



Pós-Graduação em Ciência da Computação

**“Um método de análise de problemas
multitarefas concorrentes: uma aplicação em
jogos RTS”**

Por

Fernando Rocha

Tese de Doutorado



Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, MARÇO/2015



Universidade Federal de Pernambuco

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Fernando Antônio Farias Rocha

***“UM MÉTODO DE ANÁLISE DE PROBLEMAS
MULTITAREFAS CONCORRENTES: UMA
APLICAÇÃO EM JOGOS RTS”***

Este trabalho foi apresentado à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestra em Ciência da Computação.

ORIENTADOR(A): Prof Geber Lisboa Ramalho

CO-ORIENTADOR(A): Profa Patrícia Cabral de Azevedo Rastelli Tedesco

RECIFE, (MARÇO/2015)

Catálogo na fonte
Bibliotecária Jane Souto Maior, CRB4-571

R672m Rocha, Fernando Antônio Farias

Um método de análise de problemas multitarefas concorrentes:
uma aplicação em jogos RTS. – Recife: O Autor, 2015.

94 f.: il., fig., tab.

Orientador: Geber Lisboa Ramalho.

Tese (Doutorado) – Universidade Federal de Pernambuco.
Cln, Ciência da computação, 2015.

Inclui referências.

1. Inteligência artificial. 2. Engenharia de software. 3.
Sistemas multiagentes. I. Ramalho, Geber Lisboa (orientador). II.
Título.

006.3

CDD (23. ed.)

UFPE- MEI 2015-168

Tese de Doutorado apresentada por **Fernando Antonio Farias Rocha** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título “**UM MÉTODO DE ANÁLISE DE PROBLEMAS MULTITAREFAS: UMA APLICAÇÃO EM JOGOS RTS**” orientada pelo **Prof. Geber Lisboa Ramalho** e aprovada pela Banca Examinadora formada pelos professores:

Prof. André Luis de Medeiros Santos
Centro de Informática/UFPE

Profa. Carla Taciana Lima Lourenco Silva Schuenemann
Centro de Informática / UFPE

Prof. Sergio Ricardo de Melo Queiroz
Centro de Informática / UFPE

Prof. Charles Andryê Galvão Madeira
Instituto Metrópole Digital / UFRN

Prof. Andre Mauricio Cunha Campos
Departamento de Informática e Matemática Aplicada / UFRN

Visto e permitida a impressão.
Recife, 13 de março de 2015.

Profa. Edna Natividade da Silva Barros

Coordenadora da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.

Agradecimentos

Foram cinco longos e intermináveis anos ao longo deste doutorado. Muitas vezes pensei que não conseguiria chegar no fim, ou que nem fim existiria (a segunda afirmação é verdadeira, um trabalho de pesquisa, de fato, nunca se encerra). Em conjunto com estes cinco anos, veio o amadurecimento e diversas outras responsabilidades e problemas, principalmente problemas... Problemas que só foram possíveis de serem superados com a presença de algumas pessoas e que de fato precisam ser agradecidas.

Não tenho como representar em palavras o quanto eu sou grato aos meus orientadores Geber Ramalho e Patrícia Tedesco. Neste período de trabalho foram algumas madrugadas de trabalho, discussões intermináveis e conversas motivadoras, que me moldaram e muito meu amadurecimento durante este período. Se hoje eu tenho esta tese escrita, muito é devido ao incentivos deles.

Também é necessário agradecer aos professores que estiveram presentes na banca, por terem aceitado o convite, disponibilizado um tempo para ler e avaliar o trabalho e ainda terem mudado suas rotinas para poderem participar da minha avaliação. Obrigado a cada comentário e discussão realizada durante a defesa.

Um agradecimento mais do que especial a minha esposa, Tássia Brandão, foi mais um passo dado que fizemos juntos. Devo e muito a você o poder de argumentação e de hoje ter uma visão maior do mundo, você me mostrou que a vida é muito mais ampla do que eu podia imaginar. Obrigado por ter me dado forças em cada um dos momentos complicados que passamos, você me permitiu conseguir superar cada um destes momentos.

Obrigado!

