de CD que detalha a ação* **Fazer Bolo com Cobertura***. Neste componente há apenas uma nó condicional que é definido pela tupla* $\langle 1, d1, j1\rangle$ *onde*

- 1 *é a identificação do nó condicional.*
- $d1 = \langle ib, i, oT, oF\rangle$ *é o nó de decisão, com*

  $ib = \langle b, bool\rangle$*,*

  $i = \langle\langle ima, ingMa\rangle, \langle uma, utMa\rangle\rangle$*,*

  $oT = \langle\langle imaT, ingMa\rangle, \langle umaT, utMa\rangle\rangle$ *e*

  $oF = \langle\langle imaF, ingMa\rangle, \langle umaF, utMa\rangle\rangle$

- $j1 = \langle iT, iF, o\rangle$ *é o nó de junção, com*

  $iT = \langle\langle mb, massaBolo\rangle, \langle umaT', utMa\rangle\rangle$*,*

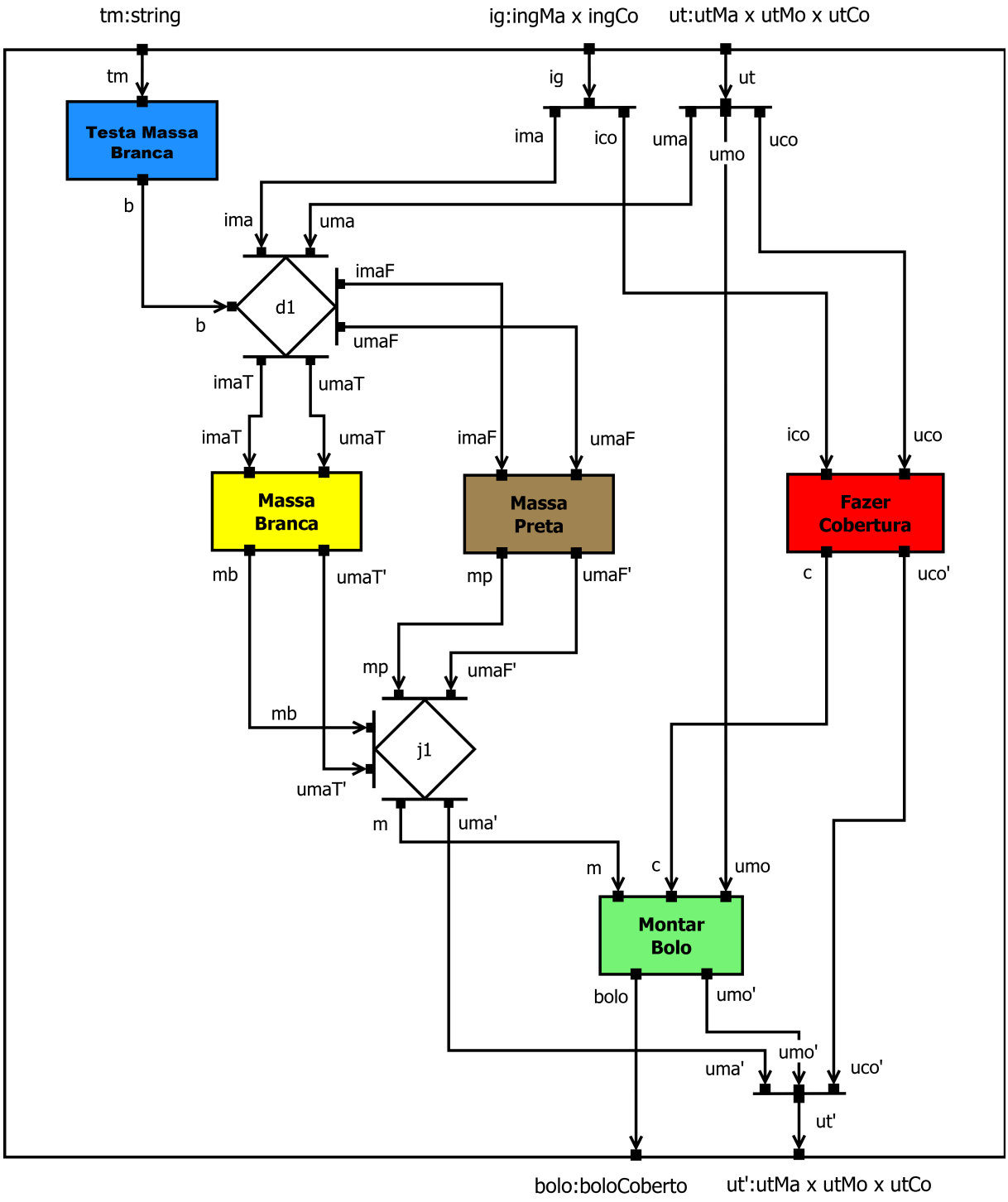
  $iF = \langle\langle mp, massaBolo\rangle, \langle umaF', utMa\rangle\rangle$*,*

  $o = \langle\langle m, massaBolo\rangle, \langle uma', utMa\rangle\rangle$

$utMa = \{UtM1, UtM2\}$*, onde*
$UtM1 = \{forma, batedeira, talheres, xícara\}$
$UtM2 = \{forma, liquidificador, talheres, xícara\}$
$massaBolo = \{massa\_branca1, massa\_branca2, massa\_branca3, massa\_preta1, massa\_preta2, massa\_preta3\}$

Esse nó condicional permite escolher entre fazer uma **Massa Branca** *ou fazer uma* **Massa Preta** *para o bolo. Essa escolha é determinada pelo valor booleano recebido pela porta* ib *do nó de decisão, que define se os dados seguirão pelas portas de saída TRUE (*oT*) ou pelas portas de saída FALSE (*oF*), chegando nos componentes elementares* **Massa Branca** *ou* **Massa Preta***, respectivamente. O nó de junção recebe as informações provenientes ou do CE* **Massa Branca** *ou do CE* **Massa Preta** *(através da portas* iT *ou* iF*, respectivamente) para levar o resultado para o próximo CE* **Montar Bolo** *e dar continuidade à receita.*

Um nó de refinamento é definido por uma porta de entrada e uma lista de portas de saída. Um nó de abstração é definido por uma lista de portas de entrada e por uma porta de saída. Além disso, ambos são identificados por um nome. Tanto em um nó de refinamento quanto em um nó de abstração deve haver compatibilidade de tipos entre entradas e saídas. Isto é, em um refinamento (respect. abstração) a entrada



Os tipos das portas dos componentes internos do CD são dados por: $tm \in string$; $b \in bool$; $ima, imaT, imaF \in ingMa$; $uma, umaT, umaT', umaF, umaF', uma' \in utMa$; $mb, mp, m \in massaBolo$; $ico \in ingCo$; $uco, uco' \in utCO$; $c \in cobertura$; $umo, umo' \in utMo$; $bolo \in boloCoberto$.

Figura 8 – CD associado ao CE Fazer Bolo com Cobertura (Figura 7).

(respect. saída) deve ser uma tupla de valores dos tipos da saída (respect. entrada). A representação visual para os nós de refinamento e abstração está ilustrada nas Figuras 5 e 6, respectivamente. Para não sobrecarregar a notação, a identificação

das portas dos nós (refinamento, abstração e condicionais) podem ser omitidas nas figuras.

Definição 3 (Nós de refinamento e de abstração)**.** *Um nó de **refinamento** é uma tupla $\langle n, i, oR\rangle$, onde $n \in \mathbb{N}$ é o identificador do nó, i é a porta de entrada e oR é uma lista de portas de saída, tal que, $i[2] = oR[1][2] \times \ldots \times oR[k][2]$, com $k = |oR|$.*

*Um nó de **abstração** é uma tupla $\langle n, iA, o\rangle$, onde $n \in \mathbb{N}$ é o identificador do nó, iA é uma lista de portas de entrada e o é uma porta de saída, tal que, $o[2] = iA[1][2] \times \ldots \times iA[k][2]$, com $k = |iA|$.*

□

Exemplo 3. ***(Nós de refinamento e de abstração)**. O CD, ilustrado na Figura 8, contém dois nós de refinamento: um para refinar os ingredientes e outro para refinar os utensílios. Estes nós estão conectados às portas de entrada do CD. O nó de refinamento dos ingredientes é definido pela tupla $\langle 1, i_1, oR_1\rangle$ onde*

- 1 *é o identificador do nó.*
- $i_1 = \langle ig, ingMa \times ingCo\rangle$ *é a porta de entrada por onde são recebidos todos os ingredientes. Os dados recebidos por essa porta são pares contendo os ingredientes da massa e os da cobertura.*
- $oR_1 = \langle\langle ima, ingMa\rangle, \langle ico, ingCo\rangle\rangle$ *é a lista de portas de saída do nó, por onde os ingredientes seguem o fluxo separados em ingredientes da massa e ingredientes da cobertura.*

Neste mesmo CD, há um nó de abstração conectado a uma de suas portas de saída. Este nó é definido pela tupla $\langle 1, iA, o\rangle$ onde

- 1 *é o identificador do nó.*
- $iA = \langle\langle uma', utMa\rangle, \langle umo', utMo\rangle, \langle uco', utCo\rangle\rangle$ *é a lista de portas de entrada do nó de abstração, onde são recebidos os diferentes tipos de utensílios (os usados para fazer a massa, para montar o bolo e para fazer cobertura).*
- $o = \langle ut', utMa \times utMo \times utCo\rangle$ *é a porta de saída do nó por onde os utensílios serão todos agrupados em um único dado (tupla) que é enviado como resultado do CD.*

onde os tipos $utMo$ e $utCo$ são definidos como segue:
$utMo = \{\{faca, prato\}, \{espátula, travessa\}\}$
$utCo = \{\{colher, panela, medidor\}\}$

Um **Componente Elementar** é definido por um nome de ação, uma lista de portas de entrada e uma lista de portas de saída.

Definição 4 (Componente Elementar (CE))**.** *Dado um conjunto N de nomes, um **Componente Elementar** é definido pela tupla $\langle n, iE, oE\rangle$ onde $n \in N$ é o nome da ação, iE é uma lista de portas de entrada e oE é uma lista de portas de saída, tal que:*

- $\forall p_1, p_2 \in iE.\ p_1[1] = p_2[1] \rightarrow p_1 = p_2$*;*
- $\forall p_1, p_2 \in oE.\ p_1[1] = p_2[1] \rightarrow p_1 = p_2$*;*
- $p_1 \in iE \wedge p_2 \in oE \rightarrow p_1[1] \neq p_2[1]$*.*

□

As restrições na definição anterior garantem que todas as portas de um CE tem nomes diferentes.

Exemplo 4. ***(Componente Elementar (CE))****. A Figura 7 ilustra o CE* **Fazer Bolo com Cobertura** *que é definido por:*

$\langle Fazer\ Bolo\ com\ Cobertura, iE, oE\rangle$ *onde*
$iE = \langle\langle tm, string\rangle, \langle ig, ingMa \times ingCo\rangle, \langle ut, utMa \times utMo \times utCo\rangle\rangle$
$oE = \langle\langle bolo, boloCoberto\rangle, \langle ut', utMa \times utMo \times utCo\rangle\rangle$

A seguir é definida uma relação que será útil para a definição de uma especificação na linguagem, bem como para a descrição das funções semânticas. Essa relação associa cada componente elementar a listas de valores para suas portas de entrada e de saída, considerando todas as combinações de valores possíveis.

Definição 5 (Valores de Portas de CEs)**.** *Dado um conjunto de componentes elementares $\mathcal{E}$, define-se a seguinte relação:*

- $RelIO = \{\langle e, lvi, lvo\rangle \mid e \in \mathcal{E} \wedge lvi[k] \in e[2][k][2] \wedge lvo[j] \in e[3][j][2] \wedge 1 \leq k \leq |e[2]| \wedge 1 \leq j \leq |e[3]|\}$ *é uma relação que associa cada elementar as suas possíveis listas de valores de entrada e de saída.*

□

A propriedade da definição de $RelIO$ garante que o k-ésimo valor de entrada é do tipo da k-ésima porta de entrada, assim como o j-ésimo valor de saída é do tipo da j-ésima porta de saída. Um CE pode ser detalhado por um componente decomposto (CD). O CD é definido por um conjunto de instâncias de CE, um conjunto de nós condicionais, um conjunto de nós de refinamento, um conjunto de nós abstração, um

conjunto de conexões e por duas funções que associam a cada conexão a sua porta de origem e de destino. Uma instância de um CE é uma cópia desse diferindo do mesmo por índice que o identifica univocamente.

Definição 6 (Componente Decomposto (CD))**.** *Dado um CE* $ce = \langle n, iE, oE \rangle$ *sua decomposição é definida pela tupla* $\langle E, Cond, R, A, C, sc, tc \rangle$*, denominada* ***Componente Decomposto****, onde:*

- E *é um conjunto não vazio de instâncias de componentes elementares* $e_i = \langle n_i, iE_i, oE_i \rangle$*, tal que* $i \in \mathbb{N}$*;*
- $Cond$ *é um conjunto de nós condicionais;*
- R *é um conjunto de nós de refinamentos;*
- A *é um conjunto de nós abstração;*
- C *é um conjunto de conexões;*
- $sc : C \to Componente \times \mathbb{N} \times \mathbb{N}$ ***e*** $tc : C \to Componente \times \mathbb{N} \times \mathbb{N}$ *são funções totais que associam cada conexão a sua origem e destino, respectivamente*[2]*. A origem ou destino de uma conexão é dada por um componente e uma porta deste componente (a porta é identificada por dois naturais: o primeiro que indica a posição da lista de portas no componente e o segundo que indica a posição da porta dentro da lista), onde*

$$Componente = E \cup R \cup A \cup D \cup J \cup \{ce\}$$

$$D = \{\langle k, i_b, i, oT, oF \rangle \mid \langle k, d, j \rangle \in Cond \wedge d = \langle i_b, i, oT, oF \rangle\}$$

$$J = \{\langle k, iT, iF, o \rangle \mid \langle k, d, j \rangle \in Cond \wedge j = \langle iT, iF, o \rangle\}$$

$$sc(c)[2] = \begin{cases} 2 & \textit{se } sc(c)[1] = ce \\ 4 & \textit{se } sc(c)[1] \in J \\ 4 \textit{ ou } 5 & \textit{se } sc(c)[1] \in D \\ 3 & \textit{caso contrário} \end{cases}$$

[2]Origens (respect. Destinos) de conexões são portas de saídas (respect. entradas) de nós ou de instâncias de CEs, ou portas de entrada (respect. saída) do CE associado.

$$tc(c)[2] = \begin{cases} 3 & \textit{se } tc(c)[1] = ce \\ 2 \textit{ ou } 3 & \textit{se } tc(c)[1] \in D \cup J \\ 2 & \textit{caso contrário} \end{cases}$$

tal que,

1. $\forall c \in C.\, comp_s[lista_s][pos_s][2] = comp_t[lista_t][pos_t][2]$,

   onde

   $comp_s = sc(c)[1]$,

   $lista_s = sc(c)[2]$,

   $pos_s = sc(c)[3]$,

   $comp_t = tc(c)[1]$,

   $lista_t = tc(c)[2]$,

   $pos_t = tc(c)[3]$.

   Isto é, os tipos das portas de origem e destino de uma conexão devem ser iguais.

2. $\forall comp \in Componente, \forall l \in LPI \cup \{oE\}, \forall j \in \{1, ..., |l|\}.\ l = comp[k] \rightarrow \exists\ c \in C.\ tc(c) = \langle comp, k, j\rangle$, *onde* $LPI = \{comp[2] \mid comp \in E \cup D \cup J \cup R \cup A\} \cup \{comp[3] \mid comp \in D \cup J\}$. *Todas as portas de entrada dos componentes do CD ou portas de saída do CE associado são destino de alguma conexão.*

3. $\forall comp \in Componente, \forall l \in LPO \cup \{iE\}, \forall j \in \{1, ..., |l|\}.\ l = comp[k] \rightarrow \exists\ c \in C.\ sc(c) = \langle comp, k, j\rangle$, *onde* $LPO = \{comp[3] \mid comp \in E \cup R \cup A\} \cup \{comp[4] \mid comp \in D \cup J\} \cup \{comp[5] \mid comp \in D\}$. *Todas as portas de saída dos componentes do CD ou portas de entrada do CE associado são origem de alguma conexão.*

4. $\forall c_1, c_2 \in C.\, tc(c_1) = tc(c_2) \rightarrow c_1 = c_2$. *Não há conexões diferentes com o mesmo destino.*

5. $\forall c_1, c_2 \in C.\, sc(c_1)[1] = sc(c_2)[1] \in A \wedge sc(c_1)[3] = sc(c_2)[3] \rightarrow c_1 = c_2$. *Se duas conexões tem mesma origem em uma porta de saída de um nó de abstração, então as conexões são as mesmas, isto é, não pode haver duas conexões saindo da mesma porta de saída de um nó de abstração.*

6. *Não existe um caminho*[3] *que comece em uma porta de saída de um componente e termine em uma porta de entrada do mesmo componente.*

[3]Considera-se caminho uma sequência de conexões $c_1, c_2, \ldots c_n$ tal que $tc(c_i)$ é uma porta de entrada de um componente e $sc(c_{i+1})$ é uma porta de saída do mesmo componente, onde $1 \leq i \leq n-1$.

7. $\forall cond \in Cond.WFC(cond)$.

   $WFC(\langle k, d, j\rangle)$
   if *((Para todo caminho começando em alguma porta de* $d[3]$ *(respect.* $d[4]$*), este caminho termina em alguma porta de* $j[1]$ *(respect.* $j[2]$*))* $\wedge$
   (Para todo caminho começando em $d[3]$ *(respect.* $d[4]$*) e terminando em* $j[1]$ *(respect.* $j[2]$*), se neste caminho existir um nó de decisão, então seu respectivo nó de junção também está neste caminho)* $\wedge$
   (Para todo caminho terminando em alguma porta de $j[1]$ *(respect.* $j[2]$*), este caminho começa em alguma porta de* $d[3]$ *(respect.* $d[4]$*)))* **then**
   $true;$
   else
   $false;$
   end

8. $(tc(c)[1] \in R \rightarrow sc(c)[1] = ce) \wedge (sc(c)[1] \in A \rightarrow tc(c)[1] = ce)$. *Toda conexão que tem destino (respect. origem) numa porta de entrada (respect. saída) de um nó de refinamento (respect. abstração), tem origem (respect. destino) numa porta de entrada (respect. saída) do CE que está sendo decomposto.*

□

Cabe salientar que os elementos de E são instâncias de CE, isto é, e_i é a instância de um CE e, previamente definido. A restrição 6 evita ciclos nas sequências de conexões, impedindo que uma entrada de um componente dependa de uma de suas saídas. A restrição 7 garante que um condicional é bem formado. Um condicional é bem formado se: (i) cada caminho que começa em uma porta de saída de uma decisão (oT ou oF) termina em uma porta de entrada da junção correspondente (iT ou iF, respectivamente); (ii) todo caminho que começa numa porta de saída de uma decisão (d_1, como esquematizado na Figura 9) e termina em uma porta de entrada da junção correspondente (j_1), que contém um nó de decisão (d_2), deve também conter a junção correspondente a esse nó de decisão (j_2); (iii) todo caminho que termina em uma porta de entrada de uma junção (iT ou iF), começa em alguma porta da decisão correspondente (oT ou oF, respectivamente). Por sua vez, a restrição 8 restringe o uso dos nós de refinamento/abstração. Os dados descritos internamente num CD estão todos no mesmo nível de abstração, assim os componentes de refinamento/abstração de dados estará sempre ligado a portas de entrada/saída do componente elementar associado.

Exemplo 5. ***(Componente Decomposto (CD))****. O CD, ilustrado na Figura 8, é definido pela tupla*

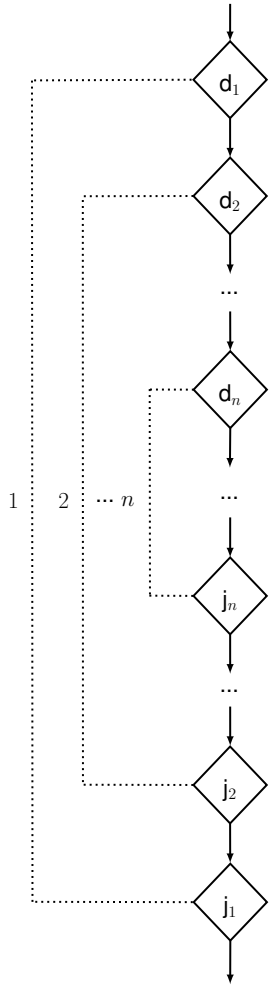


Figura 9 – Condicional Bem Formado.

$$BC = \langle E, Cond, R, A, C, sc, tc \rangle$$

onde

- $E = \{\langle Testa\ Massa\ Branca_1, iE_{T1}, oE_{T1}\rangle$,
  $\langle Massa\ Branca_1, iE_{MB1}, oE_{MB1}\rangle$,
  $\langle Massa\ Preta_1, iE_{MP1}, oE_{MP1}\rangle$,
  $\langle Fazer\ Cobertura_1, iE_{FC1}, oE_{FC1}\rangle$,
  $\langle Montar\ Bolo_1, iE_{MoB1}, oE_{MoB1}\rangle\}$ *contém cinco instâncias de CE conforme ilustrado na Figura 8*
- $Cond = \{\langle 1, d_1, j_1\rangle\}$ *(conforme def no Exemplo 2)*
- $R = \{\langle 1, i_1, oR_1\rangle, \langle 2, i_2, oR_2\rangle\}$ *(* $\langle 1, i_1, oR_1\rangle$ *está definido no Exemplo 3)*
- $A = \{\langle 1, iA, o\rangle\}$ *(conforme definição no Exemplo 3)*

- $C = \{C_1, ..., C_{24}\}$ *que define todas as conexões entre as portas do CD*
- sc *e* tc *definem a origem e destino de cada conexão. Por exemplo, se* $c1$ *é a conexão entre a porta de entrada* $\langle tm, string\rangle$ *do* ce *associado e a porta de entrada* $\langle tm, string\rangle$ *da instância do* **Testa Massa Branca**, *as funções* sc *e* tc *são definidas como segue:*

  $sc(c1) = \langle\langle Fazer\ Bolo\ com\ Cobertura, iE, oE\rangle, 2, 1\rangle$

  $tc(c1) = \langle\langle Testa\ Massa\ Branca_1, iE_{T1}, oE_{T1}\rangle, 2, 1\rangle$

  Considerando a conexão $c6$, *a conexão entre a segunda porta de saída do nó de refinamento dos ingredientes (Exemplo 3)* $\langle ico, ingCo\rangle$ *e a primeira porta de entrada da instância do* **Fazer Cobertura**, *as funções* sc *e* tc *são definidas por:*

  $sc(c6) = \langle\langle 1, i1, oR_1\rangle, 3, 2\rangle$

  $tc(c6) = \langle\langle Fazer\ Cobertura_1, iE_{FC1}, oE_{FC1}\rangle, 2, 1\rangle$

Uma **especificação** é definida por um conjunto de tipos, um conjunto de componentes elementares, um conjunto de componentes decompostos, uma função ed que associa componentes elementares a componentes decompostos e um componente elementar principal. Os tipos definem os valores que podem ser associados às portas de entrada e saída dos componentes de uma especificação. O conjunto de elementares define, de forma abstrata, todas as ações disponíveis na especificação. O conjunto de decompostos detalha algumas das ações da especificação. A função ed relaciona o CE (abstrato) ao seu detalhamento (dado pelo CD associado). O componente principal define a ação que descreve a solução como um todo.

Definição 7 (Especificação)**.** *Uma especificação é definida pela tupla* $\langle \mathcal{T}, \mathcal{E}, \mathcal{D}, ed, cp\rangle$, *onde:*

- $\mathcal{T}$ *é o conjunto de tipos da especificação. Cada* $T \in \mathcal{T}$ *é um conjunto que descreve os valores possíveis para o tipo* T.
- $\mathcal{E}$ *é um conjunto de componentes elementares* $e = \langle n, iE, oE\rangle$, *tal que:*
  - $n \in N$, *onde* N *é um conjunto de nomes e* $e_1, e_2 \in \mathcal{E} \wedge e_1[1] = e_2[1] \rightarrow e_1 = e_2$, *isto é, não existe componente elementar com o mesmo nome em uma especificação;*
  - $\forall e_1, e_2 \in \mathcal{E}.\ NP_1 \cap NP_2 = \emptyset$, *onde* $e_k = \langle n_k, iE_k, oE_k\rangle$, *com* $k \in \{1, 2\}$ *e* $NP_k = \{n \mid \langle n, T\rangle \in lSet(iE_K) \cup lSet(oE_k)\}$, *isto é, todas as portas de componentes elementares de uma especificação têm nomes diferentes.*

- $\mathcal{D}$ *é um conjunto de componentes decompostos* $cd = \langle E, Cond, R, A, C, sc, tc \rangle$*, tal que* $e_i \in E \rightarrow e \in \mathcal{E}$*.*
- ed*:* $\mathcal{E} \rightarrow \mathcal{D}$ *é uma função parcial e sobrejetora que associa um componente elementar a um decomposto.*
- $cp \in \mathcal{E}$*, onde* cp *é o componente elementar principal.*

□

De acordo com a definição, qualquer elementar cuja alguma instância apareça em um CD, deve estar presente no conjunto de componentes elementares da especificação. A função ed é parcial porque nem todos os componentes elementares precisam ser associados a um componente decomposto e é sobrejetora porque todos os decompostos devem estar associados a um elementar.

Exemplo 6. ***(Especificação)****. Pode-se definir uma especificação, utilizando os componentes definidos anteriormente, conforme detalhado a seguir:*

$Spec = \langle \mathcal{T}, \mathcal{E}, \mathcal{D}, ed, cp \rangle$

onde
$\mathcal{T} = \{string, ingMA, ingCo, ingMa \times ingCo, utMa, utMo, utCo,$
$utMa \times utMo \times utCo, bool, massaBolo, cobertura, boloCoberto\}$

$\mathcal{E} = \{\langle Fazer\ Bolo\ com\ Cobertura, iE, oE \rangle,$
$\langle Testa\ Massa\ Branca, iE_T, oE_T \rangle,$
$\langle Massa\ Branca, iE_{MB}, oE_{MB} \rangle,$
$\langle Massa\ Preta, iE_{MP}, oE_{MP} \rangle,$
$\langle Fazer\ Cobertura, iE_{FC}, oE_{FC} \rangle,$
$\langle Montar\ Bolo, iE_{MoB}, oE_{MoB} \rangle\}$

$\mathcal{D} = \{BC\}$ *(conforme Exemplo 5)*

$ed(\langle Fazer\ Bolo\ com\ Cobertura, iE, oE \rangle) = BC$

$cp = \langle Fazer\ Bolo\ com\ Cobertura, iE, oE \rangle$

Quando um componente elementar não possui um decomposto associado, ele descreve qualquer relação possível entre valores de entrada e valores de saída, respeitando os tipos das respectivas portas. Caso se deseje restringir essas relações,

deve-se definir as possibilidades por meio da função $valE$. Essa função associa elementares a possíveis relações entre seus valores de entrada e saída.

Definição 8 (Restrição de valores de CE)**.** *Dada uma especificação $\langle \mathcal{T}, \mathcal{E}, \mathcal{D}, ed, cp \rangle$, é possível restringir os valores dos componentes elementares $e \in \mathcal{E} - dom(ed)$ definindo-se a função parcial $valE : \mathcal{E} - dom(ed) \rightarrow 2^{RelIO}$ que associa um elementar e, o qual não está associado a um decomposto, a um conjunto V de pares, onde cada par define uma possibilidade de relação de valores de entrada com valores de saída de um elementar, tal que:* $\forall e \in dom(valE).\ x \in valE(e) \rightarrow x[1] = e$.

□

A restrição de $valE$ garante que os valores associados a um CE devem ser compatíveis com os tipos de portas de suas entrada e saída.

Exemplo 7. ***(Restrição de valores de CE)****. Caso se queira restringir os valores válidos para as entradas e saídas do CE define-se a função:*

$$\begin{aligned} valE(MB) = \{&\langle MB, \langle IngM2, UtM1\rangle, \langle massa\_branca1, UtM1\rangle\rangle, \\ &\langle MB, \langle IngM3, UtM1\rangle, \langle massa\_branca2, UtM1\rangle\rangle, \\ &\langle MB, \langle IngM2, UtM2\rangle, \langle massa\_branca3, UtM2\rangle\rangle\} \end{aligned}$$

onde $MB = \langle Massa\ Branca, iE_{MB}, oE_{MB}\rangle$.

Neste caso, pode-se observar que nem todas as combinações possíveis de entrada e saída são permitidas. Por exemplo, não é permitida qualquer entrada com os ingredientes $IngM1$ ou as combinações de $IngM3$ com $UtM2$.

A simulação de uma especificação ocorre a partir da instanciação de valores para as portas de entrada do seu componente principal. Para tanto, os valores dados devem respeitar os tipos das portas de entrada do componente principal.

Definição 9 (Instanciação de uma Especificação)**.** *Dada uma especificação $Spec = \langle \mathcal{T}, \mathcal{E}, \mathcal{D}, ed, cp \rangle$, uma instanciação de $Spec$ é definida por uma tupla $\langle Spec, lv \rangle$, onde $lv = \langle v_1, \ldots, v_k \rangle$ é uma lista de valores tal que:*

- $|iE| = k$ *e*
- $lv[i] \in iE[i][2]$, *com* $1 \leq i \leq k$,

onde $cp = \langle n, iE, oE \rangle$.

□

Exemplo 8. ***(Instanciação de uma Especificação)****. Considerando a especificação* $Spec$*, definida no Exemplo 6, pode-se definir a seguinte instanciação:*

$$\langle Spec, lv \rangle$$

onde
$lv = \langle \text{“}preta\text{”}, \langle IngM1, IngC2 \rangle, \langle UtM1, \{espátula, travessa\}, \{colher, panela, medidor\} \rangle \rangle$*, tal que* $IngM1$ *e* $IngC2$ *estão definidos no Exemplo 1 e* $UtM1$ *está definido no Exemplo 2.*

4.2 Semântica

Nesta seção, utiliza-se a semântica denotacional (SLONNEGER; KURTZ, 1995) para dar significado às especificações descritas na linguagem LiVE. A descrição semântica, além de permitir estudos futuros sobre análise de programas, deve também auxiliar na futura implementação de um interpretador/compilador para a linguagem. Nesta primeira versão, escolheu-se partir de um núcleo mais simples da linguagem e, por isso, a semântica aqui apresentada não considera estruturas recursivas. A definição da semântica para a linguagem completa foi deixada como um trabalho futuro. Como normalmente é feito na abordagem denotacional, o significado a uma especificação é dado em termos de objetos matemáticos (como valores verdades, tuplas de valores, funções, entre outros).

Domínio Sintático

A seguir, definem-se objetos sintáticos que são utilizados nas funções semânticas. $Identifier$ é um conjunto de pares que identificam todos os componentes de uma especificação, associando-os ao seu respectivo tipo. $Expression$ e $Command$ são conjuntos de componentes de uma especificação. Em $Command$ estão todos os componentes de uma especificação e em $Expression$ não estão incluídos os componentes decompostos e os elementares que estão associados a decompostos.

$Identifier = NCE \cup NDec \cup NJun \cup NRef \cup NAbs$, onde
$NCE = \{\langle n, \text{“}ce\text{”} \rangle \mid \langle n, iE, oE \rangle$ é um CE$\}$
$NDec = \{\langle k, \text{“}dec\text{”} \rangle \mid \langle k, d, j \rangle$ é um nó condicional$\}$
$NJun = \{\langle k, \text{“}jun\text{”} \rangle \mid \langle k, d, j \rangle$ é um nó condicional$\}$
$NRef = \{\langle n, \text{“}ref\text{”} \rangle \mid \langle n, i, oR \rangle$ é um nó de refinamento$\}$
$NAbs = \{\langle n, \text{“}abs\text{”} \rangle \mid \langle n, iA, o \rangle$ é um nó de abstração$\}$

$Command = CE \cup CD \cup Dec \cup Jun \cup Ref \cup Abs$
$Expression = CA \cup Dec \cup Jun \cup Ref \cup Abs$, onde
CE é um conjunto de componentes elementares.
CD é um conjunto de componentes decompostos.
CA é um conjunto de componentes elementares que não possuem CD associado.
$Dec = \{\langle k, d\rangle \mid \langle k, d, j\rangle$ é um nó condicional$\}$, é um conjunto de nós de decisão.
$Jun = \{\langle k, j\rangle \mid \langle k, d, j\rangle$ é um nó condicional$\}$, é um conjunto de nós de junção.
Ref é um conjunto de nós de refinamento.
Abs é um conjunto de nós de abstração.

$Type$ é um conjunto de tipos.

$\mathcal{V}al\mathcal{E}$ é um conjunto de funções de restrição de valores de CE.

$\mathcal{C}ond$ é um conjunto de nós condicionais.

$Specification$ é um conjunto de especificações.

Domínio Semântico e Funções Semânticas

A memória é modelada como uma função de identificadores para pares de listas de valores (ou indefinido):

$$Store = Identifier \rightarrow SV \cup \{undef\}$$
$$SV = Type^* \times Type^*$$

onde SV representa os valores que podem ser colocados na memória. Cada componente é associado a lista de valores de entrada e lista de valores de saída, respectivamente. Indefinido ($undef$) é um valor especial que indica que um identificador ainda não tem um valor associado. Quando o componente possui mais de uma lista de entrada ou saída, elas são concatenadas em SV.

A avaliação de componentes em $Expression$ produz listas de valores de saída. Os valores de um componente irão depender das listas de valores de entrada associadas com seus identificadores na memória. No caso de elementares que não estão associados a decompostos, os valores de saída podem ser restritos por funções em $\mathcal{V}al\mathcal{E}$. Portanto, a função semântica $evaluate$ para expressões possui a seguinte assinatura:

$$evaluate : Expression \times \mathcal{V}al\mathcal{E} \times Store \rightarrow Type^*.$$

A execução de componentes em $Command$ podem modificar a memória, então define-se o significado de um componente como uma função da memória corrente para uma nova memória. A memória é global e apenas uma memória existe. As referências à memória corrente e à nova memória dizem respeito a diferentes cenários de uma mesma memória. A assinatura da função semântica $execute$ para comandos é dada por:

$$execute : Command \times \mathcal{V}al\mathcal{E} \times Store \to Store.$$

Por fim, a partir de uma instanciação de uma especificação, as listas de valores finais dos seus componentes na memória definem a semântica da especificação. Então a assinatura de $meaning$ é dada por:

$$meaning : Specification \times \mathcal{V}al\mathcal{E} \times Type^* \to Store.$$

Funções Auxiliares

Para completar a definição denotacional de LiVE, são necessárias funções auxiliares para manipular a memória. Utiliza-se "sto" para elementos da memória e "val" para valores em SV.

$emptySto : Store$
$\quad emptySto\ I = undef$

$updateSto : Store \times Identifier \times SV \to Store$
$\quad updateSto(sto, I, val)I_1 = (\ \textbf{if}\ I = I_1\ \textbf{then}\ val\ \textbf{else}\ sto(I_1))$

$applySto : Store \times Identifier \to SV \cup \{undef\}$
$\quad applySto(sto, I) = sto(I)$

$emptySto$ é uma função constante que retorna uma memória vazia, isto é, uma memória onde todos os identificadores estão indefinidos. A função $updateSto$ recebe uma memória (sto), um identificador (I) e um valor (val – par contento uma lista de valores de entrada e um lista de valores de saída) e retorna uma memória onde o identificador I está associado ao valor val e os demais estão associados aos valores definidos na memória inicial sto. E a função $applySto$ recebe uma memória sto e um identificador I e retorna o valor associado a I na memória sto.

Equações Semânticas

As equações semânticas para a semântica denotacional de LiVE são descritas a seguir. A semântica de uma especificação $Spec$, a partir de uma instanciação $\langle Spec, lv \rangle$ e restrição de valores de componentes elementares $valE$, é dada por uma memória (a qual associa cada componente em $Spec$ a listas de valores de entrada e saída). O significado de $Spec$ é dado a partir da execução do seu componente elementar principal, a partir de uma memória sto contendo apenas o CE principal cp associado aos valores de lv como entrada e uma lista vazia de valores como saída. A restrição de valores $valE$ é dada como argumento pois é necessária para a avaliação dos componentes elementares.

$meaning[\![Spec, valE, lv]\!] = execute[\![cp, valE, sto]\!]$, onde
$sto = updateSto(emptySto, \langle n, \text{“}ce\text{”} \rangle, \langle lv, \varepsilon \rangle) \wedge cp = \langle n, iE, oE \rangle \wedge$
$\wedge\ Spec = \langle \mathcal{T}, \mathcal{E}, \mathcal{D}, ed, cp \rangle$

A execução de um componente toma uma memória sto com os valores de entrada associados a ele e a atualiza com os valores de saída, mantendo os mesmo valores de entrada (definindo sto').

Se um CE possui um CD associado, a sua execução é dada pela execução deste CD. Caso contrário, os valores de saída são obtidos a partir da avaliação do CE, considerando os seus valores de entrada armazenados na memória sto.

$execute[\![ce, valE, sto]\!] = sto'$, onde
if $ed(ce) = cd$
then $sto' = execute[\![cd, valE, sto]\!]$
else $sto' = updateSto(sto, \langle n, \text{“}ce\text{”} \rangle, \langle lv, evaluate[\![ce, valE, sto]\!] \rangle) \wedge$
$\wedge\ ce = \langle n, iE, oE \rangle \wedge \langle lv, \_ \rangle = applySto(sto, \langle n, \text{“}ce\text{”} \rangle)$

Na execução dos nós de decisão, junção, refinamento e abstração, os valores de saída também são obtidos a partir da avaliação do componente, considerando os seus valores de entrada armazenados na memória sto.

$execute[\![dec, valE, sto]\!] = sto'$, onde
$sto' = updateSto(sto, \langle dec[1], \text{“}dec\text{”} \rangle, \langle lv, evaluate[\![dec, valE, sto]\!] \rangle) \wedge$
$\wedge\ \langle lv, \_ \rangle = applySto(sto, \langle dec[1], \text{“}dec\text{”} \rangle)$

$execute[\![jun, valE, sto]\!] = sto'$, onde
$sto' = updateSto(sto, \langle jun[1], \text{“}jun\text{”} \rangle, \langle lv, evaluate[\![jun, valE, sto]\!] \rangle) \wedge$

$\wedge \langle lv, \_ \rangle = applySto(sto, \langle jun[1], \text{“}jun\text{”} \rangle)$

$execute[\![ref, valE, sto]\!] = sto'$, onde
$sto' = updateSto(sto, \langle ref[1], \text{“}ref\text{”} \rangle, \langle lv, evaluate[\![ref, valE, sto]\!] \rangle) \wedge$
$\wedge \langle lv, \_ \rangle = applySto(sto, \langle ref[1], \text{“}ref\text{”} \rangle)$

$execute[\![abs, valE, sto]\!] = sto'$, onde
$sto' = updateSto(sto, \langle abs[1], \text{“}abs\text{”} \rangle, \langle lv, evaluate[\![abs, valE, sto]\!] \rangle) \wedge$
$\wedge \langle lv, \_ \rangle = applySto(sto, \langle abs[1], \text{“}abs\text{”} \rangle)$

A execução de um CD é dada a partir da execução dos seus componentes internos, tomando e atualizando os valores de entrada (lvi) e saída (lvo), respectivamente, do CE associado a ele.

Para se obter os valores de saída lvo, o valor ($lvo[i]$) de cada porta de saída i é obtido individualmente, calculando o valor ($lvo'[k]$) da porta k (origem da conexão que chega em i). O valor da porta k é obtido executando-se o componente $comp$ a qual essa porta pertence. Essa execução é realizada sobre a memória sto'' na qual os valores de entrada do componente $comp$ já foram atualizados pela função $getI$.

$execute[\![cd, valE, sto]\!] = sto'$, onde
$sto' = updateSto(sto, \langle n, \text{“}ce\text{”} \rangle, \langle lvi, lvo \rangle)$, tal que
$ed(ce) = cd \wedge ce = \langle n, iE, oE \rangle \wedge cd = \langle E, Cond, R, A, C, sc, tc \rangle \wedge$
$\langle lvi, \_ \rangle = applySto(sto, \langle n, \text{“}ce\text{”} \rangle) \wedge lvo[i] = lvo'[k], 1 \leq i \leq |oE| \wedge$
$\langle \_, lvo' \rangle = applySto(sto'', \langle comp[1], s \rangle) \wedge$
$\wedge sto'' = execute[\![comp, valE, getI(cd, comp, valE, sto)]\!] \wedge$
$\wedge \exists c.(tc(c) = \langle ce, 3, i \rangle \wedge sc(c) = \langle comp, \_, k \rangle)$, com

$$s = \begin{cases} \text{“}ce\text{”}, \text{se } comp \in E \\ \text{“}ref\text{”}, \text{se } comp \in R \\ \text{“}abs\text{”}, \text{se } comp \in A \\ \text{“}dec\text{”}, \text{se } comp \in D \\ \text{“}jun\text{”}, \text{se } comp \in J \end{cases}$$

$D = \{\langle k, i_b, i, oT, oF \rangle \mid \langle k, d, j \rangle \in Cond \wedge d = \langle i_b, i, oT, oF \rangle\}$
$J = \{\langle k, iT, iF, o \rangle \mid \langle k, d, j \rangle \in Cond \wedge j = \langle iT, iF, o \rangle\}$

$getI$ é uma função recursiva que obtém os valores de entrada de um componente de um CD executando os componentes conectados a essas entradas. Assim, dados um CD, um componente $comp$ deste CD, uma restrição de valores de CE $valE$ e uma memória sto, essa função define uma memória sto' atualizando sto com os valores de entrada para $comp$.

Se o componente cujas entradas deseja-se obter é o próprio CD, a recursão termina e a memória sto' é exatamente a mesma memória sto, caso contrário pode-se ter ainda três diferentes situações:

(i) o componente é uma instância de CE, um nó de refinamento ou um nó de abstração. Neste caso são obtidos os valores de todas as portas de entrada do componente. (ii) o componente é um nó de junção e os valores de entrada devem ser obtidos através das portas $True$ ou $False$, dependendo do valor associado à porta booleana do correspondente nó de decisão. (iii) o componente é um nó decisão e, neste caso, deve-se obter os valores da porta booleana e das demais portas de entrada.

$getI : CD \times Component \times \mathcal{V}al\mathcal{E} \times Store \to Store$
$getI(cd, comp, valE, sto) = sto'$, onde
$cd = \langle E, Cond, R, A, C, sc, tc\rangle \wedge comp \in E \cup A \cup R \cup D \cup J \wedge$
$D = \{\langle k, i_b, i, oT, oF\rangle \mid \langle k, d, j\rangle \in Cond \wedge d = \langle i_b, i, oT, oF\rangle\} \wedge$
$J = \{\langle k, iT, iF, o\rangle \mid \langle k, d, j\rangle \in Cond \wedge j = \langle iT, iF, o\rangle\}$, tal que
if $ed(comp) = cd$
then $sto' = sto$
else
if $comp \in E \cup R \cup A$
then
$sto' = updateSto(sto, \langle comp[1], s\rangle, \langle lvi, \varepsilon\rangle)$
$lvi[i] = lvo[k], 1 \leq i \leq |comp[2]|$
$\langle \_, lvo\rangle = applySto(sto'', \langle comp'[1], s\rangle)$
$sto'' = execute[\![comp', valE, getI(cd, comp', valE, sto)]\!]$
$sc(c) = \langle comp', \_, k\rangle$
$tc(c) = \langle comp, 2, i\rangle$
else
if $comp \in J$
then
$sto' = updateSto(sto, \langle comp[1], s\rangle, \langle lviT \bullet lviF, \varepsilon\rangle)$
$dec = \langle comp[1], ib, i, oT, oF\rangle$
$sto'' = getI(cd, dec, valE, sto)$
$\langle ldec, \_\rangle = applySto(sto'', \langle comp[1], \text{“}dec\text{”}\rangle)$

if $ldec[1] = true$

then $lviF = \varepsilon$

$lviT[i] = lvo[k], 1 \leq i \leq |comp[2]|$

$\langle \_, lvo \rangle = applySto(sto''', \langle comp'[1], s \rangle)$

$sto''' = execute[\![comp', valE, getI(cd, comp', valE, sto)]\!]$

$sc(c) = \langle comp', \_, k \rangle$

$tc(c) = \langle comp, 2, i \rangle$

else $lviT = \varepsilon$

$lviF[i] = lvo[k], 1 \leq i \leq |comp[3]|$

$\langle \_, lvo \rangle = applySto(sto''', \langle comp'[1], s \rangle)$

$sto''' = execute[\![comp', valE, getI(cd, comp', valE, sto)]\!]$

$sc(c) = \langle comp', \_, k \rangle$

$tc(c) = \langle comp, 3, i \rangle$

else

if $comp \in D$

then

$sto' = updateSto(sto, \langle comp[1], s \rangle, \langle lvib \bullet lvi, \varepsilon \rangle)$

$lvib[1] = lvob[k]$

$\langle \_, lvob \rangle = applySto(sto'', \langle comp'[1], s \rangle)$

$sto'' = execute[\![comp', valE, getI(cd, comp', valE, sto)]\!]$

$sc(c) = \langle comp', \_, k \rangle$

$tc(c) = \langle comp, 2, 1 \rangle$

$lvi[i] = lvo[k'], 1 \leq i \leq |comp[3]|$

$\langle \_, lvo \rangle = applySto(sto''', \langle comp''[1], s \rangle)$

$sto''' = execute[\![comp'', valE, getI(cd, comp'', valE, sto)]\!]$

$sc(c) = \langle comp'', \_, k' \rangle$

$tc(c) = \langle comp, 3, i \rangle$

com $s = \begin{cases} \text{“}ce\text{”}, \text{se } comp \in E \\ \text{“}ref\text{”}, \text{se } comp \in R \\ \text{“}abs\text{”}, \text{se } comp \in A \\ \text{“}dec\text{”}, \text{se } comp \in D \\ \text{“}jun\text{”}, \text{se } comp \in J \end{cases}$

A avaliação de um componente toma uma memória com os valores de entrada de tal componente e uma restrição de valores de CE e calcula os valores de saída de acordo com o tipo de componente que está sendo avaliado.

Quando o componente avaliado é um CE, deve-se considerar a função de restrição de valores de CE ($valE$). Se $valE$ está indefinida para o componente, os valores de saída são selecionados, de forma não determinística, dentre os valores que compõem o tipo de cada porta de saída. Por outro lado, se a função $valE$ está definida para o componente, os valores de saída são aqueles associados pela função às entradas na memória para o referido componente.

$evaluate[\![ce, valE, sto]\!] = lvo$, onde
$ce = \langle n, iE, oE \rangle$
if $valE(ce) = undef$
then $lvo[k] \in oE[k][2]$, $1 \leq k \leq |oE| \wedge |lvo| = |oE|$
else $\langle ce, lvi, lvo \rangle \in valE(ce) \wedge$
$\langle lvi, \_ \rangle = applySto(sto, \langle ce[1], \text{“}ce\text{”} \rangle)$

Na avaliação de um nó de decisão, os valores de saída são os mesmos das entradas. O que deve ser feito na avaliação deste tipo de componente é selecionar por quais portas os valores serão enviados. Nesta definição, a lista de valores de saída lvo contém os valores de todas as portas de saída, tanto os das portas $True$ quanto os das portas $False$. Assim, na primeira metade da lista lvo estão registrados os valores das portas $True$ e na segunda metade, estão os valores das portas $False$. Caso o valor associado à porta booleana seja $true$, os valores de entrada são associados às portas $True$ e as portas $False$ terão valores indefinidos. Caso contrário, as portas $True$ terão os valores indefinidos e as portas $False$ receberão os valores de entrada.

$evaluate[\![dec, valE, sto]\!] = lvo$, onde
$\langle lvib \bullet lvi, \_ \rangle = applySto(sto, \langle dec[1], \text{“}dec\text{”} \rangle)$
if $lvib = true$
then $lvo[k] = lvi[k]$
$lvo[j] = undef$
else $lvo[k] = undef$
$lvo[j] = lvi[j - |lvi|]$,
tal que $1 \leq k \leq |lvi|, |lvi| + 1 \leq j \leq 2|lvi|, |lvib| = 1, |lvo| = 2|lvi|$

Já a avaliação de um nó de junção é realizada de forma dual à do nó de decisão. O que deve ser feito é definir por quais portas de entrada os valores estão chegando

e, então, transferi-los para as portas de saída. Essa definição se dá com base no valor associado à porta booleana do nó de decisão correspondente.