*So you run and you run to catch up with the sun but it's sinking
Racing around to come up behind you again.
The sun is the same in a relative way but you're older,
Shorter of breath and one day closer to death.*

—PINK FLOYD (Time)

Resumo

O desenvolvimento de soluções de Inteligência Artificial (IA) para sistemas computacionais é complexo dado a natureza dos problemas atacados, em particular quando envolvem problemas multiagentes e multitarefas (MAMT). Apesar de existirem vários métodos para o desenvolvimento de Sistema Multiagentes (SMA), são poucos os que dão alguma importância à compreensão do problema; e mesmo estes métodos não abordam os problemas MAMT com o devido detalhamento. Abordando a deficiência destas metodologias, estamos propondo o método Icelus que foca em guiar o analista em compreender e descrever corretamente o problema a ser solucionado. Icelus permitirá uma melhor abordagem na análise e compreensão de um problema MAMT, facilitando a distribuição do conhecimento para o restante do time de desenvolvimento, reduzindo o risco de erros de codificação ao longo do desenvolvimento do projeto.

Palavras-chave: Agentes Inteligentes, Sistemas Multi-Agentes, Método de Análise de Problemas

Abstract

The development of Artificial Intelligence (AI) to computational systems is a complex activity, given the nature of the problems attacked, in particular when they involve multi-agent problems and multitasking (MAMT). Although there are several methods for the development of Multi-agent System (MAS), there are just a few that give any importance to understanding the problem; and even these methods do not address the problems with all detailing that MAMT problems needs. Addressing the deficiency of these methods, we are proposing the Icelus method that focuses on leading the analyst to understand and describe correctly the problem to be solved. Icelus will enable a better approach in the analysis and understanding of a MAMT problem, facilitating the distribution of knowledge to the rest of the development team, reducing the risk of coding errors throughout the development of the project.

Keywords:

Intelligent Agents, Multi-Agent Systems, Problem Analysis Method

Lista de Figuras

2.1	Processo de desenvolvimento de uma solução de engenharia.	20
2.2	Processo de desenvolvimento de uma solução exploratória, muito utilizado na concepção de soluções que necessitam de IA.	21
3.1	Processo de análise e modelagem de um problema.	30
3.2	Exemplo de uma árvore de problemas, onde os nós superiores são as causas e os nós inferiores são as consequências de um determinado subproblema.	33
3.3	Fases e disciplinas de RUP. [Fonte: www.ibm.com/].	36
3.4	Modelo BPM para o problema de transações bancárias em um caixa eletrônico.	38
3.5	Modelo de inferência para o problema de avaliação utilizados por KADS.	43
3.6	Exemplo da estrutura resultante da etapa de análise de gaia	46
3.7	Diagrama hierárquico de metas. (Retirado e modificado a partir de (DELOACH, 2001))	47
3.8	Exemplo da estrutura de <i>Early Requirements</i> de um problema modelado com Tropos	48
5.1	Modelo de problemas identificados em Starcraft a partir do primeiro experimento utilizando Icelus.	65
5.2	Modelo de problemas identificados em Starcraft a partir do primeiro experimento utilizando Icelus (continuação).	66
5.3	Processo iterativo e incremental em que Icelus foi submetido durante sua evolução.	67
5.4	Processo de aplicação desta segunda versão de Icelus.	69
5.5	Perfil de jogador dos alunos que participaram do experimento.	69
5.6	Opiniões dos participantes do experimento a respeito da utilização de Icelus.	71
6.1	Processo de utilização da terceira versão de Icelus.	73
6.2	Exemplo de uma árvore de problemas de Starcraft, utilizando o subproblema da produção de uma unidade de infantaria como raiz.	75

Lista de Tabelas

5.1	Comparativo entre as características de Icelus em suas diferentes versões. . . .	63
5.2	Exemplo de planilha da aplicação da primeira versão de Icelus.	63
5.3	Resultados obtidos da avaliação das seis equipes que compuseram o grupo de participantes que utilizaram o método Icelus na atividade de descrição da IA de Starcraft.	70
5.4	Resultados obtidos da avaliação das seis equipes que compuseram o grupo de participantes que não tiveram contato com o método Icelus, realizando a atividade de descrição da IA de Starcraft de forma ad hoc.	70
7.1	Quantidade de subproblemas relevantes identificados durante o experimento pelos grupos Icelus.	83
7.2	Quantidade de subproblemas relevantes identificados durante o experimento pelos grupos com uma abordagem Ad hoc.	83
7.3	Resultados obtidos da avaliação das quatro equipes que compuseram o grupo de participantes que utilizaram a terceira versão do método Icelus na atividade de descrição da IA de Starcraft.	84
7.4	Resultados obtidos da avaliação das quatro equipes que compuseram o grupo de participantes que utilizaram uma abordagem <i>Ad hoc</i> na atividade de descrição da IA de Starcraft.	84
7.5	Valores de $ t_0 $ para a avaliação das hipóteses nulas do experimento.	84