$evaluate[\![jun, valE, sto]\!] = lvo$, onde
$\langle lviT \bullet lviF, \_\rangle = applySto(sto, \langle jun[1], \text{“}jun\text{”}\rangle)$
$\langle lvib \bullet lvi, \_\rangle = applySto(sto, \langle jun[1], \text{“}dec\text{”}\rangle)$
if $lvib = true$
then $lvo = lviT$
else $lvo = lviF$,
tal que $|lviT| = |lviF|, |lvib| = 1$

A avaliação de um nó de refinamento constrói a lista de valores de saída, obtendo cada valor na correspondente posição da tupla de valores da entrada.

E por fim, a avaliação de um nó de abstração, constrói a tupla de valores de saída, agrupando os valores da lista de valores de entrada.

$evaluate[\![ref, valE, sto]\!] = lvo$, onde
$lvo[k] = proj_k^{|lvi[1]|}(lvi)$, tal que $1 \leq k \leq |lvi[1]|, |lvo| = |lvi[1]| \wedge$
$\langle lvi, \_\rangle = applySto(sto, \langle ref[1], \text{“}ref\text{”}\rangle)$

$evaluate[\![abs, valE, sto]\!] = lvo$, onde
$proj_k^{|lvi|}(lvo) = lvi[k]$, tal que $1 \leq k \leq |lvi|, |lvo[1]| = |lvi| \wedge$
$\langle lvi, \_\rangle = applySto(sto, \langle abs[1], \text{“}abs\text{”}\rangle)$

5 METODOLOGIAS DE USO DA LINGUAGEM LiVE

Neste capítulo, propõe-se algumas metodologias para o uso da linguagem visual LiVE, de forma a sistematizar o uso das técnicas do PC na resolução de problemas. As técnicas/estratégias abordadas são: refinamento, decomposição, composição, abstração e generalização.

Estas metodologias propostas podem ser consideradas ativas, pois tem como características o aluno como um sujeito ativo no processo de aprendizagem, onde ele deve aprender fazendo. As metodologias guiam para uma solução de um problema, dentre várias possíveis. É o aluno quem deve escolher qual caminho seguir, sendo o protagonista na construção da solução (VALENTE; ALMEIDA; GERALDINI, 2017; MORAN, 2015).

Este capítulo foi dividido em 4 seções, conforme as habilidades citadas: metodologia de refinamento, metodologia da decomposição e composição, metodologia da abstração e metodologia da generalização. Cada seção traz a metodologia e um exemplo de aplicação da metodologia utilizando a linguagem visual LiVE.

5.1 Metodologia de Refinamento

Como foi mostrado na definição de abstração na seção 3.1 a solução de um problema pode ser apresentada em diferentes níveis. Na metodologia de refinamento, propõe-se a construção da solução partindo de um nível mais alto para o mais baixo.

O refinamento consiste em detalhar um problema até que se chegue ao nível de detalhamento da solução desejada. Esta metodologia se propõe a solucionar problemas usando a linguagem LiVE através do refinamento. Inicialmente, a solução é dada através de uma ação principal, identificando suas entradas e suas saídas. A partir disso, detalhamentos são feitos para a ação principal e eventualmente para ações que refinam a ação principal, até que o resultado, no nível de detalhamento desejado, seja alcançado.

Partindo de um problema, a descrição da solução deve seguir as seguintes etapas (sintetizadas na Figura 10):

R1 Especificar a solução do problema por um componente elementar, definindo o componente elementar principal. Nesta etapa são identificadas todas informações de entrada (mesmo em um nível mais alto de abstração) assim como os resultados a serem alcançados.

R2 Refinar o componente principal por meio de um componente decomposto. Nesta etapa são identificados os componentes elementares que detalham a ação do componente principal e suas conexões, definindo caso necessário nós condicionais. Além disso, definem-se os refinamentos e abstrações dos dados de entrada e saída do componente elementar associado.

R3 Refinar cada componente elementar identificado na etapa anterior. Esta etapa é executada sucessivas vezes até que o nível de detalhamento desejado seja alcançado (podendo não ser executada caso o nível de detalhamento seja alcançado na R2). Caso se identifique a falta/inconsistência de alguma informação em algum nível de refinamento, deve-se voltar na especificação mais abstrata para adicioná-la. Neste caso, todos os componentes de níveis anteriores devem ser revistos.

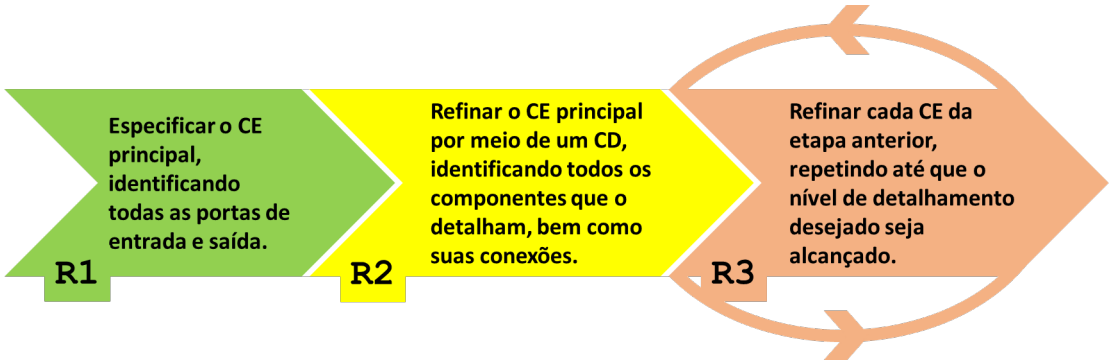


Figura 10 – Síntese da Metodologia de Refinamento.

Para exemplificar o uso da linguagem, especifica-se a solução do problema de confeccionar um bolo com cobertura. O exemplo aqui descrito consiste numa simplificação do apresentado no capítulo 4, descrevendo as etapas da preparação de um bolo com cobertura com uma única opção de massa.

R1 Nesta etapa, no nível mais alto de abstração, descreveu-se o componente elementar principal, conforme Figura 11, identificando a ação a ser executada (**Fazer Bolo com Cobertura**), as informações de entrada (ingredientes e utensílios) e os resultados/saídas (o bolo coberto e os utensílios).

R2 Nesta etapa, detalhou-se a ação **Fazer Bolo com Cobertura** em um conjunto de ações que levam ao resultado esperado. Neste nível identificou-se quais

Figura 11 – Etapa R1: CE **Fazer Bolo com Cobertura**.

ações menores compõem a ação principal. Em particular, a ação **Fazer Bolo com Cobertura** foi decomposta em 3 ações: **Fazer Massa**, **Fazer Cobertura** e **Montar Bolo** conforme ilustrado na Figura 12.

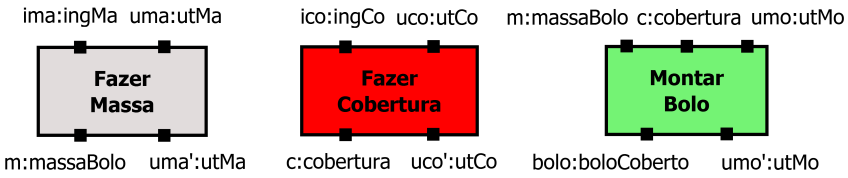


Figura 12 – Etapa R2: CEs **Fazer Bolo com Cobertura**.

As relações (dependências) entre elas estão especificadas pelas conexões entre os componentes elementares. Neste caso, a ação **Montar Bolo** depende dos resultados das ações **Fazer Massa** e **Fazer Cobertura**. Na descrição da solução estas dependências podem ser observadas pelas conexões representadas na Figura 13. Também é possível observar que as ações **Fazer Massa** e **Fazer Cobertura** são independentes entre si, já que as entradas de ambas não são resultados de outros componentes elementares. Além disso, as entradas do componente decomposto foram detalhadas. O refinamento da entrada ingredientes, foi detalhado diferenciando os ingredientes da massa (*ingMa*) dos ingredientes da cobertura (*ingCo*). Já o refinamento da entrada utensílios, foi detalhado diferenciando os utensílios usados para fazer a massa (*utMa*), daqueles para montar o bolo (*utMo*) e dos usados para fazer a cobertura (*utCo*). A saída utensílios também foi detalhada, onde o nó de abstração reuniu todos os tipos de utensílios utilizados para compor a saída do componente decomposto.

R3 O nível de detalhamento desejado foi alcançado, mas como algumas informações de entrada/saída foram detalhadas/abstraídas, retorna-se na especificação mais abstrata e atualizam-se os tipos de entradas e saídas (conforme Figura 14).

5.2 Metodologia da Decomposição e Composição

A **decomposição** refere-se à solução de um problema a partir da sua divisão em subproblemas mais simples, cujas soluções são combinadas para resolver o problema

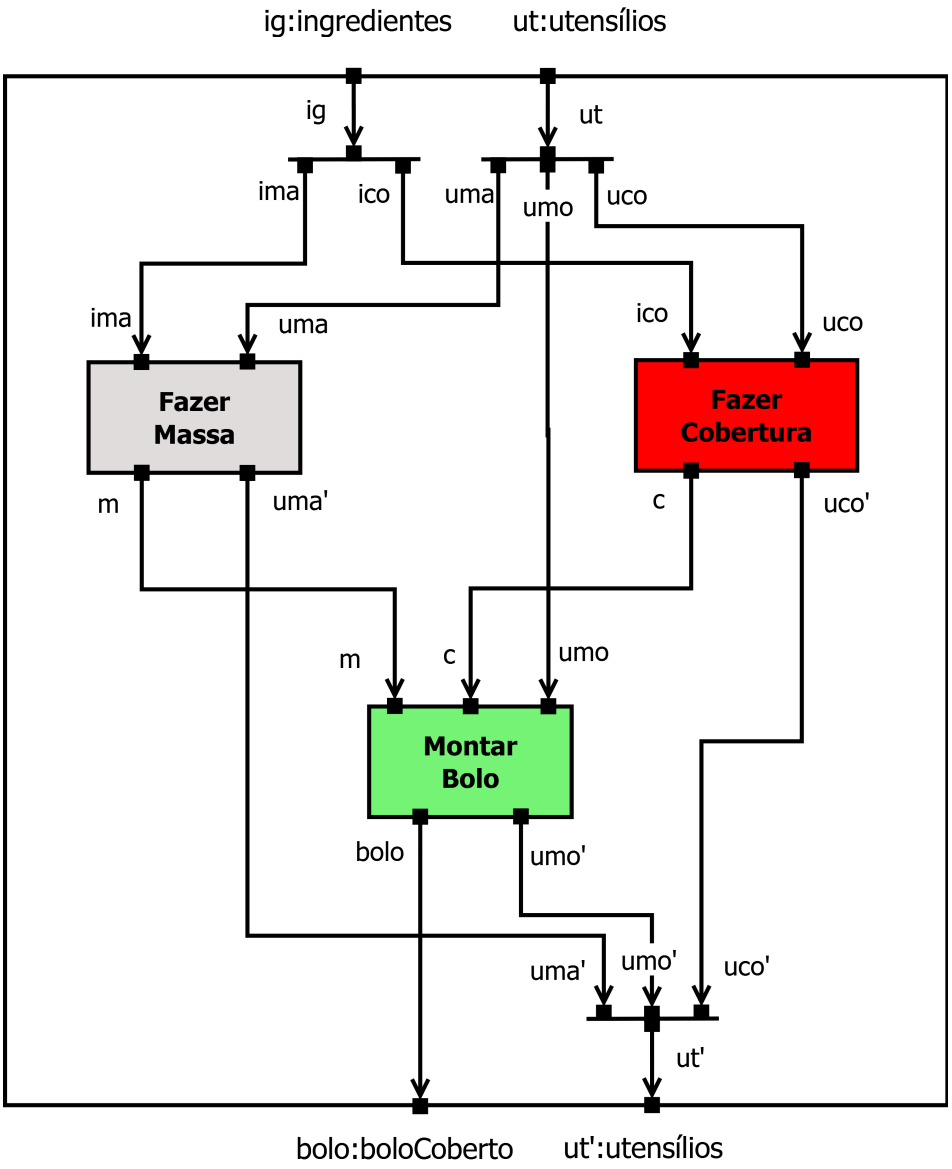


Figura 13 – Etapa R2: CD **Fazer Bolo com Cobertura**.

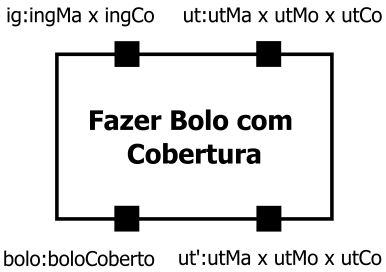


Figura 14 – Etapa R3: CE principal Fazer Bolo com Cobertura.

maior. Já a **composição** estabelece como as soluções dos subproblemas devem ser combinadas para resolver o problema original. A **Metodologia da Decomposição e Composição** sistematiza a solução de um problema pela combinação destas duas técnicas.

Nesta metodologia, quer-se descrever a solução de um problema a partir da divisão

do problema original em subproblemas, da definição de CEs que solucionam cada subproblema e da composição destes CEs (ações) que definem um CD, o qual resolve o problema original.

A solução do problema é dada por uma especificação, obtida por meio da construção de um componente decomposto a partir de instâncias de CE e conexões entre esses componentes. Dado um conjunto de CEs, o processo de solução deve seguir as seguintes etapas (sintetizado na Figura 15):

D1 Dividir o problema maior em subproblemas menores (mais simples) onde as saídas de cada subproblema dependem exclusivamente de suas entradas, cujas combinações das soluções permitam resolver o problema original. É desejável que para quaisquer dois subproblemas menores, nenhum seja subproblema do outro. Represente a divisão estabelecida como uma árvore: onde a raiz é o problema e os filhos são os subproblemas identificados.

D2 Dividir os subproblemas até que eles tenham soluções triviais. Para cada subproblema a ser dividido, selecione na árvore o nó que representa esse subproblema e adicione um filho para cada uma de suas divisões.

D3 Definir um CE para cada subproblema trivial (i.e., subproblemas relacionados às folhas da árvore) distinto identificado nas etapas anteriores (D1 ou D2), estabelecendo suas entradas e saídas.

C1 Adicionar uma instância de CE para resolver cada um dos subproblemas triviais e marcá-los como resolvidos.

C2 Identificar os menores subproblemas (i.e., os de nível mais baixo na árvore) que ainda não foram resolvidos. Selecionar um subproblema e identificar as ações necessárias para resolvê-lo combinando soluções anteriores. Definir um CE para cada ação de combinação (i.e., ação (de convergência) que liga outras ações para a resolução de um subproblema) distinta. Adicionar no CD em construção as instâncias necessárias das ações de combinação e estabelecer as conexões necessárias para resolver o subproblema selecionado. Cada entrada de uma ação de combinação deve ser conectada a uma saída de alguma ação das ações combinadas. Repetir a etapa C2 até que o problema principal seja resolvido.

C3 Definir o CE cuja decomposição foi estabelecida nas etapas anteriores. Criar para o CE: uma porta de entrada para cada entrada independente do CD (portas de entrada das instâncias de CE que compõem o CD e que não estão conectadas a outras portas), estabelecendo as conexões entre tais entradas; e, uma porta de saída para cada saída independente do CD (portas de saída das instâncias de CE que compõem o CD e que não estão conectadas a outras portas),

estabelecendo as conexões com tais saídas. Além disso, um nome deve ser escolhido para este CE que será o componente principal da especificação.

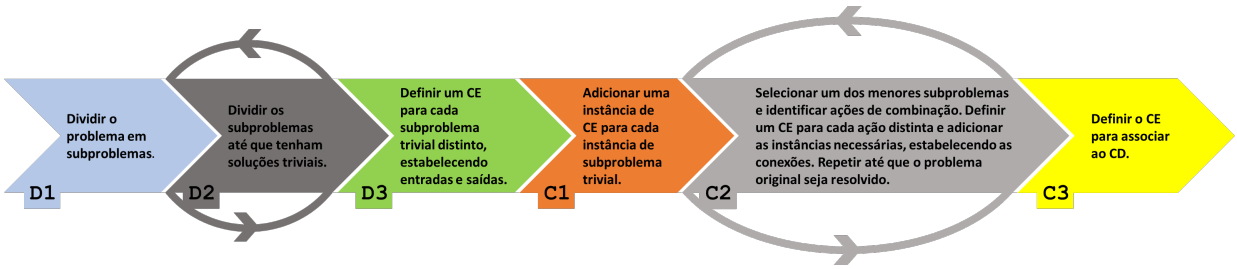


Figura 15 – Síntese da Metodologia da Decomposição e Composição.

Para exemplificar a aplicação da metodologia da Decomposição e Composição, descreve-se a solução do problema de desenhar a bandeira ilustrada na Figura 16.

A bandeira é obtida a partir de quatro retângulos de mesmo tamanho, sendo dois com uma estrela no centro. São consideradas as seguintes ações triviais que solucionam este problema: desenhar **Retângulo** de qualquer cor e qualquer tamanho; desenhar **Estrela** de qualquer cor e com qualquer tamanho de lado; posicionar uma figura **Em cima** de outra figura, alinhando a borda esquerda das duas figuras; posicionar uma figura **Ao lado** de outra figura, alinhando a borda superior das duas figuras; e posicionar uma figura **Sobre** outra figura e alinhar o centro de uma figura sobre o centro da outra.

Figura 16 – Bandeira Panamá.

Usando a metodologia da decomposição e composição, a construção da solução pode ser obtida por meio das etapas descritas a seguir. Cabe observar que existem diferentes maneiras de aplicar a metodologia para obter o mesmo resultado, dependendo da quantidade de CEs selecionados na primeira ocorrência de C1. Na descrição apresentada, todas as ações independentes foram selecionadas já na primeira ocorrência, tornando desnecessária a sua repetição.

D1 O problema de **Desenhar Bandeira** foi dividido em subproblemas menores conforme ilustrado na Figura 17.

D2 Nesta etapa, os subproblemas da etapa D1 foram divididos até que tenham soluções triviais, conforme árvore ilustrada na Figura 18 (as folhas da árvore – em azul – representam as ações triviais).

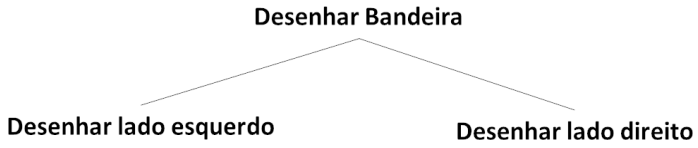


Figura 17 – Etapa D1: Árvore da Bandeira.

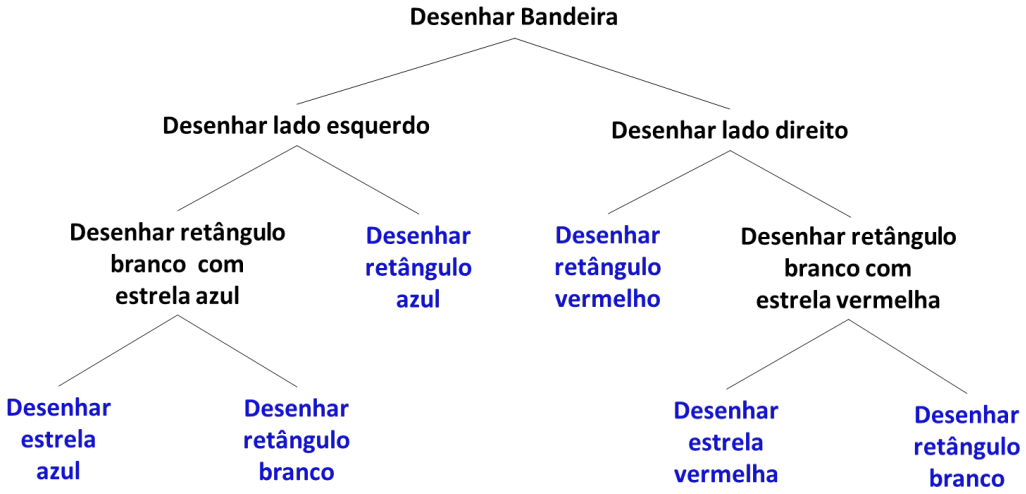


Figura 18 – Etapa D2: Árvore da Bandeira.

D3 Nesta etapa, foram definidos os CEs ilustrados na Figura 19. Note-se que alguns subproblemas compartilham soluções e que apenas um CE foi criado para cada subproblema trivial distinto. Para o CE **Retângulo** foram definidas 3 entradas: *larg* (largura) e a *alt* (altura), ambas do tipo *num* (numérico) e a *cor* do tipo *string*; e uma saída *ret* do tipo *imagem*. Para o CE **Estrela** foram definidas 2 entradas: *lado* do tipo *num* e *cor* do tipo *string*; e uma saída *est* do tipo *imagem*.

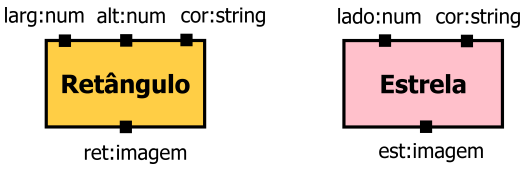


Figura 19 – Etapa D3: CEs para subproblemas triviais.

C1 Nesta etapa, adicionou-se uma instância de CE para cada instância de subproblema trivial definido na etapa D2. Foram adicionadas 4 instâncias de **Retângulo** e 2 de instâncias de **Estrela**, conforme ilustrado na Figura 20.

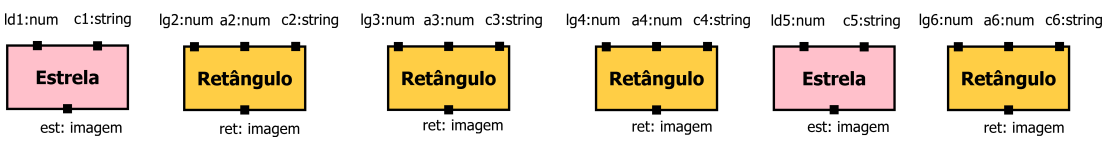


Figura 20 – Etapa C1: Instâncias de CEs para resolver os subproblemas do problema de desenhar a Bandeira.

C2 Esta etapa foi repetida 5 vezes.

Na primeira repetição foram identificados dois subproblemas menores (Figura 21): **Desenhar retângulo branco com estrela azul** e **Desenhar retângulo branco com estrela vermelha**, e selecionou-se o primeiro para começar a resolver.

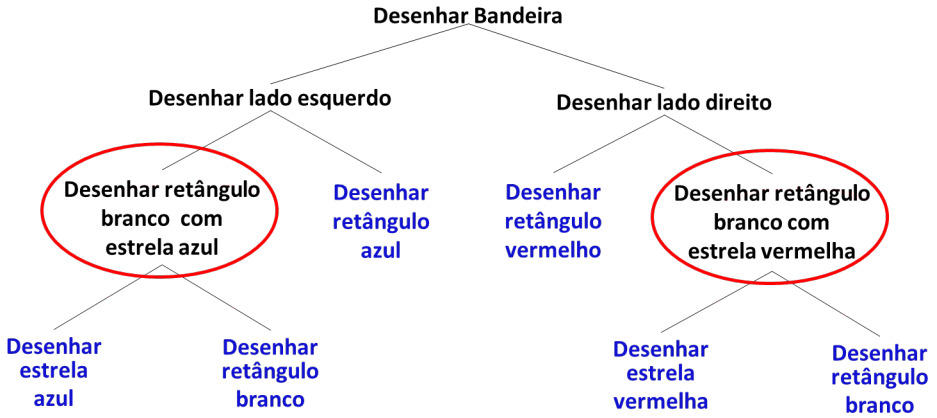


Figura 21 – Etapa C2: Primeira repetição, subproblemas menores.

Para combinar as soluções dos subproblemas de **Desenhar retângulo branco com estrela azul**, identificou-se a ação de combinação **Sobre** e definiu-se um CE para esta ação. Adicionou-se uma instância de **Sobre** estabelecendo as conexões, conforme Figura 22.