Lista de Acrônimos

ADP	Agent Development Process	49
BDD	Behavior Driven Development - Desenvolvimento baseado em comportamento	50
BPM	Business Process Management	37
BPMN	Business Process Modeling Notation	37
BWAPI	BroodwarAPI	16
EC	Engenharia de Conhecimento	34
ES	Engenharia de Software	19
IA	Inteligência Artificial	13
KBS	Knowledge Based System - Sistema baseado em conhecimento	41
MA	Multiagentes	24
MAMT	Multitarefas e Multiagentes	14
MaSE	Multi-Agent System Engineering	46
MT	Multitarefas	23
NPCs	Non-Player Characters	13
PO	Pesquisa Operacional	34
RTS	Real-Time Strategy	14
SI	Sistemas Inteligentes	18
SMA	Sistema Multiagentes	13
TDD	Test Driven Development - Desenvolvimento baseado em testes	36
TDI	Test-Driven Implementation	49
XP	Extreming Programming	36

Sumário

1	Introdução	13
1.1	Objetivos	17
1.2	Organização	18
2	Um Olhar Crítico sobre a Pesquisa em SMA	19
2.1	O Problema na IA	19
2.2	A Existência de Problemas Complicados na IA	23
2.3	Pontos Importantes na Compreensão de um Problema de IA	26
2.4	Conclusão	28
3	A compreensão de problemas em diversos campos do conhecimento	29
3.1	Campo conceitual	29
3.1.1	Análise	31
3.1.2	Modelagem	33
3.2	Em outros campos da pesquisa	34
3.2.1	O Campo da Engenharia de Software	34
3.2.2	O Campo da Pesquisa Operacional	38
3.2.3	Em Engenharia do Conhecimento	41
3.3	O campo dos SMA	44
3.3.1	Gaia	45
3.3.2	MaSE	46
3.3.3	Tropos	47
3.3.4	SADAAM	49
3.3.5	Beast	49
3.3.6	MAS-CommonKADS	50
3.3.7	Prometheus	51
3.4	Por que algo novo	52
3.5	Conclusão	52
4	A Concepção de Icelus	54
4.1	Visão Geral	54
4.2	Motivação	55
4.3	Icelus	57
4.3.1	Pilares	57
4.3.1.1	Métodos de expansão de problemas	58
4.3.2	Método	59

4.4	Conclusão	60
5	A Evolução de Icelus	61
5.1	Refinamento	61
5.2	Versões	62
5.2.1	Icelus v1	62
5.2.2	Icelus v2	65
5.3	Conclusão	71
6	Icelus	72
6.1	Coletar dados do ambiente	73
6.1.1	Entrega	74
6.2	Gerar árvore de problemas	74
6.2.1	Entrega	75
6.3	Selecionar tarefas	76
6.3.1	Entrega	76
6.4	Descrever tarefas	76
6.4.1	Entrega	77
6.5	Conclusão	77
7	Icelus - Resultados	78
7.1	Experimento	78
7.1.1	Definição das metas	79
7.1.2	Planejamento do experimento	79
7.1.3	Design do Experimento	80
7.1.4	Validação	81
7.2	Análise dos Resultados	82
7.3	Execução	83
7.3.1	Dados Resultantes	83
7.3.2	Análise Estatística	84
7.4	Conclusão	85
8	Conclusão	86
	Referências	89