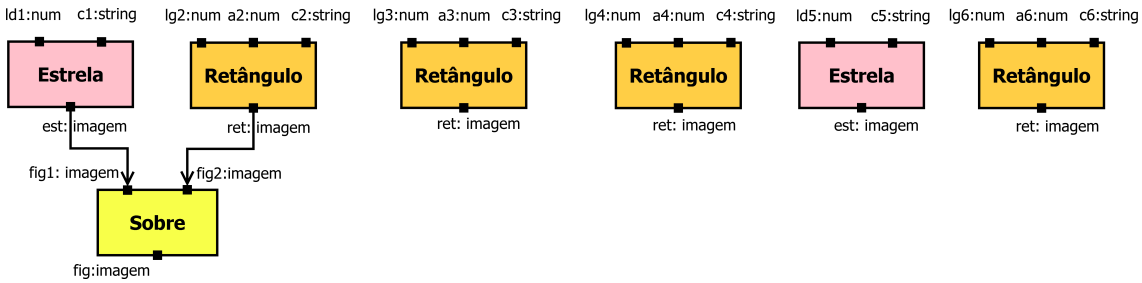


Figura 22 – Etapa C2: Primeira repetição.

Na segunda repetição selecionou-se o subproblema **Desenhar retângulo branco com estrela vermelha**. Identificou-se a ação de combinação **Sobre**, a mesma definida anteriormente, não precisando assim definir novamente um CE para esta ação. Adicionou-se outra instância de **Sobre** e estabeleceu-se as conexões, conforme Figura 23.

Na terceira repetição, foram identificados novamente dois subproblemas menores (Figura 24[1]): **Desenhar lado esquerdo** e **Desenhar lado direito**, selecionou-se o primeiro para continuar a resolver do problema.

[1]Os subproblemas já resolvidos estão descritos na cor cinza.

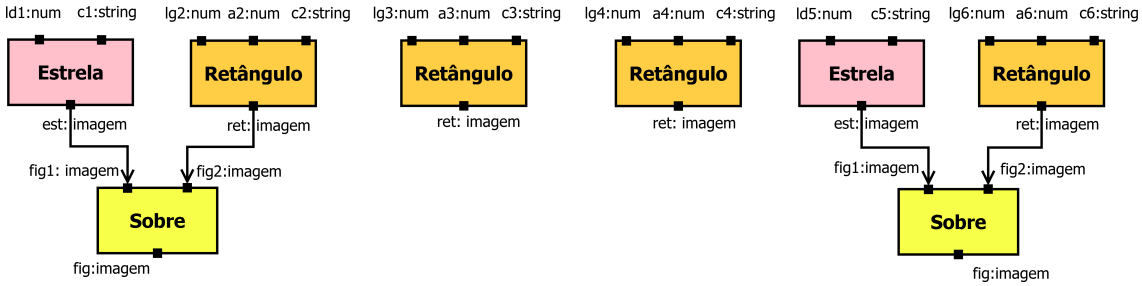


Figura 23 – Etapa C2: Segunda repetição.

Figura 24 – Etapa C2: Terceira repetição, subproblemas menores.

Identificou-se a ação de combinação **Em cima** e definiu-se um CE para esta ação. Adicionou-se uma instância de **Em cima** e estabeleceu-se as conexões, conforme Figura 25.

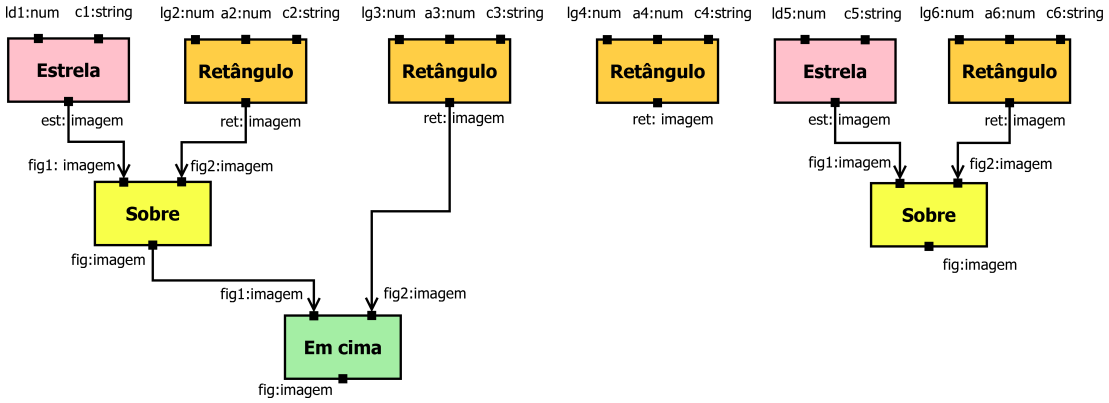


Figura 25 – Etapa C2: Terceira repetição.

Na quarta repetição selecionou-se o subproblema **Desenhar lado direito**. Identificou-se a ação de combinação **Em cima**, já definida, adicionou-se outra instância de **Em cima** e estabeleceu-se as conexões, conforme Figura 26.

Na quinta repetição, identificou-se um único subproblema menor (Figura 27) **Desenhar Bandeira** e selecionou-se este.

Identificou-se a ação de combinação **Ao lado** e definiu-se um CE para esta ação.

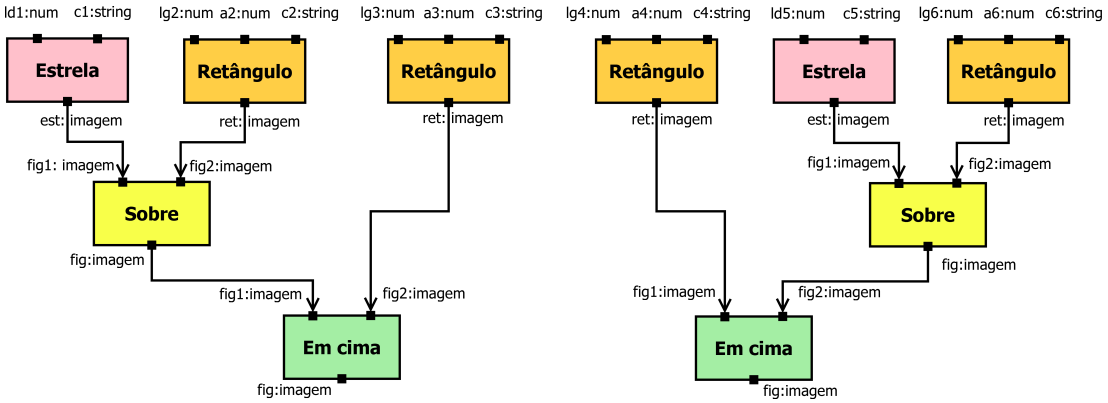


Figura 26 – Etapa C2: Quarta repetição.

Figura 27 – Etapa C2: Quinta repetição, subproblema menor.

Adicionou-se uma instância de **Ao lado** e estabeleceu-se as conexões, conforme Figura 28.

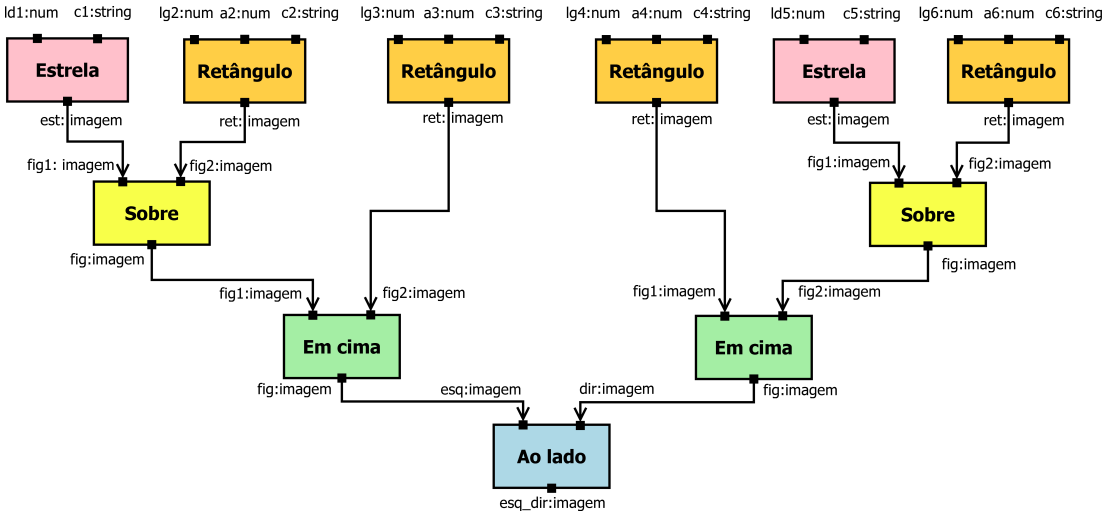


Figura 28 – Etapa C2: Componente decomposto para Desenhar Bandeira.

Após a quinta repetição, verificou-se que o problema havia sido resolvido, isto

é, o problema principal de Desenhar a Bandeira foi resolvido, o que resultou no componente decomposto ilustrado na Figura 28.

Nessas interações foram criados os CEs para as ações de combinação, conforme ilustrado na Figura 29.

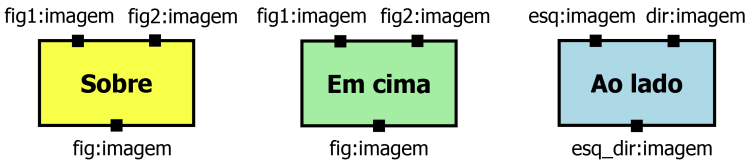


Figura 29 – Etapa C2: CEs para ações de combinação.

C3 Definiu-se o CE (Figura 30) para associar ao CD definido na etapa anterior, denominado **Desenhar Bandeira**, esquecendo toda a estrutura interna do CD. Esse CE possui uma porta de entrada/saída para cada entrada/saída independente do CD.

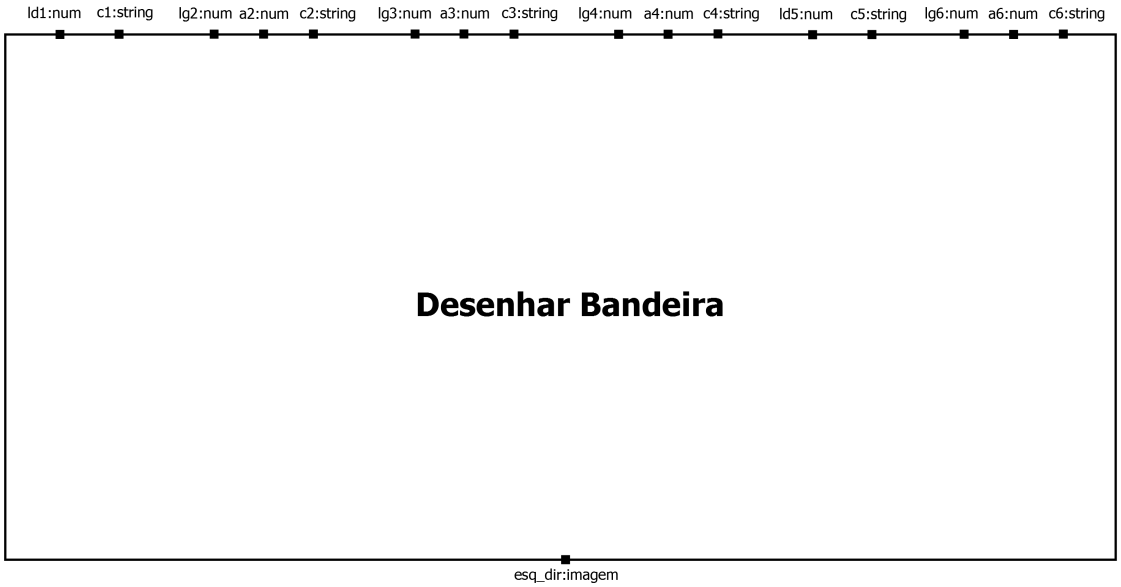


Figura 30 – Etapa C3: CE principal Desenhar Bandeira.

5.3 Metodologia da Abstração

Na **Metodologia da Abstração**, objetiva-se simplificar a solução de um problema a partir da identificação de estruturas (conjuntos de componentes) que podem ser omitidas. Para isso, definem-se componentes elementares que abstraem as estruturas selecionadas e substitui-se todas as ocorrências destas estruturas por suas respectivas abstrações. Dado um CD C de uma especificação S, o processo de abstração deve seguir as seguintes etapas (sintetizadas na Figura 31):

A1 Selecionar dois ou mais componentes (instancias de CE, nós condicionais, de refinamento ou de abstração) de C que serão abstraídos. Esta seleção deve respeitar as seguintes restrições: (i) Para cada nó condicional selecionado, todos os componentes que estão nos caminhos que partem do nó de decisão e chegam no nó de junção devem ser selecionados; (ii) Para quaisquer dois componentes selecionados, para os quais existe um caminho que os conecta, todos os componentes deste caminho, também devem ser selecionados; (iii) Pelo menos um componente elementar de C não deve fazer parte da seleção.

A2 Identificar as portas de entrada/saída externas (portas de entrada/saída que não estão conectadas com portas de componentes selecionados) ao conjunto de componentes selecionados em A1.

A3 Definir o componente elementar E que abstrai a estrutura selecionada, criando uma porta de entrada/saída para cada porta externa identificada em A2, respeitando os tipos.

A4 Adicionar E em S e uma instância de E em C. Altere o destino/origem de toda conexão ligada a uma porta de entrada/saída externa, redefinindo-o/a para a respectiva porta de entrada/saída de E.

A5 Definir um CD D, transfira os componentes abstraídos e suas respectivas conexões internas (não ligadas a portas externas) de C para D, conecte as portas de entrada/saída independentes de D com as respectivas portas de entrada/saída de E. Associando-o ao componente elementar E.

A6 Verificar na especificação se algum CD contém a mesma estrutura abstraída. Em caso afirmativo, substitua todas as estruturas pelo CE abstrato.

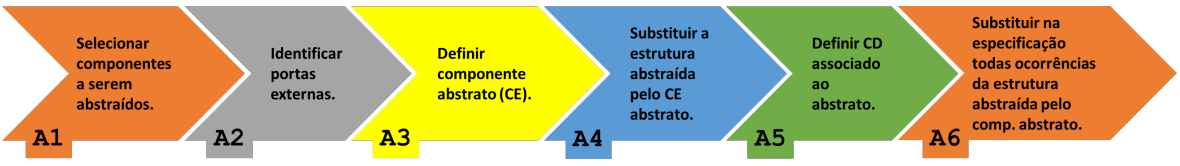


Figura 31 – Síntese da Metodologia da Abstração.

Para exemplificar a aplicação da metodologia da abstração, parte-se da especificação da solução do problema de construir a maquete ilustrada na Figura 32. A maquete é obtida a partir do posicionamento das seguintes construções em um tabuleiro base dado como entrada: duas casas, um edifício de 3 andares, uma igreja de duas torres, uma praça arborizada e uma praça com brinquedos. Os componentes elementares, ilustrados na Figura 33, descrevem as ações que foram utilizadas na especificação

da solução. Em particular, a ação **Construir Piso**, recebe duas entradas do tipo inteiro (int), indicando respectivamente largura e profundidade, e resulta numa saída do tipo piso. Já a ação **Construir Parede** recebe como entrada um piso e retorna uma estrutura.

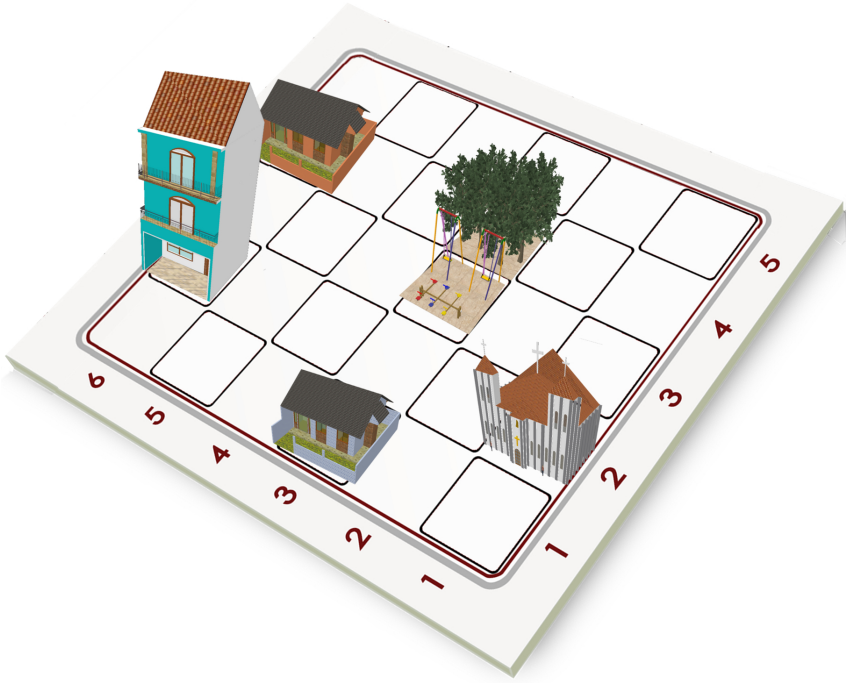


Figura 32 – Maquete.

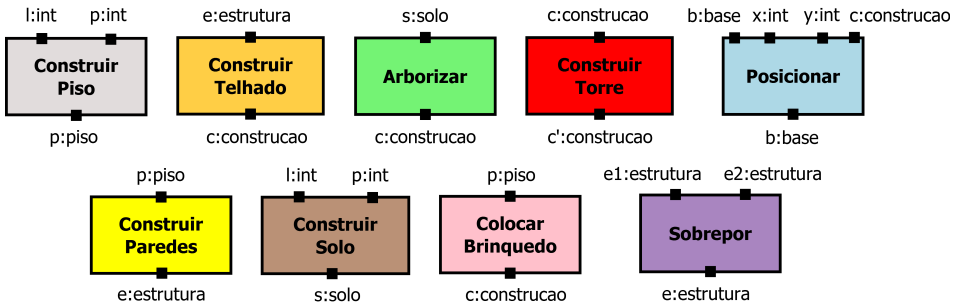


Figura 33 – CEs da maquete.

A Figura 34 apresenta o CD da especificação da solução do problema **Construir Maquete**. Nesta especificação, identificam-se conjuntos de componentes que podem ser abstraídos: **Construir Casa**, **Construir Edifício**, **Construir Igreja**, **Construir Praça com Brinquedo** e **Construir Praça Arborizada**. Aqui, como exemplo, aplicou-se a metodologia para abstrair a construção do edifício, a partir das seguintes etapas:

A1 Selecionou-se os componentes delimitados pelo retângulo pontilhado da Figura 34.

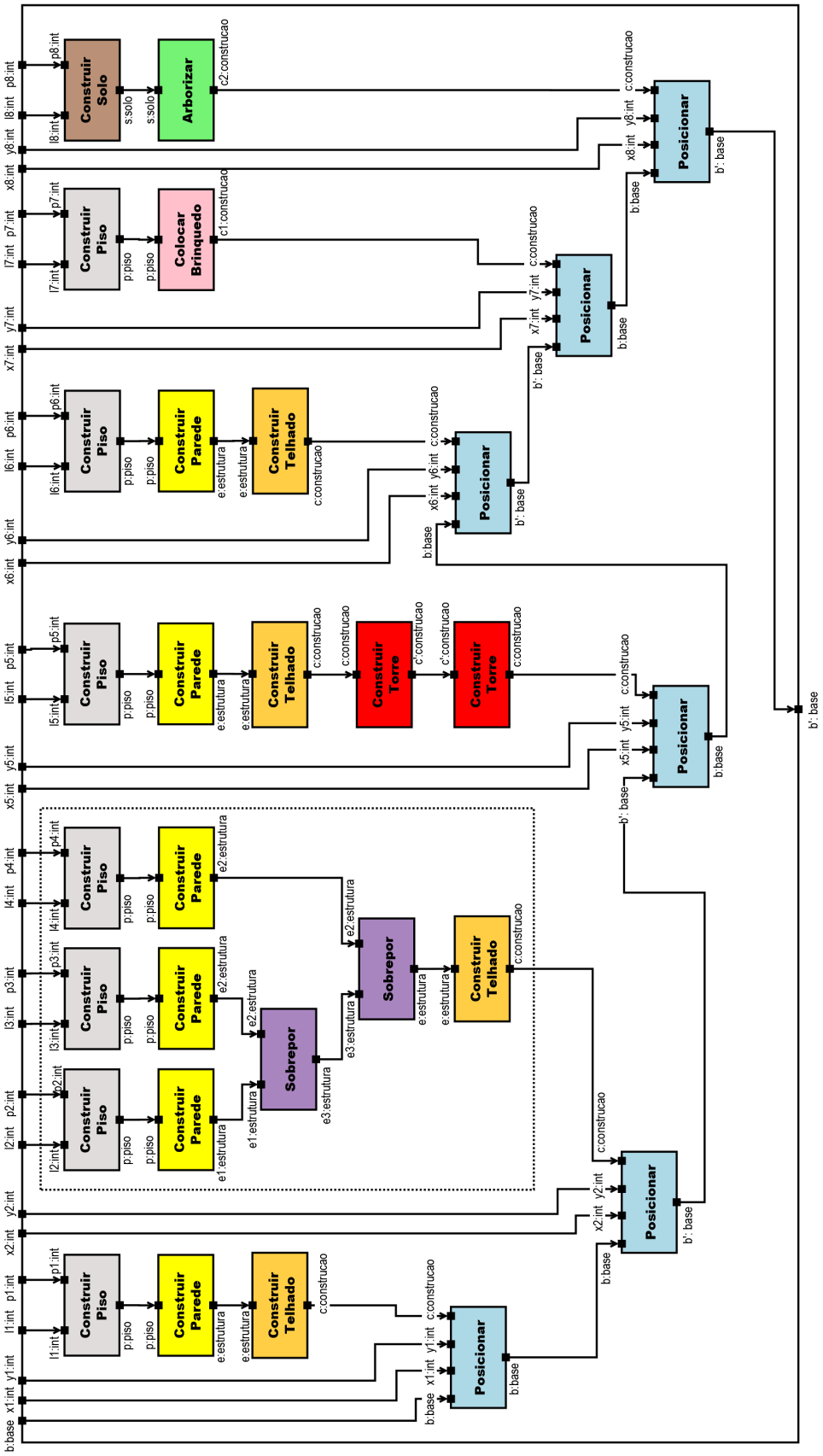


Figura 34 – Exemplo de CD para **Construir Maquete**.

A2 Nesta etapa foram identificadas as portas de entrada externas *l2*, *p2*, *l3*, *p3*, *l4* e *p4* e a porta de saída externa *c*.

A3 Criou-se o CE **Construir Edifício** com as portas identificadas na etapa A2, conforme ilustrado na Figura 35.

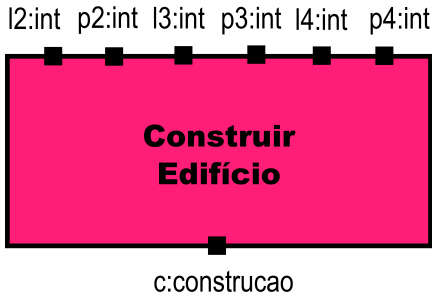


Figura 35 – Exemplo de CE Abstrato.

A4 Substituiu-se a região pontilhada na Figura 34 pelo CE obtido na etapa A3 (Figura 36).

A5 Definiu-se o CD ilustrado na Figura 37 que contém a estrutura abstraída.

A6 Não é aplicada, já que não tem outra ocorrência da construção de edifício.

Esta metodologia também poderia ser aplicada para abstrair as demais construções, resultando na especificação da Figura 38.

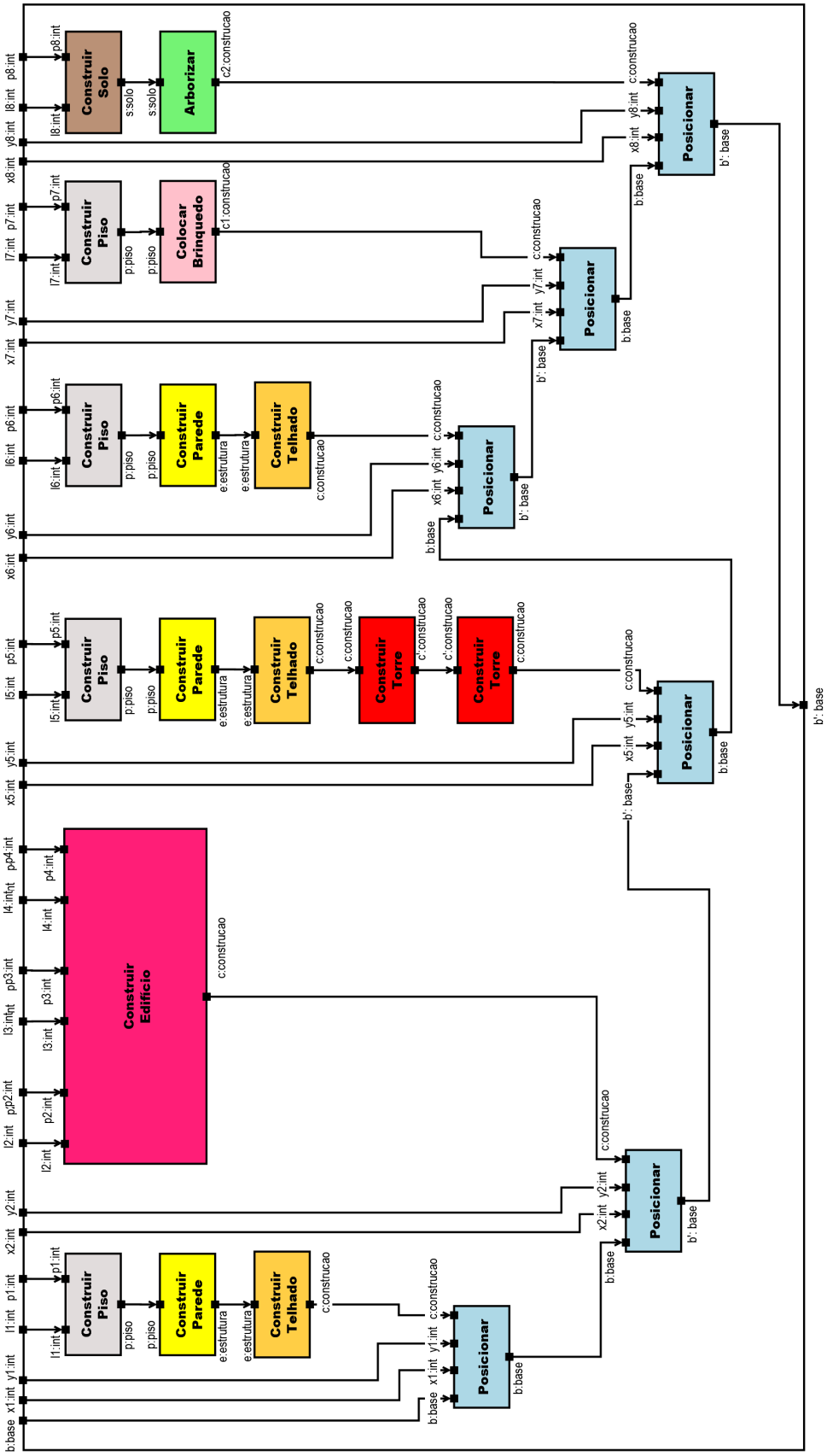


Figura 36 – CD **Construir Maquete** com o CE abstraído **Construir Edifício**.

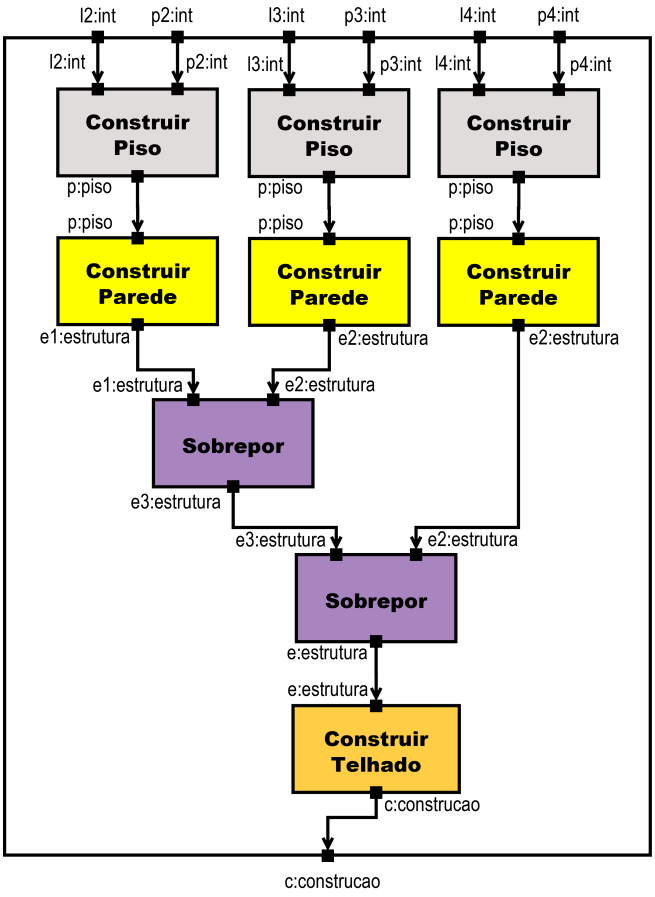


Figura 37 – Exemplo de CD Associado ao CE Abstrato.

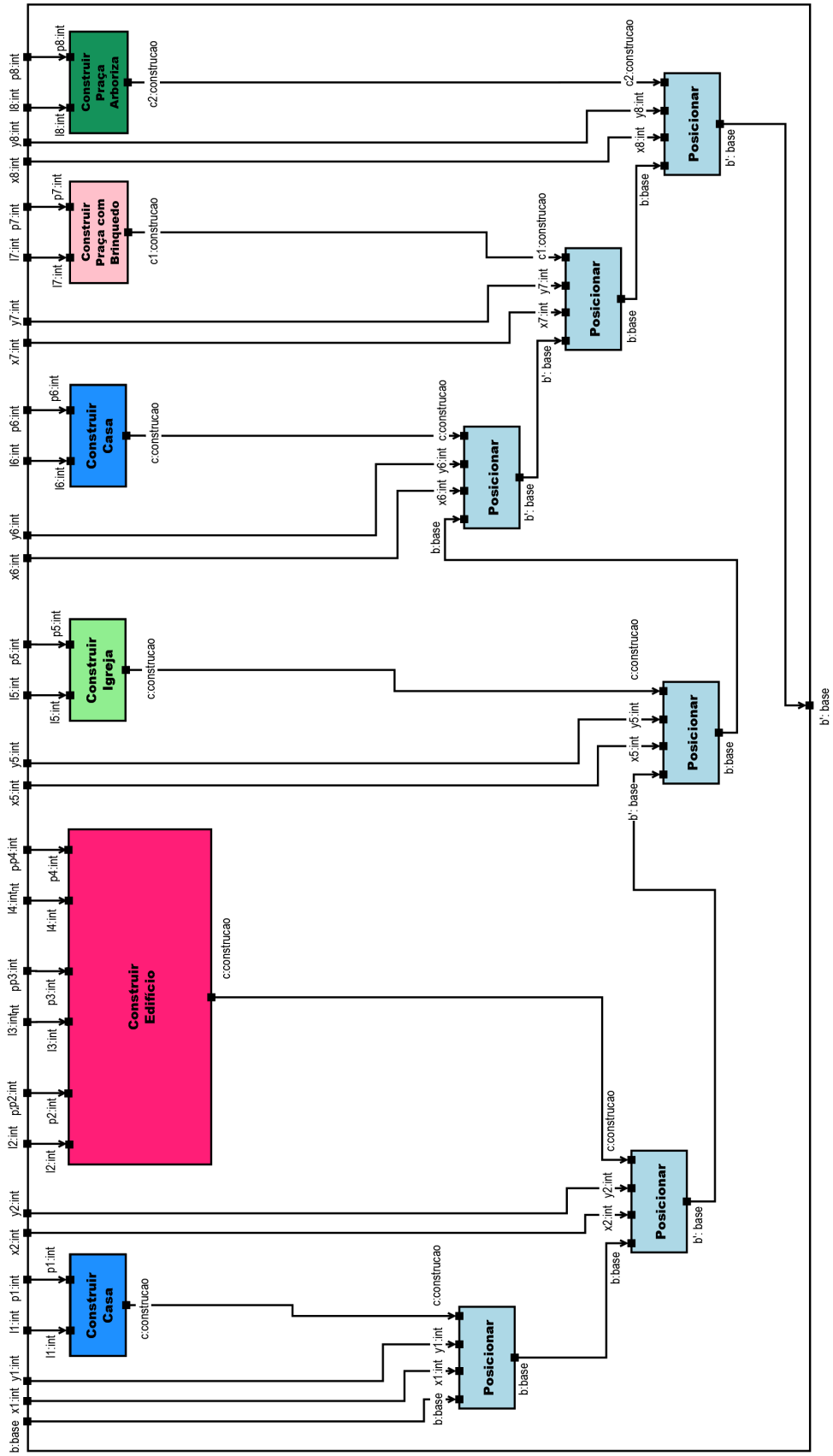


Figura 38 – Abstração de **Construir Maquete**.

5.4 Metodologia da Generalização

Na metodologia da generalização, a partir de uma solução identificam-se ações que podem ser generalizadas para resolver instâncias diferentes do mesmo problema, isto é, parte-se de uma ou mais soluções concretas e constrói-se uma solução genérica que se aplica a várias instâncias. Dado um componente decomposto D, é possível criar a sua generalização, partindo de uma cópia de D e alterando-a seguindo as etapas abaixo (sintetizadas na Figura 39):

G1 Identificar cada ação x que, ao ser substituída por um parâmetro variável, torna o componente decomposto reutilizável.

G2 Adicionar uma porta de entrada nova y para cada x, atribuindo um tipo que permita identificar a ação substituída.

G3 Caso x seja uma ação constante (que não possua entrada e sempre tenha o mesmo retorno), excluí-a e conectar y à porta de entrada que está recebendo a saída de x.

Caso contrário, excluir x e criar uma ação a_1 para a instância de x e outras ações a_i, $i = 2..n$, uma para cada variação de x. Todas as ações criadas devem ter as mesmas portas de entrada e saída que x. Sendo n o número total de ações criadas, criar os nós condicionais c_1 a c_{n-1}.

Criar uma ação b_j (com uma porta de saída booleana e uma porta de entrada do mesmo tipo de y) para cada nó condicional c_j que permita selecionar entre as variações de x. Conectar a porta y à porta de entrada de cada componente b_j e conectar a porta de saída de b_j à entrada booleana do nó de decisão de c_j. Todas as conexões que chegavam em x são conectadas às portas de entrada do nó de decisão de c_1.

Todas as portas de saída oT dos nós de decisão de c_1 até c_{n-1} são conectadas às portas de entrada de a_1 até a_{n-1}, respectivamente. Todas as portas de saída oF dos nós de decisão de c_1 até c_{n-2} são conectadas às portas de entrada dos nós de decisão de c_2 até c_{n-1}, respectivamente. Todas as portas de saída oF do nó de decisão c_{n-1} são conectadas as portas de entrada da ação a_n. Todas as portas de saída das ações a_1 até a_{n-1} são conectadas às portas de entrada iT dos nós de junção de c_1 até c_{n-1}, respectivamente. Todas as portas de saída de a_n são conectadas as portas de entrada iF do nó de junção de c_{n-1}. Todas as portas de saída dos nós de junção de c_2 até c_{n-1} são conectadas as portas de entrada iF dos nós de junção de c_1 até c_{n-2}, respectivamente. Todas as portas de saída do nó de junção c_1 são conectadas as portas que estavam conectadas com as portas de saída de x.

G4 Criar um CE associado ao CD generalizado, nomeando-o.

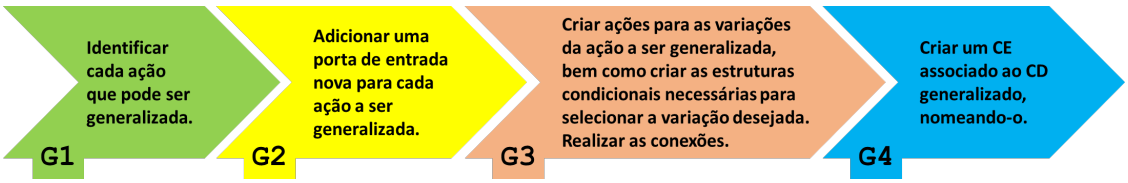


Figura 39 – Síntese da Metodologia da Generalização.

A partir do CD **Construir Praça com Brinquedo** ilustrado na Figura 40, propõe-se o uso desta metodologia, para generalizar a solução para a construção de outras praças, tais como a de ginástica e a com árvores.

No CD da Figura 40, os CEs descrevem as ações que constroem uma praça com brinquedo. Em particular, a ação **Construir Piso**, recebe duas entradas do tipo inteiro (int), indicando respectivamente largura (*l*) e profundidade (*p*), e resulta numa saída do tipo piso (*ps*). Já a ação **Colocar Brinquedo** recebe como entrada um piso (*ps*) e retorna uma construção (*c*). E a ação **Construir Cerca** recebe como entrada uma construção (*c'*) e retorna uma construção (*c*).

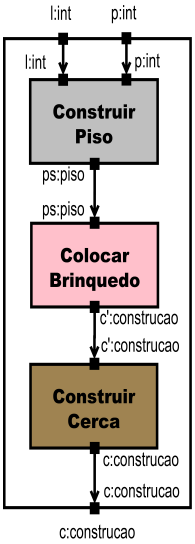


Figura 40 – CD Construir Praça com Brinquedo.

Usando a metodologia da generalização a solução pode ser obtida por meio das etapas descritas a seguir:

G1 Identificou-se que a ação **Colocar Brinquedo** pode ser generalizada para permitir a construção de outros tipos de praças (praça com academia e praça com árvores).

G2 Adicionou-se uma porta de entrada y do tipo *string* que permitirá escolher o tipo de praça a ser construída.

G3 Criou-se três ações: **Colocar Brinquedo**, **Instalar Academia** e **Plantar Arvores**, onde **Colocar Brinquedo** é a instância da ação original e as demais são as suas variações. Todas com as mesmas entradas e saídas.

Criou-se os dois nós condicionais $\langle d1, j1 \rangle$ e $\langle d2, j2 \rangle$, bem como as ações para os testes e as devidas conexões foram estabelecidas (conexões em vermelho ilustradas na Figura 41).

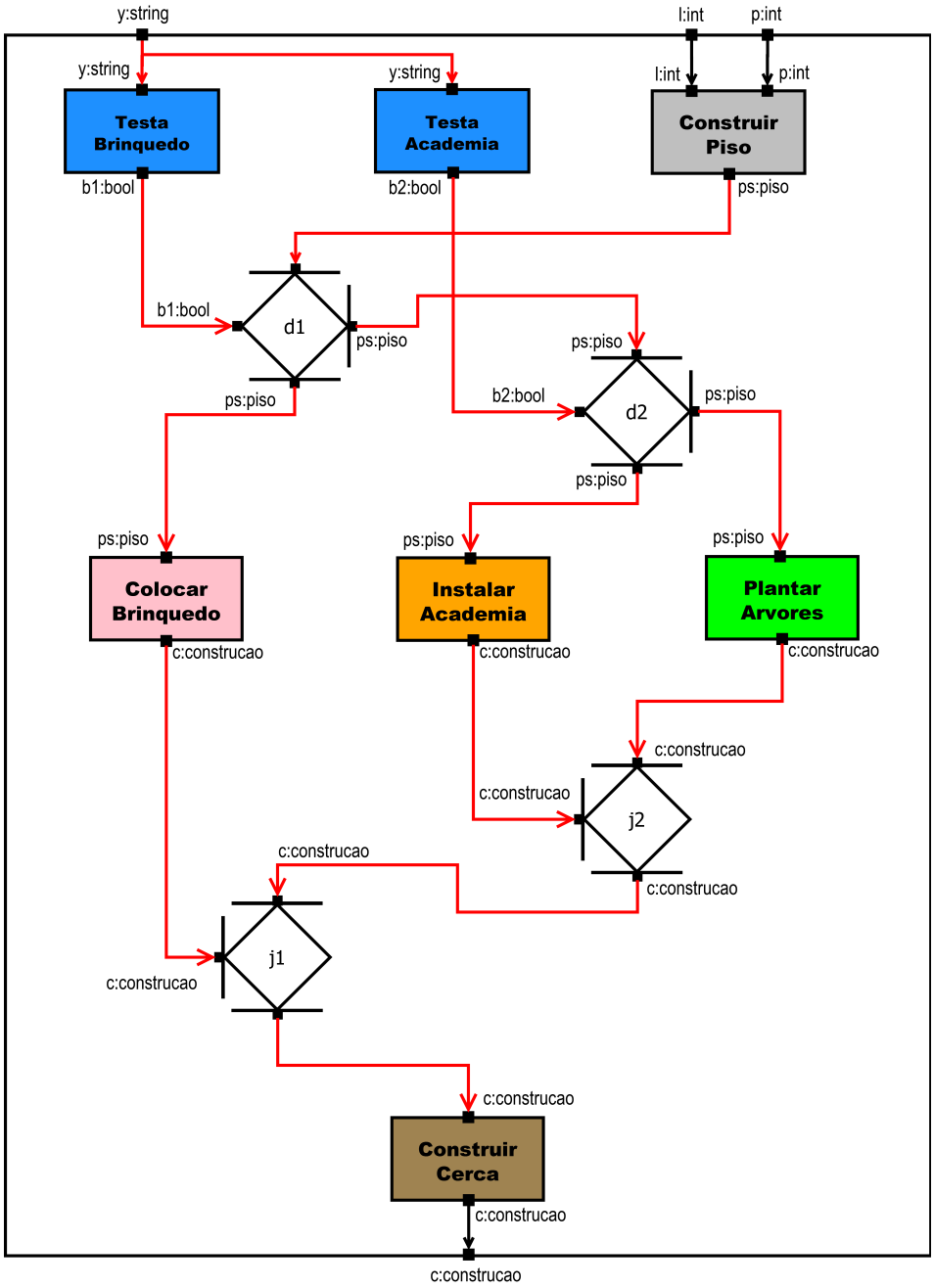


Figura 41 – Etapas G1 a G3: Generalização resultante.

G4 Criou-se um CE associado ao CD generalizado, nomeando-o como **Construir Praça** (Figura 42).

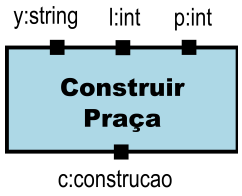


Figura 42 – Etapa G4: CE **Construir Praça**.

6 ESTUDOS DE CASO E RESULTADOS

Foram realizados três estudos de caso utilizando a linguagem LiVE e as metodologias do refinamento, decomposição e composição. O público nos três estudos foram alunos de diferentes turmas, do primeiro semestre do curso superior de Ciência da Computação da UFPel. Foram realizados diferentes estudos de caso com o objetivo de avaliar o uso da linguagem e a aplicação das metodologias propostas na solução de problemas.

Nos estudos de caso, utilizou-se de características das metodologias ativas, dando aos alunos a liberdade de escolherem a solução do problema, seguindo as etapas das metodologias propostas. Além disso, eles trabalharam em grupo permitindo a troca de informações entre os colegas, comunicação e colaboração. A ideia aqui não foi a de seguir nenhuma metodologia ativa existente e/ou citada na seção 3.3, mas trabalhar algumas características desta metodologia, de forma que aluno crie e construa o seu conhecimento através do aprender fazendo, do trabalho em equipe e da resolução de problemas, habilidades consideradas importantes para o futuro.

Cada estudo de caso envolveu um problema a ser solucionado. O primeiro estudo de caso, descrito na seção 6.1, incluiu o uso da Metodologia do Refinamento e envolveu o cálculo do lucro de uma empresa. No segundo estudo de caso, descrito na seção 6.2, apresentou-se as etapas da Composição e se trabalhou o problema de construir uma maquete. No terceiro, apresentado na seção 6.3, a metodologia da decomposição e composição foram aplicadas para resolver o problema de construir uma imagem, utilizando para isso formas geométricas básicas, tais como círculo, retângulo e triângulo.

6.1 Estudo de Caso 1: Metodologia do Refinamento

Um estudo de caso com duração de 90 minutos foi realizado com 24 alunos do primeiro semestre de um curso superior de CC da UFPel. A turma foi dividida em 3 grupos de 8 alunos, cada um organizado em 4 duplas. Foi apresentado o problema de Calcular Pro-Labore e Investimentos (descrito no quadro a seguir e solucionado

no Apêndice A.1) para que os alunos o solucionassem: um grupo (4 duplas) usando a linguagem visual e a metodologia do refinamento; um grupo (4 duplas) usando linguagem natural e um grupo (4 duplas) que não participou da atividade de resolução do problema, mas realizou simulações das soluções propostas pelo grupo que utilizou linguagem natural.

PROBLEMA:

Uma empresa deseja calcular mensalmente o valor que deve ser destinado ao pagamento de Pro Labore de seus sócios e o valor que deve ser investido na própria empresa. O valor destinado ao Pro Labore deve ser de 40% do lucro e o restante deve ser destinado a investimentos na empresa. Os valores recebidos podem ser obtidos através da listagem dos produtos vendidos durante o mês. Já os valores gastos podem ser obtidos através das notas fiscais dos insumos comprados e da listagem dos salários pagos aos funcionários no mês. Alguns custos associados devem ser considerados: ICMS é o imposto sobre circulação de mercadorias que deve ser pago sobre os produtos vendidos; e o FGTS é um valor recolhido pela empresa sobre o salário dos funcionários. O ICMS corresponde a 18% do valor bruto de vendas e o valor do FGTS é de 8% do valor dos salários pagos aos funcionários.

A apresentação da linguagem visual e da metodologia do refinamento foi feita em aproximadamente 25 minutos somente para o grupo que trabalhou nesta linguagem. Para o grupo que utilizou linguagem natural para a descrição da solução do problema, houve tempo para realizar simulações das soluções propostas. Estas simulações foram realizadas pelos colegas do terceiro grupo, sendo que estes não participaram de atividades de resolução do problema em nenhuma das linguagens. Já para o grupo que trabalhou com a linguagem visual não foi possível realizar simulações devido à necessidade de tempo para apresentação da linguagem. Para este grupo, as simulações foram realizadas posteriormente pelos próprios autores.

Diversos pontos positivos foram observados nas soluções propostas na linguagem visual. Cabe destacar: todos os grupos usaram corretamente a linguagem e chegaram em uma solução correta para o problema; apenas um dos grupos não utilizou diferentes níveis de abstração, já identificando todos os subcomponentes na etapa R2; todos os grupos identificaram corretamente as entradas e saídas, sendo que dois deles também utilizaram abstração de dados; embora os grupos tenham dado soluções diferentes para o problema, todos aplicaram o refinamento de forma adequada.

Por sua vez, nas soluções descritas usando a linguagem natural, observou-se que: nenhum grupo forneceu uma solução totalmente correta, isto é, seguindo literalmente as soluções descritas, não foi possível chegar ao resultado esperado, porém dois dos grupos descreveram uma sequência de passos que leva a uma solução parcialmente correta; algumas simulações apresentaram erros, devido à ambiguidade e detalhamento insuficiente na descrição das soluções (por exemplo, operações ficaram suben-

tendidas) ou uso de terminologias equivocadas (por exemplo, “calcular o lucro bruto da empresa listando todos os produtos vendidos”); os grupos tiveram dificuldade em estabelecer as relações entre os resultados de um passo e as entradas de outros, sendo que dois deles não conseguiram estabelecer relação alguma.

Comparando as soluções nas duas linguagens, tem-se algumas observações: os alunos que usaram a linguagem visual, por terem decomposto o problema, acabaram chegando a um nível maior de detalhamento e exatidão; e o uso da linguagem visual, exigiu que todas as conexões (dependências) entre as ações fossem especificadas, o que não ocorreu com as descrições em linguagem natural.

6.2 Estudo de Caso 2: Metodologia da Composição

O segundo estudo de caso foi realizado com 28 alunos do primeiro semestre do curso superior em CC da UFPel. Foi solicitado que os alunos utilizassem as etapas da composição para resolver o problema de construção da maquete da Figura 32 (solução descrita no Apêndice A.2). A turma foi dividida em 15 grupos, dos quais 13 duplas e 2 individuais. Cada grupo recebeu um kit, contendo: uma folha de papel tamanho 42cm $\times$ 89cm, onde deveria ser construída a solução, uma folha com a síntese das etapas da metodologia e a maquete, um tabuleiro com os CEs impressos[1] e 21 instâncias de cada um dos CEs (Figura 33). A atividade teve duração de 2h30min. A solução devia conter os CEs com suas portas de entrada e saída e as devidas conexões. Além disso, foi solicitado que indicassem em qual etapa cada CE foi adicionado.