1

Introdução

Cada vez mais os jogos eletrônicos tem exigido comportamentos mais inteligentes, propondo desafios cada vez mais realistas para o jogador (DANIELSIEK et al., 2008). Muito deste comportamento realista e mais inteligente dos personagens não controlados pelo jogador - *Non-Player Characters (NPCs)* - se dá devido a inserção de técnicas de Inteligência Artificial (IA) cada vez mais elaboradas e complexas. No início, estas técnicas, eram limitadas à baixa quantidade de memória e poder de processamento dos computadores e consoles, porém estas barreiras estão cada vez menos presentes hoje em dia (LUCAS et al., 2013). Muito desta complexidade envolvida ocorre devido à simulação de tarefas de natureza complexa (como planejamento, aprendizado, predição, etc.), bem como algumas características do ambiente que tornam sua compreensão ainda mais difícil: o ambiente pode ser parcialmente observável, não-episódico, não-determinístico, etc. (BURO; FURTAK, 2004). Outro fator importante e que aumenta a dificuldade em propor uma solução é a atividade de transportar as informações do problema, presentes no mundo real, para o ambiente computacional, onde a solução estará modelada (MCLEOD; MACDONELL, 2011).

A IA é empregada em algumas situações dos jogos, como por exemplo tutoriais adaptativos (SOTTILARE; GILBERT, 2011), desenvolvimento de enredo adaptativo (EL-NASR, 2007) e o controle dos NPCs (LUCAS et al., 2013). Focando no controle dos NPCs, muitos destes mecanismos de IA são desenvolvidos baseando-se em Sistema Multiagentes (SMAs). A principal razão é pela possível correspondência entre as entidades não controladas pelo jogador e agentes. Os NPCs precisam apresentar um comportamento mais dinâmico e autônomo, possuindo a capacidade de perceber modificações no ambiente e gerir suas próprias reações de acordo com a necessidade em alcançar um objetivo (RUSSELL; NORVIG, 2003). Desta forma, a IA para controle dos NPCs em jogos será, geralmente, formada por um SMA envolvendo diversos agentes que realizarão tarefas variadas (e.g. planejamento, *pathfinding*, exploração, predição, tomada de decisão, etc.). Esta multiplicidade de tarefas e a presença de diversos agentes irão elevar significativamente a dificuldade de identificar como será estruturada a IA do jogo e como esta será desenvolvida.

Estes tipos de problemas não são uma particularidade presente apenas no desenvolvi-

mento da IA para jogos. É verdade que a IA de certos jogos (como os de estratégia) é difícil de ser desenvolvida. Porém, várias outras aplicações de IA podem demandar uma solução Multiagente que seja difícil de ser concebida e desenvolvida. Um exemplo disto é um sistema de gestão municipal, onde este sistema irá guiar a máquina pública em ações que podem ser realizadas para aprimorar onde o dinheiro será empregado, aumentando a gestão da documentação existente, minimizando possíveis fraudes neste sistema de gestão pública. Nestes casos, é possível identificar muitas variáveis, muitas tarefas e muitos pontos de vista a serem considerados, que consequentemente aumentam drasticamente a dificuldade em compreender o problema e identificar uma solução possível. Desta forma, estes seriam problemas que para serem solucionados necessitarão envolver muitos agentes inteligentes que precisarão realizar muitas tarefas em busca de um objetivo em comum. Estes problemas passarão a ser denominados ao longo desta tese como problemas Multitarefas e Multiagentes (MAMT).

Quanto maior a quantidade de agentes e tarefas envolvidas em um problema, mais complexa será a sua compreensão. Os problemas MAMT naturalmente apresentarão uma grande dificuldade em serem modelados e solucionados. As principais dificuldades existentes são:

- A interdependência das tarefas existentes, onde a realização de uma destas tarefas poderá implicar em diversas consequências em outras;
- A necessidade de coordenar e organizar os agentes existentes ([WEISS, 1999](#))([WOOLDRIDGE; CIANCARINI, 2001](#));
- A quantidade de informações envolvidas no problema, resultando em simplificações deste problema, que ocorrem devido às limitações da capacidade cognitiva do cérebro humano ([HALFORD et al., 2005](#)).