Ao final foi aplicado um questionário individual (Apêndice B), com 7 questões objetivas e 3 expositivas, que tinham por objetivo identificar a impressão dos alunos sobre o uso da linguagem e aplicação da metodologia. As 7 questões objetivas, estão descritas na Tabela 3 e as respostas mostradas graficamente na Figura 43. A grande maioria dos alunos (84,6%) concorda que a linguagem é simples e intuitiva de ser utilizada e que a metodologia auxiliou no processo de solução do problema. Um total de 61,5% respondeu que sabia o que deveria fazer em cada passo da metodologia. Todos os estudantes afirmaram que conseguiram identificar facilmente os tipos de entradas e saídas dos componentes, assim como os componentes independentes. A maioria também afirmou que conseguiu identificar facilmente tanto a dependência entre componentes (96,2%), quanto as conexões (88,5%).

Nas respostas expositivas, observou-se que a linguagem foi bem aceita pela grande maioria dos estudantes, já que 22 alunos identificaram algumas vantagens na sua utilização: abordagem visual, intuitiva, fácil, simples, prática, entre outras. Apenas 10 alunos identificaram algumas desvantagens na sua utilização: trabalhosa, demorada, requer bastante espaço, entre outras. Dentre os comentários realizados, alguns

[1] uma folha impressa com todos os CEs e suas respectivas portas de entrada e saída

Tabela 3 – Pesquisa de uso e aplicação da linguagem.

Questões	Concordo Fortemente	Concordo	Neutro	Discordo	Discordo Fortemente
1. A linguagem é simples e intuitiva de ser utilizada.	9	13	3	1	
2. A metodologia auxiliou no processo de solução do problema.	7	15	2	1	1
3. Eu sabia o que deveria ser feito a cada passo de aplicação da metodologia.	4	12	3	6	1
4. Consegui identificar facilmente os tipos de entradas e saídas dos componentes.	18	8			
5. Consegui identificar facilmente os componentes independentes.	22	4			
6. Consegui identificar facilmente dependência entre componentes.	13	12		1	
7. Consegui identificar facilmente as conexões entre componentes.	17	6	1	1	1

alunos citaram a atividade como divertida, interessante e intuitiva.

Analisando as especificações, percebeu-se que todos os alunos conseguiram chegar à solução do problema. Houve algumas dificuldades, como mostra a Tabela 4, onde as duas mais frequentes foram: na definição dos nomes e tipos das portas de entrada e/ou saída dos componentes elementares e na definição do nome do CE principal. O mais frequente inclui: nomes iguais para portas diferentes; falta de identificação dos nomes e/ou tipos das portas de entrada e saída; e definição de tipos diferentes para portas conectadas.

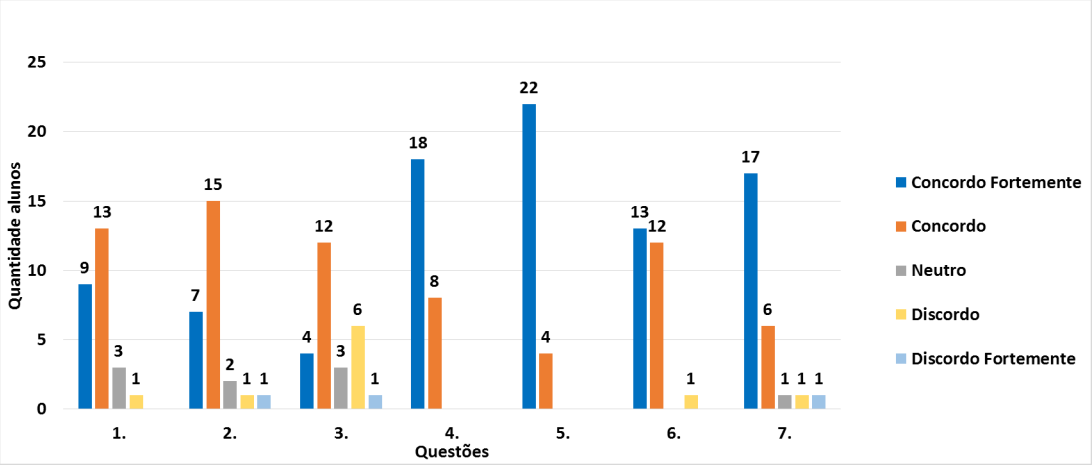


Figura 43 – Gráfico das questões objetivas.

Tabela 4 – Dificuldades na construção da especificação de uso da Metodologia da Composição.

Dificuldades	**Número Grupos**
Nomes e tipos das portas: os nomes e/ou tipos das portas não foram identificados corretamente.	6
Nome do CE principal: não nomearam o CE principal.	4
Ausência de portas: não identificaram as portas dos componentes.	2
Metodologia: não identificaram os passos da metodologia.	2
Semântica das portas: confundiram os nomes das portas com seus valores.	1
Assinatura dos CEs: não respeitaram as entradas e saídas dos CEs.	1

6.3 Estudo de Caso 3: Metodologia da Decomposição e Composição

O terceiro estudo de caso foi realizado com 37 alunos do primeiro semestre do curso superior de CC da UFPel. Inicialmente foi apresentada a linguagem e a metodologia da decomposição e composição, introduzida por meio do exemplo da Bandeira, descrito na seção 5.2. Foi solicitado que os alunos utilizassem essa metodologia para resolver o problema de construir a imagem ilustrada na Figura 44 (solução descrita no Apêndice A.3). São consideradas as seguintes ações triviais que solucionam este problema: desenhar **Retângulo** de qualquer cor e qualquer tamanho; desenhar **Triângulo** equilátero (de lados iguais) de qualquer cor e qualquer tamanho de lado; desenhar **Círculo** de qualquer cor e de qualquer diâmetro; posicionar uma figura **Ao lado** de outra figura e alinhar a borda superior das duas figuras; **Deslocar** uma figura com relação a outra, fornecendo as coordenadas (a primeira figura fica no (0,0) e a segunda é deslocada sobre a primeira); e **Girar** uma figura no sentido horário em qualquer ângulo.

A turma foi dividida em 17 grupos, dos quais 15 duplas, 1 trio e 1 individual. Cada grupo recebeu um kit, contendo: uma folha de papel tamanho 42cm $\times$ 89cm, onde deveria ser construída a solução, um formulário (etapas citadas a seguir) para preenchimento passo a passo da atividade e 7 conjuntos de cores diferentes, contendo 8 papéis de 4,2cm $\times$ 2,7cm para representar as instâncias dos CEs triviais. A atividade teve duração de 4h. A solução deveria ser realizada seguindo as etapas solicitadas no formulário de avaliação, que consistiam em 3 etapas de decomposição (geração da árvore de solução e definição dos CEs) e 3 etapas de composição (construção da solução), conforme descrito a seguir:

- D1 - Desenhar a árvore gerada pela divisão do problema em subproblemas menores.
- D2 - Completar a árvore gerada na etapa anterior, dividindo os subproblemas até que tenham soluções triviais.

- D3 - Definir um Componente Elementar (CE) para cada subproblema trivial distinto, estabelecendo entradas e saídas.
- C1 - Adicionar uma instância de CE para cada instância de subproblema trivial.
- C2 - Selecionar um dos menores subproblemas e identificar ações de combinação. Definir um CE para cada ação distinta e adicionar as instâncias necessárias, estabelecendo as conexões. Repetir até que o problema original seja resolvido.
- C3 - Definir CE para associar ao CD.

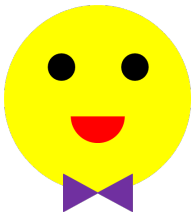


Figura 44 – Problema **Desenhar Imagem**.

A maioria dos grupos, 14 (82%), conseguiu completar o exercício com todas as ações triviais necessárias para a solução, assim como com todos os CEs, todas as conexões e todas as portas de entrada e saída. Apenas 3 grupos (18%) não conseguiram completar a solução, faltando a ação **Deslocar** e algumas conexões entre os CEs.

Com respeito a etapa D1 (desenhar uma árvore dividindo o problema em subproblemas menores), a grande maioria, 16 grupos (94%), como mostra a Figura 45, conseguiu propor uma divisão adequada a solução do problema. Apenas 1 grupo não conseguiu esboçar uma solução.

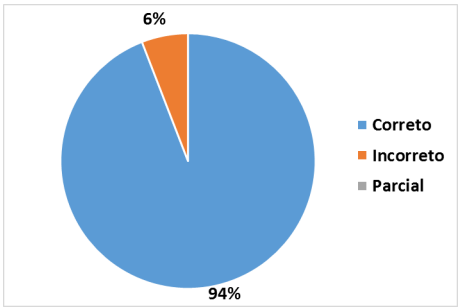


Figura 45 – Gráfico etapa D1: árvore do problema dividido em subproblemas.

Na etapa D2 (dividir os subproblemas até que eles tenham soluções triviais, representando na árvore), a maioria (88%, como ilustrado na Figura 46) desenhou a árvore completa que leva a uma solução, assim como identificaram as ações triviais. Apenas 1 grupo (6%) não conseguiu realizar a atividade e um outro grupo (6%) conseguiu

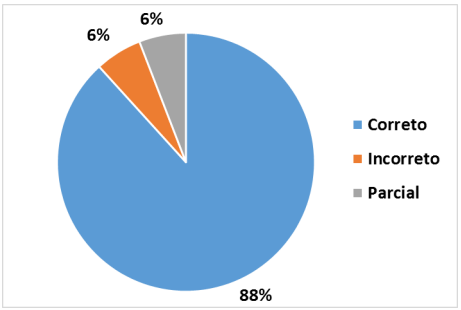


Figura 46 – Gráfico etapa D2: árvore dos subproblemas divididos até solução trivial.

chegar a uma solução parcial, faltando poucos detalhes (tal como, omitiram o desenho do círculo do rosto e não giraram os triângulos da gravata).

Na etapa D3 (definir um CE para cada subproblema trivial distinto, estabelecendo suas entradas e saída), novamente a maioria, 12 grupos (71%), definiu os CEs corretamente, identificando suas portas de entrada e saída. Um (1) grupo definiu os CEs sem portas de entrada e saída e os outros 4 grupos (24%) não definiram os CEs ou o fizeram de forma incorreta.

Nas etapas C1 a C3, onde se realiza a composição, a maioria (82%) definiu um CD completo. Quase metade, 8 grupos (47%), conseguiu chegar na solução correta do problema. Um (1) grupo não conseguiu solucionar, pois criou ações de combinação que não existiam, tais como **Abaixo** e **Sobre**, e ainda aplicou uma única ação **Girar** para os dois triângulos do laço. Os outros 8 grupos (47%) conseguiram uma solução parcial. A Tabela 5 mostra uma descrição mais detalhada dos erros mais comuns, apontando que não foram tão graves. A maior dificuldade foi na definição e uso da ação de combinação **Deslocar**, que pode não ter sido apresentada de forma clara e na definição de portas de entrada/saída.

Tabela 5 – Erros na construção do CD: etapas C1 a C3.

Erros - Etapas C1 a C3	**Quant.**
Ação Deslocar usada de forma incorreta: rosto e/ou boca não foram posicionados corretamente.	7
Ausência de portas: algumas portas foram omitidas.	4
Posicionamento das portas: posicionamento incorreto das portas de entrada dos CEs.	2
Conexões: faltaram conexões.	2
Sintaxe das portas: utilizaram representação diferente do proposto para as portas.	1
Ações de combinação: usou uma ação de combinação que não foi previamente estabelecida.	1

6.4 Considerações Finais Sobre os Estudos de Caso

Os estudos de caso permitiram identificar correções e melhorias nas descrições metodológicas: nomenclaturas foram alteradas; descrições de algumas etapas foram modificadas e/ou melhor detalhadas; a divisão das etapas foram revistas e, conforme a necessidade, reestruturadas.

Cabe destacar que algumas dessas alterações ocorreram posteriormente a algumas publicações (como (BORDINI et al., 2018) e (BORDINI; CAVALHEIRO; FOSS, 2019a,b)), o que faz com que haja diferenças de nomenclatura e/ou estruturação entre o presente texto e os artigos já publicados.

Como resultado dos três estudos de caso, podemos dizer:

- Os estudantes conseguiram utilizar a linguagem e tiveram êxito nas aplicações das metodologias para solucionar os problemas (mesmo com explicações breves);
- Muitos erros foram de representação sintática, os quais podem ser facilmente corrigidos;
- Nos três estudos a maioria dos alunos ficaram motivados para a realização das atividades, demostrando interesse e envolvimento, conseguindo assim êxito na execução da tarefa.

Alguns comentários positivos em relação ao uso da linguagem e da metodologia:

> *"Muito legal para estimular a criatividade e a capacidade de abstração dos alunos."*
> *"Bastante intuitiva, com diversas possibilidades de resolução para vários problemas."*
> *"Interessante metodologia, ela é prática e pode ter bons usos de aplicações de lógica e criação de algoritmos simples."*
> *"Interessante, mostrou um jeito simples para resolver problemas."*
> *"Atividade prática e intuitiva."*
> *"Muito interessante, o objetivo da linguagem é facilmente cumprido."*
> *"Muito legal para estimular a criatividade e capacidade de abstração dos alunos."*
> *"Interessante. É importante incluir mais a computação na educação."*
> *"Divertida e intuitiva."*
> *"A atividade foi interessante para aprender mais sobre pensamento computacional."*
> *"A atividade foi divertida de ser feita, e os passos ficaram claros conforme iam sendo realizados fisicamente."*

> *“A atividade foi interessante, proporcionou o uso da técnica de resolução de forma criativa, facilitando o entendimento da linguagem.”*

Apenas 3 alunos acharam que para problemas complexos a linguagem se tornaria muito trabalhosa de ser utilizada. Acredita-se que com o desenvolvimento de uma ferramenta para edição da linguagem esta dificuldade seja suprida.

7 CONCLUSÃO E TRABALHOS FUTUROS

Com as mudanças no cotidiano dos estudantes, novas habilidades são essenciais para resolver problemas no século XXI, tais como: criatividade e imaginação, pensamento crítico, resolução de problemas, comunicação oral e escrita, colaboração, trabalho em equipe, alfabetização tecnológica, iniciativa, responsabilidade social e ética, entre outros (Partnership for 21st Century Skills, 2019; NOURI et al., 2019; ENVISION EXPERIENCE, 2018). Concomitantemente, o PC é uma metodologia que pode auxiliar no desenvolvimento de diversas dessas habilidades, provendo técnicas principalmente para resolução de problemas.

Este trabalho propôs a linguagem de especificação visual LiVE e algumas metodologias que sistematizam a resolução de problemas fundamentadas em técnicas do Pensamento Computacional (refinamento, decomposição, composição, abstração e generalização), as quais podem ser utilizadas para resolver problemas de diversas áreas. A linguagem LiVE e as metodologias visam sistematizar o uso dos fundamentos do PC por outros profissionais, não só da computação, disseminando assim o pensar computacionalmente, uma computação para todos, como propõe Wing (2006). Por meio desta proposta, é possível especificar a solução de um problema, partindo de um alto nível de abstração, a qual vai sendo detalhada até o nível desejado. Mesmo quando o objetivo é automatizar a solução de um problema, a especificação constitui, em geral, o primeiro passo para a solução. Na definição da linguagem detalhou-se os seus componentes visuais e especificou-se formalmente a sintaxe e a semântica. Estas definições ajudam na implementação de um compilador/interpretador para LiVE.

A linguagem LiVE foi desenvolvida com o intuito de ser usada em todos os níveis de ensino, desde a educação básica até o ensino superior. Estudos de caso foram desenvolvidos para avaliar o uso, por parte de estudantes, da linguagem bem como as metodologias propostas. Tais avaliações permitiram tanto o aprimoramento da linguagem quanto das metodologias.

LiVE foi inspirada em características de três linguagens visuais da Computação: DFD, Diagramas de Atividades da UML e o Simulink. A Tabela 6 destaca algumas características abordadas em cada uma delas, apontando aquelas presentes na lin-

guagem LiVE. Todas as linguagens possuem representação para processos, fluxo de dados e regras de decisão. Por outro lado, enquanto essas linguagens representam a repetição por meio de ciclos, LiVE se utiliza da recursão para especificar repetições. Assim como no Simulink o fluxo de controle do LiVE é inferido pelo fluxo de dados. Além disso, LiVE também possui uma composição hierárquica. Diferentemente do Simulink, cujos tipos de dados são restritos aos tipos do MatLab, em LiVE esta representação é livre e definida pelo usuário. Diferenciando-se das três linguagens, LiVE não tem o foco no desenvolvimento de sistemas de computação e possui componentes que permitem definir refinamento e abstração de dados.

Tabela 6 – Características entre LiVE e as linguagens.

Características/Linguagens	**DFD**	**UML**	**Simulink**	**LiVE**
Processos/ações	X	X	X	X
Fluxo de dados	X	X	X	X
Armazenamento de dados	X	-	-	-
Regras de decisão	X	X	X	X
Estrutura de repetição	ciclos	ciclos	ciclos	recursão
Fluxo de controle	explícito	explícito	implícito	implícito
Composição hierárquica	-	-	X	X
Tipos de dados	-	-	restritos ao MatLab	diversos
Refinamento/abstração dados	-	-	-	X

Em LiVE, com a liberdade de definir e nomear os dados, existem tipos de dados que não podem ser especificados nas outras linguagens. Por exemplo, um tipo de dado definido em LiVE foi “boloCoberto”, especificado como o tipo de saída da ação **Fazer Bolo com Cobertura**. Tal recurso possibilita qualquer pessoa, desde uma criança até um profissional de qualquer área, definir e utilizar os mais diversos tipos de dados.

Com LiVE e as metodologias, é possível identificar e talvez mensurar uma solução parcial de um problema, avaliando até onde o aluno consegue chegar. Por exemplo, pode ser que ele não consiga especificar a solução esperada, por exemplo, um bolo com massa preta e com cobertura de brigadeiro, mas se ele conseguiu especificar a solução da confecção de um bolo preto, é possível definir estratégias de avaliação parcial e de auxílio para a conclusão da solução.

Nos estudos de caso, observou-se que a maioria dos estudantes conseguiram utilizar a linguagem e as metodologias corretamente, chegando à solução dos problemas propostos. Das vantagens citadas, em relação ao uso da linguagem e das metodologias, as que se destacaram foram: facilidade, simplicidade, estímulo do raciocínio lógico, aprendizado rápido e intuitivo. Por outro lado, observou-se que o uso da linguagem e das metodologias de forma desplugada se torna viável para abordar problemas cuja especificação não seja muito extensa.

O foco deste trabalho foi dado na definição da linguagem e proposição das metodo-

logias. As propostas metodológicas e planos didáticos para introdução da abordagem na Educação Básica constitui-se de um trabalho futuro, pois a linguagem visual e as metodologias criadas necessitam ser adaptadas para se tornarem mais lúdicas e práticas de serem utilizadas em diferentes níveis de ensino. As metodologias também são um suporte para o educador criar atividades (planos de ensino) para trabalhar habilidades do PC.

Também como trabalho futuro, destacam-se: a definição completa da semântica (incluindo estruturas recursivas); a extensão da linguagem com *fork* e *join* para tratar paralelismo; a extensão das metodologias para outras estratégias do PC, tais como, simulação, paralelismo e automação; a implementação de um interpretador/compilador para a linguagem; o desenvolvimento de uma ferramenta de edição que dê suporte ao uso das metodologias; e a consolidação de um processo de avaliação do uso da linguagem LiVE, onde seja possível mensurar o quanto de cada habilidade do PC foi desenvolvida em cada solução.

REFERÊNCIAS

ABRIAL, J.-R. **Modeling in Event-B**: system and software engineering. [S.l.]: Cambridge University Press, 2010.

ALMEIDA, M. E. B. d.; TEIXEIRA, A. R. A.; ALMEIDA, R. Work in progress: Improving learning performance using programming methodology. In: IEEE GLOBAL ENGINEERING EDUCATION CONFERENCE, 2019. **Proceedings...** [S.l.: s.n.], 2019. p.1462–1466.

ALTAHER, M.; FERCHICHI, A. AlgoThink: An Algorithmic Computational Thinking Approach. In: JOINT INTERNATIONAL CONFERENCE ON ICT IN EDUCATION AND TRAINING, INTERNATIONAL CONFERENCE ON COMPUTING IN ARABIC, AND INTERNATIONAL CONFERENCE ON GEOCOMPUTING, 2018. **Proceedings...** [S.l.: s.n.], 2018. p.1–7.

ANDRADE, D. et al. Proposta de Atividades para o Desenvolvimento do Pensamento Computacional no Ensino Fundamental. In: WORKSHOP DE INFORMÁTICA NA ESCOLA, 2013. **Anais...** [S.l.: s.n.], 2013. p.169–178.

BENDER, W. N. **Aprendizagem baseada em projetos**: educação diferenciada para o século XXI. [S.l.]: Penso Editora, 2014.

BOCK, C. UML 2 activity and action models, Part 2: Actions. **Journal of Object Technology**, [S.l.], v.2, n.5, p.41–56, 2003.

BORDINI, A.; CAVALHEIRO, S.; AVILA, C.; FOSS, L. Linguagem Visual para Resolução de Problemas Fundamentada no Pensamento Computacional: uma proposta. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2018. **Anais...** [S.l.: s.n.], 2018. p.81–90.

BORDINI, A.; CAVALHEIRO, S.; FOSS, L. Metodologia de resolução de problemas utilizando a linguagem LiVE. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2019. **Anais...** [S.l.: s.n.], 2019. p.259–268.

BORDINI, A.; CAVALHEIRO, S.; FOSS, L. Proposta de uma metodologia de generalização para resolução de problemas utilizando a linguagem LiVE. In: WORKSHOP ESCOLA DE INFORMÁTICA TEÓRICA, 2019. **Anais...** [S.l.: s.n.], 2019.

BORDINI, A. et al. Computação na Educação Básica no Brasil: o Estado da Arte. **Revista de Informática Teórica e Aplicada**, [S.l.], v.23, n.2, p.210–238, 2016a.

BORDINI, A. et al. Desdobramentos do Pensamento Computacional no Brasil. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2016b. **Anais...** [S.l.: s.n.], 2016b. p.200–209.

BORDINI, A. et al. Pensamento Computacional nos Ensinos Fundamental e Médio: uma revisão sistemática. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2017. **Anais...** [S.l.: s.n.], 2017. p.123–132.

BRASIL. **Base Nacional Comum Curricular**. Disponível em: `http://basenacionalcomum.mec.gov.br`. Acesso em: 20 de março de 2020.

BURNETT, M. M. Visual programming. **Wiley Encyclopedia of Electrical and Electronics Engineering**, [S.l.], 1999.

CAS. **Computational thinking, A guide for teachers**. Disponível em: `http://community.computingatschool.org.uk/resources/2324/single`. Acesso em: 19 de fevereiro de 2019.

CESAR, E. et al. Introducing computational thinking, parallel programming and performance engineering in interdisciplinary studies. **Journal of Parallel and Distributed Computing**, [S.l.], v.105, p.116–126, 2017.

CHEN, J.; SHEN, L.; YIN, J.; ZHANG, C. Parallel programming course development based on parallel computational thinking. In: ACM TURING CELEBRATION CONFERENCE-CHINA, 2018. **Proceedings...** [S.l.: s.n.], 2018. p.103–109.

CRYSTAL, D. **A revolução da linguagem**. [S.l.]: Zahar, 2005.

DEMO, P. **O porvir**: desafio das linguagens do século XXI. [S.l.]: IBPEX, 2007.

DORLING, M.; WHITE, D. Scratch: A way to logo and python. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 46., 2015. **Proceedings...** [S.l.: s.n.], 2015. p.191–196.

ENVISION EXPERIENCE. **13 Essential 21st Century Skills for Todays Students**. Disponível em: `https://goo.gl/xnSrXE`. Acesso em: 17 de março de 2019.

FIGUEIREDO, J.; GOMES, N.; GARCÍA-PEÑALVO, F. J. Ne-course for learning programming. In: FOURTH INTERNATIONAL CONFERENCE ON TECHNOLOGICAL ECOSYSTEMS FOR ENHANCING MULTICULTURALITY, 2016. **Proceedings...** [S.l.: s.n.], 2016. p.549–553.

FLOS, J. A.; VILAHUR, J. V. eSeeCode: Creating a Computer Language from Teaching Experiences. **Olympiads in Informatics**, [S.l.], v.10, p.3–18, 2016.

FOSS, L. et al. From uml to simulink caam: Formal specification and transformation analysis. **Revista de Informática Teórica e Aplicada**, [S.l.], v.20, n.1, p.102–139, 2013.

GARDELI, A.; VOSINAKIS, S. Creating the computer player: an engaging and collaborative approach to introduce computational thinking by combining unplugged activities with visual programming. **Italian Journal of Educational Technology**, [S.l.], v.25, n.2, p.36–50, 2017.

GIORDANO, D.; MAIORANA, F. Teaching algorithms: Visual language vs flowchart vs textual language. In: IEEE GLOBAL ENGINEERING EDUCATION CONFERENCE, 2015. **Proceedings...** [S.l.: s.n.], 2015. p.499–504.

GOLIN, E. J.; REISS, S. P. The specification of visual language syntax. **Journal of Visual Languages & Computing**, [S.l.], v.1, n.2, p.141–157, 1990.

GONÇALVES, J. et al. Educational Robotics Summer Camp at IPB: A Challenge based learning case study. In: SEVENTH INTERNATIONAL CONFERENCE ON TECHNOLOGICAL ECOSYSTEMS FOR ENHANCING MULTICULTURALITY, 2019. **Proceedings...** [S.l.: s.n.], 2019. p.36–43.

GONZÁLEZ, Y. A. C.; MUÑOZ-REPISO, A. G.-V. Development of computational thinking skills and collaborative learning in initial education students through educational activities supported by ICT resources and programmable educational robots. In: INTERNATIONAL CONFERENCE ON TECHNOLOGICAL ECOSYSTEMS FOR ENHANCING MULTICULTURALITY, 5., 2017. **Proceedings...** [S.l.: s.n.], 2017. p.1–6.

GROVER, S.; PEA, R.; COOPER, S. Factors influencing computer science learning in middle school. In: ACM TECHNICAL SYMPOSIUM ON COMPUTING SCIENCE EDUCATION, 47., 2016. **Proceedings...** [S.l.: s.n.], 2016. p.552–557.

HAREL, D.; MARRON, A. Toward scenario-based algorithmics. In: **Adventures Between Lower Bounds and Higher Altitudes**. [S.l.]: Springer, 2018. p.549–567.

HOWLAND, K.; GOOD, J.; NICHOLSON, K. Language-based support for computational thinking. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES AND HUMAN-CENTRIC COMPUTING, 2009. **Proceedings...** [S.l.: s.n.], 2009. p.147–150.

HUANG, W.; DENG, Z.; RONGSHENG, D. Programming courses teaching method for ability enhancement of computational thinking. In: INTERNATIONAL ASSOCIATION OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY-SPRING CONFERENCE, 2009. **Proceedings...** [S.l.: s.n.], 2009. p.182–185.

CHANG, S.-K.; ICHIKAWA, T.; LIGOMENIDES, P. A. (Ed.). **Introduction**: Visual Languages and Iconic Languages. Boston: Springer, 1986. p.1–7.

ISTE; CSTA. **Computational thinking leadership toolkit**. Disponível em: `https://id.iste.org/docs/ct-documents/ct-leadershipt-toolkit.pdf?sfvrsn=4`. Acesso em: 15 de março de 2020.

KAFAI, Y. B.; BURKE, Q. The social turn in K-12 programming: moving from computational thinking to computational participation. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 44., 2013. **Proceedings...** [S.l.: s.n.], 2013. p.603–608.

KITCHENHAM, B. A. Procedures for Performing Systematic Reviews. In: KEELE, UK, KEELE UNIVERSITY, 2004. **Anais...** [S.l.: s.n.], 2004. p.1–26.

KOH, K. H.; BASAWAPATNA, A.; BENNETT, V.; REPENNING, A. Towards the automatic recognition of computational thinking for adaptive visual language learning. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES AND HUMAN-CENTRIC COMPUTING, 2010. **Proceedings...** [S.l.: s.n.], 2010. p.59–66.

LAKANEN, A.-J.; ISOMÖTTÖNEN, V. What Does It Take to Do Computer Programming?: Surveying the K-12 Students' Conceptions. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 46., 2015. **Proceedings...** [S.l.: s.n.], 2015. p.458–463.

LEE, M. J. et al. Principles of a debugging-first puzzle game for computing education. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES AND HUMAN-CENTRIC COMPUTING, 2014. **Proceedings...** [S.l.: s.n.], 2014. p.57–64.

LUDOVICO, L. A.; MALCHIODI, D.; ZECCA, L. A multimodal LEGO®-based learning activity mixing musical notation and computer programming. In: ACM SIGCHI INTERNATIONAL WORKSHOP ON MULTIMODAL INTERACTION FOR EDUCATION, 1., 2017. **Proceedings...** [S.l.: s.n.], 2017. p.44–48.

MALIZIA, A. et al. TAPASPlay: A game-based learning approach to foster computation thinking skills. In: IEEE SYMPOSIUM ON VISUAL LANGUAGES AND HUMAN-CENTRIC COMPUTING, 2017. **Proceedings...** [S.l.: s.n.], 2017. p.345–346.

MAQUIL, V.; MOLL, C.; SCHWARTZ, L.; HERMEN, J. Kniwwelino: A Lightweight and WiFi Enabled Prototyping Platform for Children. In: TWELFTH INTERNATIONAL CONFERENCE ON TANGIBLE, EMBEDDED, AND EMBODIED INTERACTION, 2018. **Proceedings...** [S.l.: s.n.], 2018. p.94–100.

MARQUES, M. et al. Uma Proposta para o Desenvolvimento do Pensamento Computacional Integrado ao Ensino de Matemática. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2017. **Anais...** [S.l.: s.n.], 2017. p.314–323.

MARRIOTT, K.; MEYER, B. **Visual language theory**. [S.l.]: Springer Science & Business Media, 1998.

MARTINS, I.; GOUVÊA, G.; PICCININI, C. Aprendendo com imagens. **Ciência e Cultura**, [S.l.], v.57, n.4, p.38–40, 2005.

MARTINS, L. et al. Ensinando Lógica de Programação aplicada a Robótica para alunos do Ensino Fundamental. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2016. **Anais...** [S.l.: s.n.], 2016. p.31–41.

MATHWORKS. **Simulink**. Disponível em: `https://www.mathworks.com/products/simulink.html`. Acesso em: 15 de fevereiro de 2020.

MAZUR, E. **Peer instruction**: a revolução da aprendizagem ativa. [S.l.]: Penso Editora, 2015.

MENG, F.; CHU, D.; ZHAN, D. Transformation from Data Flow Diagram to UML2.0 activity diagram. In: IEEE INTERNATIONAL CONFERENCE ON PROGRESS IN INFORMATICS AND COMPUTING, 2010. **Proceedings...** [S.l.: s.n.], 2010. v.2, p.1010–1014.

MIOTO, F. et al. bASES21-Um Modelo para a Autoavaliação de Habilidades do Século XXI no Contexto do Ensino de Computação na Educação Básica. **Revista Brasileira de Informática na Educação**, [S.l.], v.27, n.01, p.26, 2019.

MOHAGHEGH, D. M.; MCCAULEY, M. Computational thinking: the skill set of the 21st century. **International Journal of Computer Science and Information Technologies**, [S.l.], p.1524–1530, 2016.

MORAN, J. Mudando a educação com metodologias ativas. **Coleção mídias contemporâneas. Convergências midiáticas, educação e cidadania: aproximações jovens**, [S.l.], v.2, n.1, p.15–33, 2015.

MORAN, J. M. **Mudar a forma de ensinar e de aprender**. 57–72p. v.V, n.9. Disponível em: http://www.eca.usp.br/prof/moran/site/textos/tecnologias_eduacacao/uber.pdf. Acesso em: 15 de maio de 2019.

MORAN, J. M. **A educação que desejamos**: novos desafios e como chegar lá. [S.l.]: Papirus Educação, 2007.

MORAN, J. M. **Educação Transformadora**. Disponível em: http://www2.eca.usp.br/moran/. Acesso em: 15 de maio de 2019.

MORAN, J. M. **Educação do Futuro**. Disponível em: http://www2.eca.usp.br/moran/wp-content/uploads/2019/09/educa%C3%A7ao_futuro.pdf. Acesso em: 15 de março de 2020.

MORESI, E. A. D.; OLIVEIRA BRAGA FILHO, M. de; BARBOSA, J. A.; HARTMANN, V. C. Metodologias ativas de ensino e aprendizagem: o emprego da aprendizagem baseada em desafios na elaboração de revisão de literatura. **Indagatio Didactica**, [S.l.], v.11, n.3, p.57–78, 2019.

MOTA, F. et al. Desenvolvendo o Raciocínio Lógico no Ensino Médio: uma proposta utilizando a ferramenta Scratch. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2014. **Anais...** [S.l.: s.n.], 2014. p.377–381.

NICHOLS, M.; CATOR, K.; TORRES, M. Challenge based learner user guide. **Redwood City, CA: Digital Promise**, [S.l.], p.24–36, 2016.

NOURI, J.; ZHANG, L.; MANNILA, L.; NORÉN, E. Development of computational thinking, digital competence and 21st century skills when learning programming in K-9. **Education Inquiry**, [S.l.], p.1–17, 2019.

OLIVEIRA, E. et al. Pensamento Computacional e Robótica: Um Estudo Sobre Habilidades Desenvolvidas em Oficinas de Robótica Educacional. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 2016. **Anais...** [S.l.: s.n.], 2016. p.530–539.

OMG. **UML 2.5.1**. Disponível em: https://www.omg.org/spec/UML/2.5.1/PDF. Acesso em: 15 de março de 2020.

PAPADOPOULOS, Y.; TEGOS, S. Using microworlds to introduce programming to novices. In: PANHELLENIC CONFERENCE ON INFORMATICS, 16., 2012. **Proceedings...** [S.l.: s.n.], 2012. p.180–185.

PAPERT, S. **A máquina das crianças**: repensando a escola na era da informática. [S.l.]: Artes Médicas, 1994.

Partnership for 21st Century Skills. **Framework for 21st Century Learning**. Disponível em: http://static.battelleforkids.org/documents/p21/P21_Framework_DefinitionsBFK.pdf. Acesso em: 15 de março de 2020.

PELLAS, N.; VOSINAKIS, S. The effect of simulation games on learning computer programming: A comparative study on high school students' learning performance by assessing computational problem-solving strategies. **Education and Information Technologies**, [S.l.], p.1–30, 2018.

PÉREZ-MARÍN, D.; HIJÓN-NEIRA, R.; MARTÍN-LOPE, M. A Methodology Proposal based on Metaphors to teach Programming to children. **IEEE Revista Iberoamericana de Tecnologias del Aprendizaje**, [S.l.], v.13, n.1, p.46–53, 2018.

PRESSMAN, R. **Engenharia de Software**. 7.ed. [S.l.]: McGraw, 2011.

REZENDE, C. M.; BISPO, E. L. Comparison between the use of pseudocode and visual programming in programming teaching: An evaluation from scratch tool. In: IBERIAN CONFERENCE ON INFORMATION SYSTEMS AND TECHNOLOGIES, 13., 2018. **Proceedings...** [S.l.: s.n.], 2018.

RIBEIRO, L.; FOSS, L.; CAVALHEIRO, S. A. d. C. **Entendendo o pensamento computacional**. Disponível em: https://arxiv.org/abs/1707.00338. Acesso em: 23 de maio de 2018.

RICH, K. M.; BINKOWSKI, T. A.; STRICKLAND, C.; FRANKLIN, D. Decomposition: A k-8 computational thinking learning trajectory. In: ACM CONFERENCE ON INTERNATIONAL COMPUTING EDUCATION RESEARCH, 2018. **Proceedings...** [S.l.: s.n.], 2018. p.124–132.

RODRIGUES, J. F. R. et al. Parallel Computing: Unplugging to Learn. In: XIII LATIN AMERICAN CONFERENCE ON LEARNING TECHNOLOGIES, 2018. **Proceedings...** [S.l.: s.n.], 2018. p.41–44.

ROJAS-LÓPEZ, A.; GARCÍA-PEÑALVO, F. J. Learning scenarios for the subject Methodology of Programming from evaluating the Computational Thinking of new students. **IEEE Revista Iberoamericana de Tecnologias del Aprendizaje**, [S.l.], v.13, n.1, p.30–36, 2018.

SÁEZ-LÓPEZ, J.-M.; ROMÁN-GONZÁLEZ, M.; VÁZQUEZ-CANO, E. Visual programming languages integrated across the curriculum in elementary school: A two year case study using scratch in five schools. **Computers & Education**, [S.l.], v.97, p.129–141, 2016.

SAVERY, J. R. Overview of problem-based learning: Definitions and distinctions. **Interdisciplinary Journal of Problem-Based Learning**, [S.l.], v.9, p.5–15, 2015.

SENSKE, N. Evaluation and Impact of a Required Computational Thinking Course for Architecture Students. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 2017. **Proceedings...** [S.l.: s.n.], 2017. p.525–530.

SHU, N. C. **Visual programming languages**: A perspective and a dimensional analysis. [S.l.]: Visual Languages, 1986. p.11–34.

SLONNEGER, K.; KURTZ, B. **Formal syntax and semantics of programming languages**. [S.l.]: Addison-Wesley Reading, 1995.

SU, J.-M.; WANG, S.-J. A Web-Based learning activity integrated with scratch tool to support programming learning. In: INTERNATIONAL CONFERENCE ON UBI-MEDIA COMPUTING AND WORKSHOPS, 10., 2017. **Proceedings...** [S.l.: s.n.], 2017. p.1–4.

SULLIVAN, A.; BERS, M.; PUGNALI, A. The impact of user interface on young children's computational thinking. **Journal of Information Technology Education: Innovations in Practice**, [S.l.], v.16, n.1, p.171–193, 2017.

SULLIVAN, K. et al. CodePlus-Designing an after school computing programme for girls. In: IEEE FRONTIERS IN EDUCATION CONFERENCE, 2015. **Proceedings...** [S.l.: s.n.], 2015. p.1–5.

TABESH, Y. Computational Thinking: A 21st Century Skill. **Olympiads in Informatics**, [S.l.], v.11, p.65–70, 2017.

TSORTANIDOU, X.; DARADOUMIS, T.; BARBERÁ, E. Connecting moments of creativity, computational thinking, collaboration and new media literacy skills. **Information and Learning Sciences**, [S.l.], 2019.

URRUTIA, E. K. M.; OCAÑA, J. M.; PÉREZ-MARÍN, D.; TAMAYO, S. A first proposal of Pedagogic Conversational Agents to develop Computational Thinking in children. In: INTERNATIONAL CONFERENCE ON TECHNOLOGICAL ECOSYSTEMS FOR ENHANCING MULTICULTURALITY, 5., 2017. **Proceedings...** [S.l.: s.n.], 2017. p.1–6.

VALENTE, J. A.; ALMEIDA, M. E. B. de; GERALDINI, A. F. S. Metodologias ativas: das concepções às práticas em distintos níveis de ensino. **Revista Diálogo Educacional**, [S.l.], v.17, n.52, p.455–478, 2017.

WANGENHEIM, C. Gresse von et al. Desenvolvimento e Avaliação de um Jogo de Tabuleiro para Ensinar o Conceito de Algoritmos na Educação Básica. **Revista Brasileira de Informática na Educação**, [S.l.], v.27, n.03, p.310–335, 2019.

WARD, P. T. The transformation schema: An extension of the data flow diagram to represent control and timing. **IEEE Transactions on Software Engineering**, [S.l.], n.2, p.198–210, 1986.

WELLER, M. P.; DO, E. Y.-L.; GROSS, M. D. Escape machine: teaching computational thinking with a tangible state machine game. In: INTERACTION DESIGN AND CHILDREN, 7., 2008. **Proceedings...** [S.l.: s.n.], 2008. p.282–289.

WING, J. M. Computational Thinking. **Communications of the ACM**, [S.l.], v.49, n.3, p.33–35, 2006.

WING, J. M. Computational thinking and thinking about computing. In: PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY OF LONDON A: MATHEMATICAL, PHYSICAL AND ENGINEERING SCIENCES, 2008. **Proceedings...** IEEE, 2008. v.366, n.1881, p.3717–3725.

YADAV, A. et al. Computational thinking for all: pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. **TechTrends**, [S.l.], v.60, n.6, p.565–568, 2016.

Apêndices

APÊNDICE A – Soluções dos Problemas dos Estudos de Caso

Neste apêndice são apresentadas soluções para os problemas abordados nos estudos de caso. Cabe observar que para cada problema, há diversas soluções possíveis, e apenas uma possibilidade é descrita.

A.1 Solução do Estudo de Caso 1: Metodologia do Refinamento

Abaixo segue a solução do estudo de caso utilizando a metodologia do refinamento para resolver o problema de Calcular o Pro-Labore e Investimentos de uma empresa.

R1 Nesta etapa, no nível mais alto de abstração, descreveu-se o componente elementar principal, conforme Figura 47, identificando-se a ação a ser executada (**Calcular Pro-Labore e Investimentos**), as informações de entrada (os *produtos vendidos*, os *insumos* e os *salarios*) e os resultados/saídas (o *pro-labore* e o *investimento*).

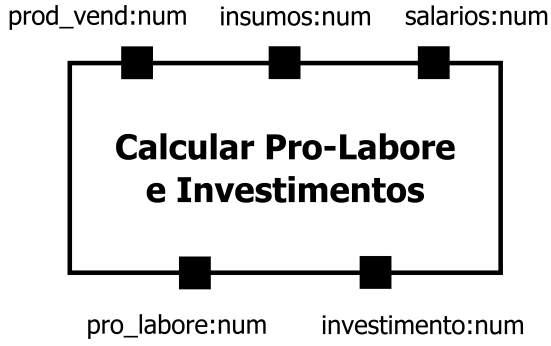


Figura 47 – Etapa R1: CE Calcular Pro-Labore e Investimentos.

R2 Nesta etapa, em um nível mais baixo de abstração, detalhou-se a ação **Calcular Pro-Labore e Investimentos** em um conjunto de ações que levam ao resultado esperado. Neste nível identificou-se quais ações menores compõem a ação principal. Neste caso, a ação **Calcular Pro-Labore e Investimentos** foi decomposta em 3 ações: **Valor Recebido**, **Valor Gasto** e **Lucro**.

São especificadas as conexões entre as instâncias dos CE. Neste caso, a ação **Lucro** depende dos resultados das ações **Valor Recebido** e **Valor Gasto**, o que pode ser observado pelas conexões representadas na Figura 48. Também é possível observar que as ações **Valor Recebido** e **Valor Gasto** são independentes

entre si, já que as entradas de ambas não são resultados de outros componentes elementares.

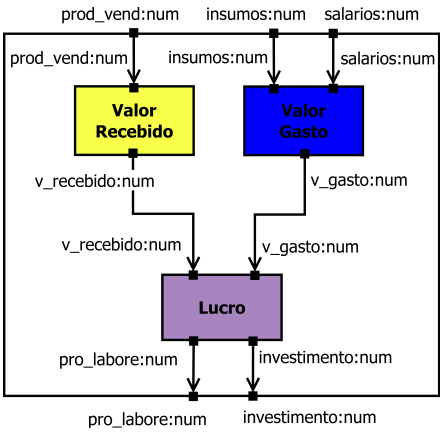


Figura 48 – Etapa R2: CD Calcular Pro-Labore e Investimentos.

R3 Nesta etapa, refinou-se as ações que ainda precisam ser mais detalhadas para o cálculo do Pro-labore e Investimentos, neste caso, **Valor Gasto** e **Lucro**, especificando-se um componente decomposto para cada uma, como mostra as Figuras 49 e 50.

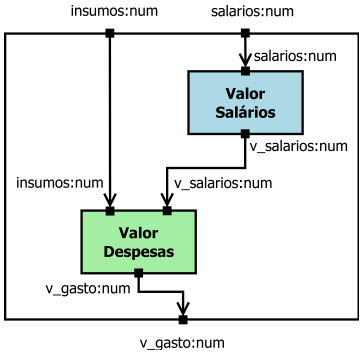


Figura 49 – Etapa R3: CD Valor Gasto.

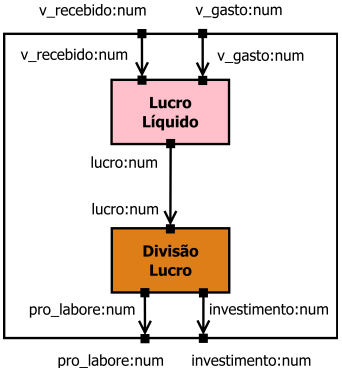


Figura 50 – Etapa R3: CD Lucro.

Tabela 7 – Mudanças nas etapas da Metodologia da Decomposição e Composição.

Inicial	**Final**
C1: Adicionar CE com entradas independentes.	C1: Adicionar uma instância de CE para cada instância de subproblema trivial.
C2: Adicionar/selecionar componentes que se conectem às saídas descobertas, estabelecendo as conexões.	C2: Selecionar um dos menores subproblemas e identificar ações de combinação. Definir um CE para cada ação distinta e adicionar as instâncias necessárias, estabelecendo as conexões. Repetir até que o problema original seja resolvido.
C3: Repetir C2.	
C4: Repetir C1, C2 e C3.	
C5: Definir o CD contendo todos os componentes.	
C6: Definir CE para associar ao CD.	C3: Definir o CE para associar ao CD.

A.2 Solução do Estudo de Caso 2: Metodologia da Composição

Nesta seção é apresentada a solução do problema **Construir Maquete**. Esta solução está apresentada na primeira versão da Metodologia da Composição (publicada no artigo (BORDINI; CAVALHEIRO; FOSS, 2019a)), onde a Decomposição e Composição eram propostas como metodologias individuais. A Tabela 7 mostra a versão inicial da Metodologia da Composição e sua relação com a versão final.

C1 Foram adicionadas sete instâncias de **Construir Piso** e uma instância de **Construir Solo**.

C2 Foram adicionadas: seis instâncias de **Construir Parede**, que são conectadas às instâncias 1 à 6 de **Construir Piso**; uma instância de **Arborizar**, a qual é conectada a **Construir Solo**; e uma instância de **Colocar Brinquedo**, que é conectada a **Construir Piso**$_7$.

C3 A etapa C2 foi repetida 5 vezes (C3.1 à C3.5 na Figura 51). Na primeira repetição foram adicionadas: as instâncias **Construir Telhado**$_1$, **Construir Telhado**$_2$ e **Construir Telhado**$_3$, conectadas a **Construir Parede**$_1$, **Construir Parede**$_5$ e **Construir Parede**$_6$, respectivamente; a instância **Sobrepor**$_1$ conectada às instâncias **Construir Parede**$_2$ e **Construir Parede**$_3$; a instância **Sobrepor**$_2$, conectada à instância **Construir Parede**$_4$; e as instâncias **Posicionar**$_1$ e **Posicionar**$_2$, conectadas às instâncias **Colocar Brinquedo** e **Arborizar**, respectivamente. Na segunda repetição foram adicionadas: as instâncias **Posicionar**$_3$ e **Posicionar**$_4$, conectadas às instâncias **Construir Telhado**$_1$ e **Construir Telhado**$_3$, respectivamente; a instância **Construir Telhado**$_4$ conectada a **Sobrepor**$_2$; e a instância **Construir Torre**$_1$, conectada a **Construir Telhado**$_2$. Nesta repetição ainda foram selecionadas algumas ações (adicionadas em etapas anteriores) para estabelecer as conexões (coloridas em vermelho na Figura 51) com as portas de saída

que ainda estão desconectadas. As instâncias selecionadas são: **Sobrepor**$_2$ que é conectada a **Sobrepor**$_1$; e **Posicionar**$_2$ que é conectada a **Posicionar**$_1$. Na terceira repetição foram adicionadas: a instância **Posicionar**$_5$, conectada a **Posicionar**$_3$ e a **Construir Telhado**$_4$; e a instância **Construir Torre**$_2$ conectada a **Construir Torre**$_1$. Nesta repetição foi selecionada a instância **Posicionar**$_1$ que foi conectada a **Posicionar**$_4$. Na quarta repetição foi adicionada a instância **Posicionar**$_6$ que foi conectada a **Posicionar**$_5$ e a **Construir Torre**$_2$. Na última repetição, foi apenas selecionada a instância **Posicionar**$_4$ para conectar com a saída da instância **Posicionar**$_6$.