Em alguns gêneros de jogos é possível encontrar mais problemas MAMT do que em outros. Isto se verifica, por exemplo, nos jogos de estratégia em tempo real (*Real-Time Strategy* (RTS)) que envolvem inúmeros agentes realizando diversas tarefas em tempo real. O subgênero mais comum dos RTS é um simulador de guerra e gerenciamento tático/estratégico, onde o jogador comanda um exército tomando decisões e realizando ações como coleta e gerenciamento de recursos, construção de edificações que irão compor sua base militar, desenvolvimento tecnológico e controle de alto nível das unidades existentes. Um exemplo de jogo RTS é o jogo Starcraft ([STARCRAFT, 1998](#)) da produtora Blizzard, que se passa em um futuro, ano de 2229, e envolve a luta pela sobrevivência entre três raças: *Terran*, *Protoss* e *Zerg*.

Outros sistemas de software que também sejam estruturados como um SMA e que devam solucionar diversas tarefas podem ser associados como um problema MAMT e que, consequentemente, são difíceis de serem tratados. Esta dificuldade já é apresentada na concepção do projeto, antes mesmo de se pensar em uma solução técnica específica de IA. Esta primeira etapa de compreensão do problema é responsável pela captação das informações do que de fato seria o problema a ser solucionado e consequentemente um dos mais difíceis ao longo de todo o

ciclo de desenvolvimento do projeto, pois não é possível termos uma "solução correta" de um "problema errado".

Desta forma, esta tese procura afirmar que antes do desenvolvedor discutir a solução de IA que será empregada no controle dos NPCs para um jogo RTS, é primordial fazer uma melhor avaliação do que realmente é o problema. Esta definição prévia irá aumentar a compreensão do problema, reduzindo erros durante a codificação da solução devido à possíveis escolhas realizadas na definição desta solução. Mais precisamente, esta compreensão de problemas consiste em duas etapas: análise e modelagem. A primeira está definida na quebra do problema principal em subproblemas menores e mais simples de serem compreendidos (BLACKBURN, 2008), enquanto a segunda consiste na escolha dos subproblemas relevantes e a construção de uma representação em um modelo (YU; MYLOPOULOS, 1994).

Esta proposta de compreensão de problemas é baseada nas práticas já consolidadas existentes em outras áreas da Computação, como Engenharia de Software, Engenharia do Conhecimento e Pesquisa Operacional. Estas áreas já despendem um tempo significativo do projeto na etapa inicial de análise do problema permitindo, respectivamente, uma melhor compreensão do problema e proposta da solução (SOMMERVILLE, 1995), o reuso da especificação em outros problemas similares (WIELINGA; SCHREIBER; BREUKER, 1992) e a busca por uma solução ótima (HILLIER; LIEBERMAN, 2001).

Apesar de parecer ser óbvia a ideia de despendar tempo para analisar um problema antes de tentar resolvê-lo, dos vários métodos para desenvolvimento de SMA (BRESCIANI et al., 2004; WOOLDRIDGE; JENNINGS; KINNY, 2000; PADGHAM; WINIKOFF, 2002; CLYNCH; COLLIER, 2007; CARRERA BARROSO; SOLITARIO; IGLESIAS FERNANDEZ, ???), poucas dão importância para as etapas iniciais do desenvolvimento. Mesmo sendo reconhecida como uma etapa que pode gerar diversos erros durante o restante do projeto (PRESSMAN, 2001), estes métodos deixam a cargo do time de desenvolvimento realizar a análise e compreensão do problema de forma *ad hoc*. Enquanto na literatura da Engenharia de Software, por exemplo, é atribuída uma grande importância às etapas iniciais, indicando inclusive que uma boa especificação do problema e posteriormente uma boa especificação da solução antes de codificá-la irá evitar erros durante esta fase de desenvolvimento (PRESSMAN, 2001); este não é, infelizmente, o estado da arte no desenvolvimento de SMA, mesmo quando o problema a ser tratado é complicado, como é o caso dos MAMT.

Entre os métodos para SMA abordados ao longo deste documento, Tropos (BRESCIANI et al., 2004) é o único que demonstra algum cuidado com a etapa inicial de entendimento do problema. Entretanto, a fase de "análise de problema", denominada em Tropos de "*Early requirements*", é dedicada apenas à identificação de dados relevantes no ambiente, como atores, suas metas e suas dependências. A fase seguinte de Tropos, "*Late requirements*", já é uma fase dedicada à identificação das mudanças ocorridas no domínio quando o sistema é inserido como um novo ator, como também estabelecer as dependências entre o sistema e os outros atores identificados previamente. Desta forma, existe um intervalo considerável entre estas duas fases

de Tropos que o próprio método deixa a cargo do time de desenvolvimento solucionar. Em outras palavras, Tropos deixa a desejar na questão da orientação do desenvolvedor de SMA na fase de compreensão do problema, deixando uma grande lacuna a ser preenchida e de forma ad hoc.