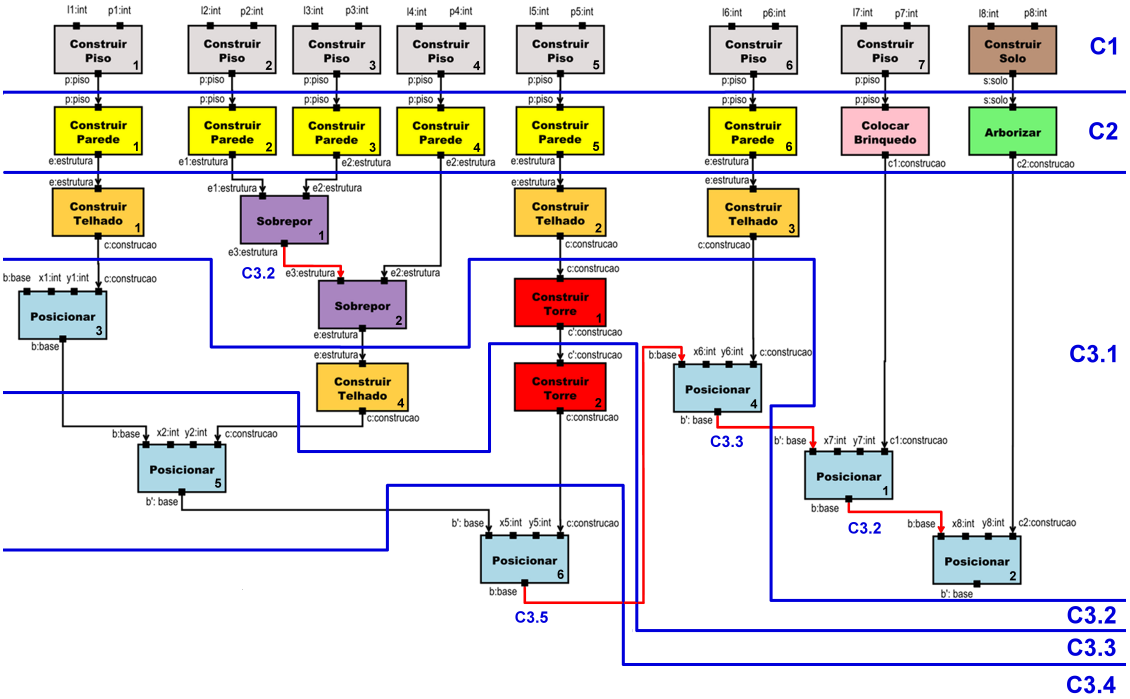


Figura 51 – Composição resultante das etapas C1, C2 e C3.

C4 Esta etapa não se aplica, pois todos os resultados foram alcançados na etapa anterior.

C5 Construiu-se o CD conforme a Figura 34.

C6 Definiu-se o CE, denominado **Construir Maquete**, esquecendo toda a estrutura interna do CD apresentado na Figura 52.

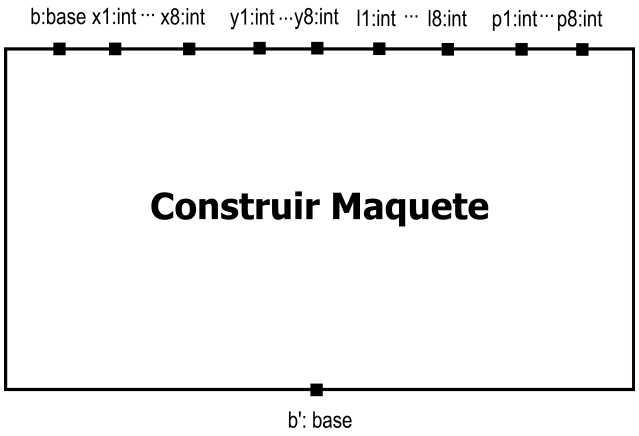


Figura 52 – CE **Construir Maquete**.

A.3 Solução do Estudo de Caso 3: Metodologia da Decomposição e Composição

A imagem é obtida a partir de cara (círculo maior), olhos (dois círculos paralelos), boca (um retângulo sobreposto a um círculo) e um laço (dois triângulos lado a lado). Para isso, tem-se as seguintes ações triviais: desenhar **retângulo** de qualquer cor e qualquer tamanho; desenhar **triângulo** equilátero (de lados iguais) de qualquer cor e qualquer tamanho de lado; desenhar **círculo** de qualquer cor e de qualquer raio; posicionar uma figura **ao lado** de outra figura e alinhar a borda superior das duas figuras; **deslocar** uma figura com relação a outra, fornecendo as coordenadas (a primeira figura fica no (0,0) e a segunda é deslocada sobre a primeira); e **girar** uma figura no sentido horário em qualquer ângulo.

D1 O problema de **Desenhar Imagem** foi dividido nos seguintes subproblemas menores (Figura 53), desenhar círculo amarelo, desenhar olhos, desenhar boca e desenhar laço.

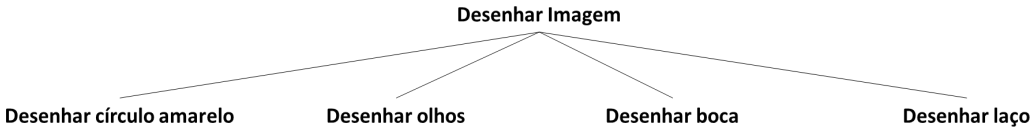


Figura 53 – Etapa D1: Árvore subproblemas menores.

D2 Nesta etapa, os subproblemas da etapa D1 foram divididos até que tivessem soluções triviais. A divisão pode ser observada na árvore ilustrada na Figura 54. As folhas em azul representam as ações triviais.

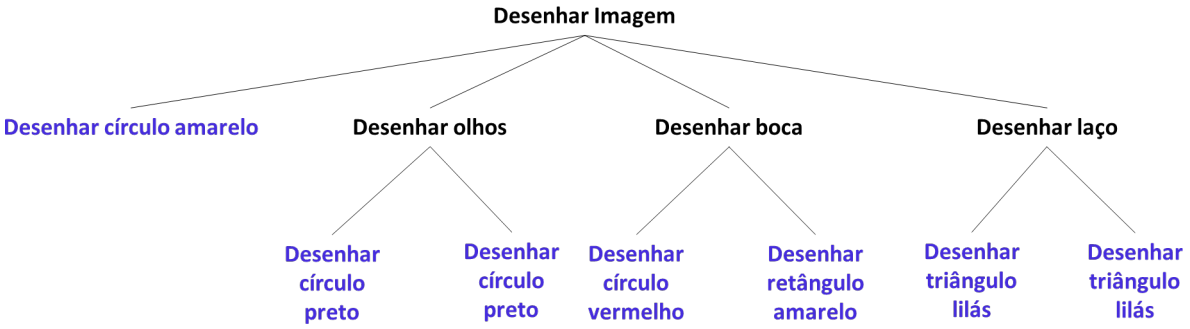


Figura 54 – Etapa D2: Árvore com subproblemas até soluções triviais.

D3 Foram definidos os CEs ilustrados na Figura 55, onde as entradas e saídas estão indicadas em cada componente. Note-se que alguns subproblemas compartilham soluções e que apenas um CE foi descrito para cada subproblema trivial distinto. Para o CE **Retângulo** foram definidas 3 entradas: a largura (*l*) e a altura (*a*), ambas do tipo numérico (*num*) e a cor *c* do tipo *string*, e uma saída *ret* do

tipo *imagem*. Para o CE **Círculo** foram definidas 2 entradas: raio (*r*) do tipo *num* (numérico) e cor (*c*) do tipo *string*; e uma saída *cir* do tipo *imagem*. Para o CE **Triângulo** foram definidas 2 entradas: lado (*l*) do tipo *num* (numérico) e cor (*c*) do tipo *string*; e uma saída *tri* do tipo *imagem*.

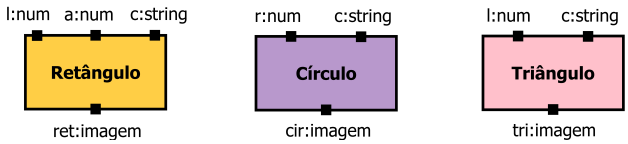


Figura 55 – Etapa D3: CEs para subproblemas triviais.

C1 Nesta etapa, adicionou-se uma instância de CE para cada instância de subproblema trivial definido na etapa D2. Foram adicionadas 4 instâncias de **Círculo**, 1 instância de **Retângulo** e 2 de instâncias de **Triângulo** (Figura 56).

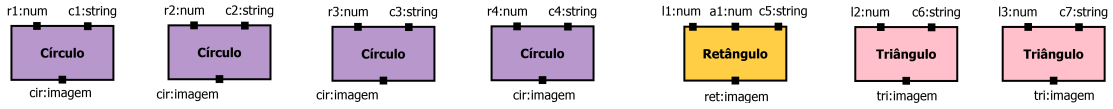


Figura 56 – Etapa C1: Instâncias de CEs para desenhar a Imagem.

C2 Esta etapa foi repetida 4 vezes.

Na primeira repetição identificou-se 3 menores subproblemas (Figura 57): *Desenhar olhos*, *Desenhar boca* e *Desenhar laço*, selecionando-se o primeiro para começar a resolver.

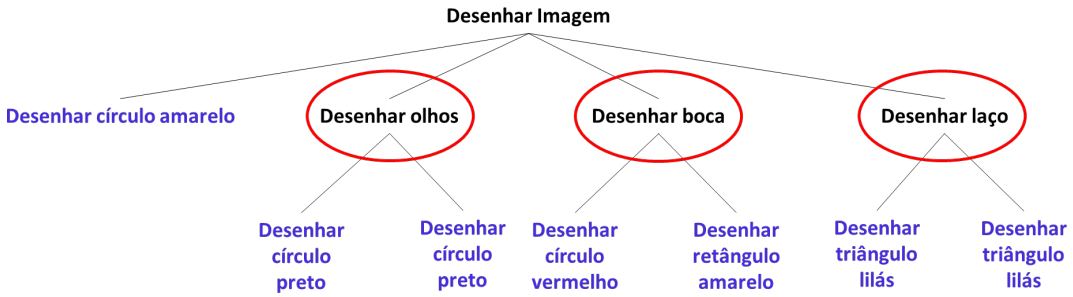


Figura 57 – Etapa C2: Primeira repetição, escolha de um subproblema menor.

Identificou-se a ação de combinação **Deslocar** e definiu-se um CE para esta ação. Para o CE **Deslocar** foram definidas 3 entradas: *f1* e *f2* do tipo *imagem* e *xy1* do tipo *coord* (um par de valores (x, y)); e uma saída *fig* do tipo *imagem*. Adicionou-se a primeira instância **Deslocar** estabelecendo as conexões. Conforme Figura 58.

Na segunda repetição selecionou-se o segundo menor subproblema *Desenhar boca*. Identificou-se a ação de combinação **Deslocar**. Como o CE para esta

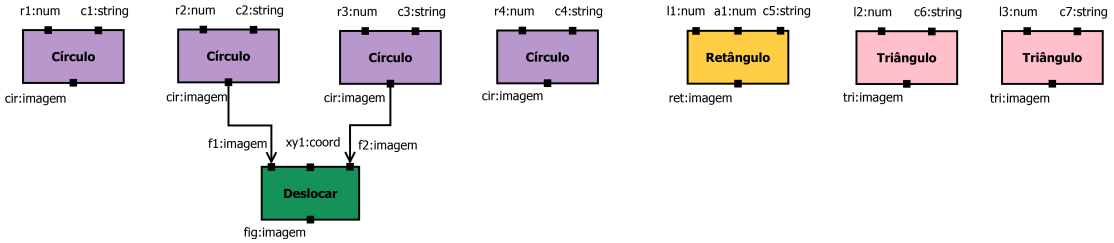


Figura 58 – Etapa C2: Primeira repetição.

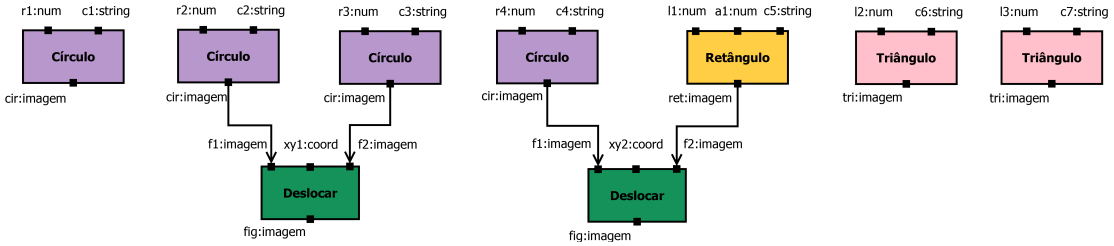


Figura 59 – Etapa C2: Segunda repetição.

ação já foi previamente definido, adicionou-se a segunda instância **Deslocar** e estabeleceu-se as conexões, conforme ilustrado na Figura 59.

Na terceira repetição, selecionou-se o terceiro menor subproblema *Desenhar laço*. Identificou-se 2 ações de combinação: **Girar** e **Ao lado**, pois antes de colocar os triângulos um ao lado do outro, precisa-se girá-los. Defini-se um CE para cada ação distinta: **Girar** e **Ao lado**, e adicionou-se 2 instâncias da ação de combinação **Girar**, uma para cada triângulo, e após uma para a ação **Ao lado**. Então, estabeleceu-se as conexões (Figura 60). Para o CE **Girar** foram definidas 2 entradas: *fig* do tipo *imagem* e ângulo (*ang*) do tipo *num*; e uma saída *fig* do tipo *imagem*. Para o CE **Ao lado** foram definidas 2 entradas: *esq* e *dir* do tipo *imagem* e uma saída *esq-dir* do tipo *imagem*.

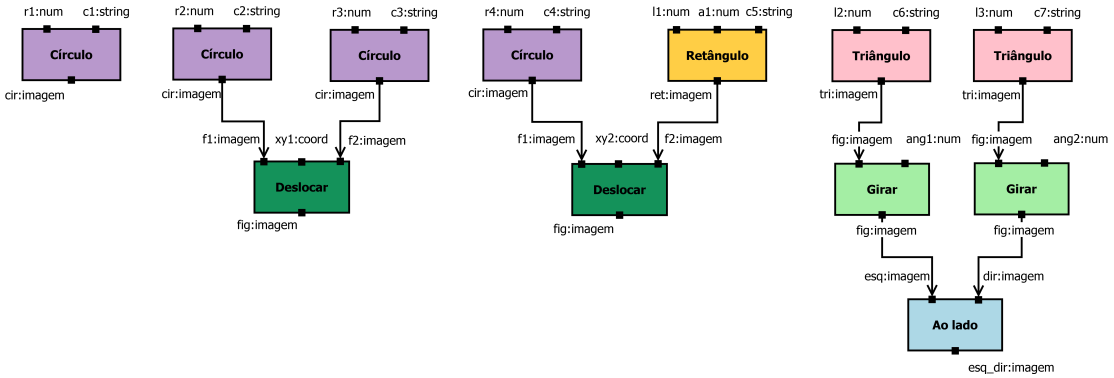


Figura 60 – Etapa C2: Terceira repetição.

Na quarta e última repetição, selecionou-se o problema de **Desenhar Imagem** (Figura 61).

Identificou-se a ação de combinação **Deslocar**, já foi definida, então não se criou

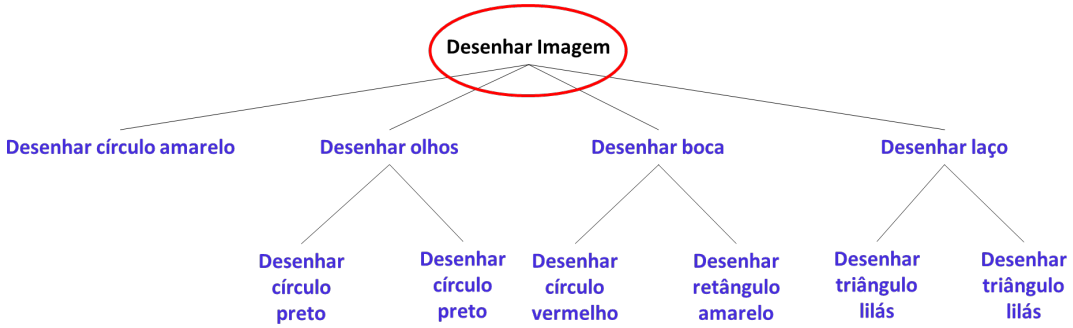


Figura 61 – Etapa C2: Quarta repetição, árvore.

um novo CE. Adicionou-se 3 instâncias **Deslocar** e estabeleceu-se as conexões, conforme Figura 62.

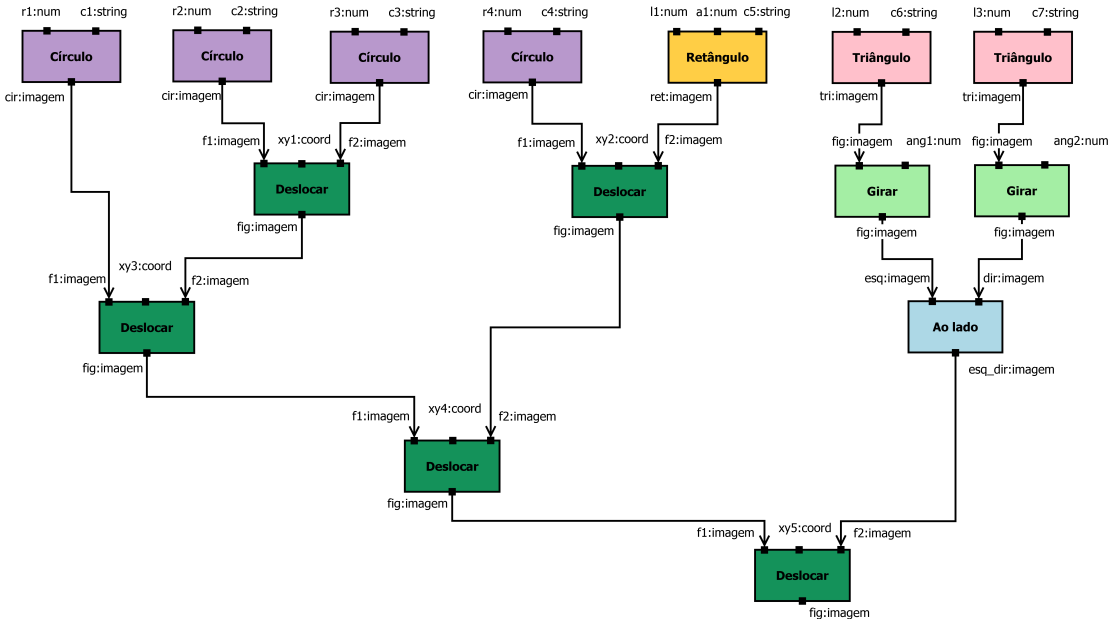


Figura 62 – Etapa C2: Quarta repetição.

Após a quarta repetição, verificou-se que o problema foi resolvido, isto é, foi construído o componente decomposto do problema **Desenhar Imagem**.

Nessas interações foram criados os CEs para as ações de combinação, conforme ilustrado na Figura 63.

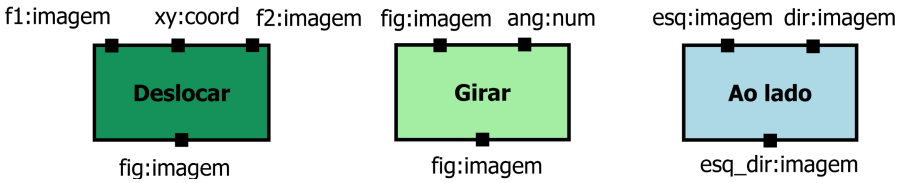
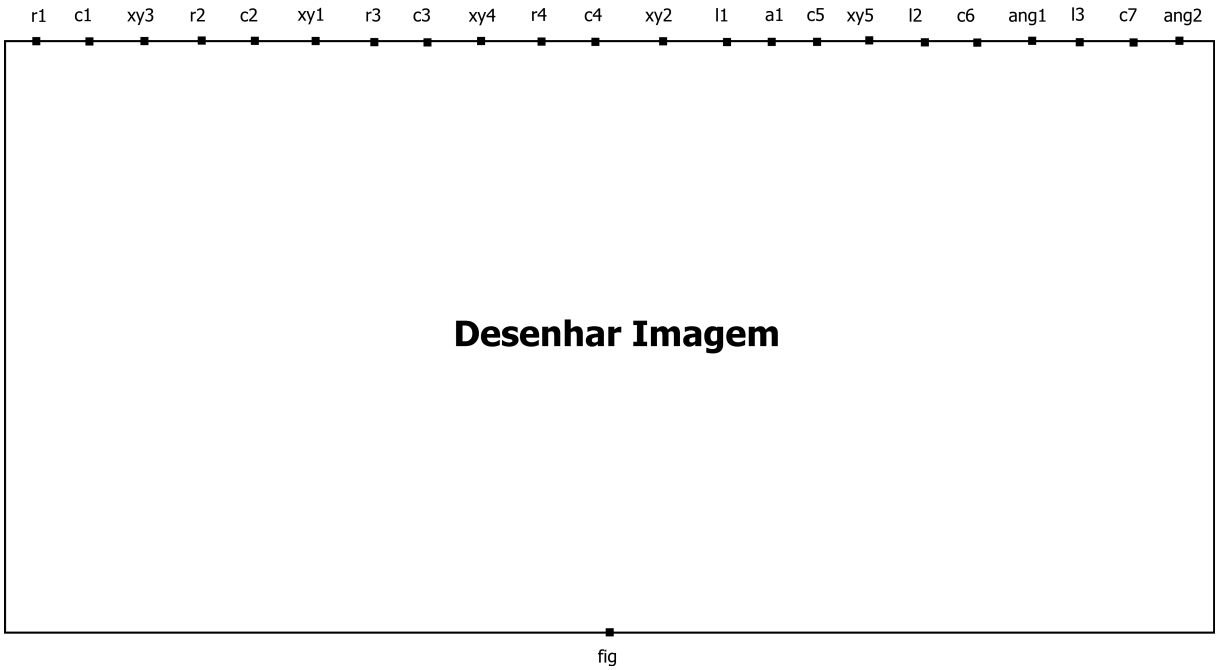


Figura 63 – Etapa C2: CEs para ações de combinação.

C3 Definiu-se o CE para associar ao CD definido na etapa anterior (Figura 64), denominado **Desenhar Imagem**, esquecendo-se de toda a estrutura interna do

CD. Esse CE têm as mesmas portas de entrada/saída do CD definido na etapa anterior.



Os tipos das portas do CE são dados por: $r1, r2, r3, r4, l1, l2, l3, a1, ang1, ang2 \in num$; $c1, c2, c3, c4, c5, c6, c7 \in string$; $xy1, xy2, xy3, xy4, xy5 \in coord$; $fig \in imagem$.

Figura 64 – Etapa C3: CE principal **Desenhar Imagem**.

APÊNDICE B – Questionário - Estudo de Caso da Metodologia da Composição

Pesquisa de Uso e Aplicação da Linguagem

1. A linguagem é simples e intuitiva de ser utilizada.
( ) Concordo Fortemente
( ) Concordo
( ) Neutro
( ) Discordo
( ) Discordo Fortemente

2. A metodologia auxiliou no processo de solução do problema.
( ) Concordo Fortemente
( ) Concordo
( ) Neutro
( ) Discordo
( ) Discordo Fortemente

3. Eu sabia o que deveria ser feito a cada passo de aplicação da metodologia.
( ) Concordo Fortemente
( ) Concordo
( ) Neutro
( ) Discordo
( ) Discordo Fortemente

4. Consegui identificar facilmente os tipos de entradas e saídas dos componentes.
( ) Concordo Fortemente
( ) Concordo
( ) Neutro
( ) Discordo
( ) Discordo Fortemente

5. Consegui identificar facilmente os componentes independentes.
( ) Concordo Fortemente

( ) Concordo
( ) Neutro
( ) Discordo
( ) Discordo Fortemente

6. Consegui identificar facilmente dependência entre componentes.
( ) Concordo Fortemente
( ) Concordo
( ) Neutro
( ) Discordo
( ) Discordo Fortemente

7. Consegui identificar facilmente as conexões entre componentes.
( ) Concordo Fortemente
( ) Concordo
( ) Neutro
( ) Discordo
( ) Discordo Fortemente

8. Liste (se achar que há), até 3 vantagens de uso da linguagem.
9. Liste (se achar que há), até 3 desvantagens de uso da linguagem.
10. Deixe seus comentários sobre a atividade.