Analizando esta dificuldade em compreender problemas MAMT e o fato dos métodos para SMA não darem a devida atenção às etapas iniciais, este trabalho propõe o método Icelus¹, que surgiu a partir da observação da dificuldade apresentada pelos alunos em propor e desenvolver o projeto de fim de semestre de uma disciplina de Agentes Inteligentes do curso de Ciência da Computação na Universidade Federal de Pernambuco. Este projeto consiste em propor e posteriormente codificar um SMA para controlar os exércitos no jogo Starcraft Broodwar (STARCRAFT, 1998), jogo classificado como RTS, através da interface de programação BroodwarAPI (BWAPI). Porém, um sistema de controle para um jogo RTS é tipicamente um problema MAMT, onde é possível encontrar inúmeras atividades, como coleta de recursos, defesa do centro de comando, ataque ao centro de comando adversário, exploração do mapa, ataque às unidades adversárias, entre outras; e que no geral são divididas por diversos agentes que deverão estar trabalhando em conjunto em busca do objetivo comum que é derrotar o adversário. Além desses critérios básicos de um problema MAMT, um jogo RTS ainda possui uma característica que torna este problema ainda mais especial, tudo ocorre em tempo real modificando constantemente o ambiente e sendo necessário uma verificação constante de como o problema será solucionado.

Devido a estas complexidades, muitos alunos apresentavam uma grande dificuldade inicial na concepção do projeto, não conseguindo lidar com a quantidade de tarefas existentes nem muitas vezes montar uma linha de raciocínio de uma possível solução. De fato, os jogos RTS apresentam diversas tarefas e muitas delas necessitam de uma soluções baseada em técnicas de IA dificultando a identificação e coordenação destas tarefas; tarefa como: predição (LAIRD, 2001; WEBER; MATEAS, 2009), planejamento (PÉREZ, 2011), gestão de recursos (CHURCHILL; BURO, 2011), reconhecimento de padrão (SHARIFI; ZHAO; SZAFRON, 2010), etc.

Desta forma, Icelus procura:

- Guiar o time de desenvolvimento na análise de um problema MAMT existente em jogos RTS, de forma a captar informações relevantes na construção da solução de IA para este jogo;
- Gerar uma documentação capaz de servir como primeira validação do sistema a ser desenvolvido;
- Permitir sua utilização de forma mais ágil, utilizando o método apenas em partes da IA para o jogo RTS, partes que exijam um conhecimento mais aprofundado.

¹Icelus (Ikelos) na mitologia Grega é um dos Oneiros, filho de Hypnos e irmão de Morpheus e Phantasos. Um Oneiro é a personificação do ato de sonhar, sendo Icelus a personificação dos pesadelos (EVSLIN, 2006). De forma similar, o método de nome homônimo, procura representar o problema que será abordado, gerando sua representação e tornando-o mais claro, impedindo que este se torne um "pesadelo" para a equipe de desenvolvimento.

Com este intuito, o método Icelus foi desenvolvido e refinado nos últimos quatro anos, tendo sido aplicado por dezenas de estudantes de graduação em seu último ano de curso e também por alunos da pós-graduação (mestrado e doutorado) em Ciência da Computação.

Icelus tem sido desenvolvido com foco principal na fase de análise, em que foram estruturadas duas ferramentas para melhoria dos resultados. A primeira consiste na utilização de uma árvore de problemas (SNOWDON; SCHULTZ; SWINBURN, 2008) permitindo identificar mais facilmente subproblemas relevantes a partir da identificação das causas e consequências de um subproblema raiz. Como por exemplo, o caso da coleta de impostos em um sistema de gestão pública. Utilizando este subproblema como raiz de uma árvore de problemas utilizada por Icelus, sua consequência seria a entrada de recursos nos cofres públicos, gerando um novo subproblema de como realizar e o gasto deste recurso, enquanto como causa da coleta de imposto seria o aumento das transações financeiras realizadas pelo comércio existente no município. Enquanto a segunda ferramenta consiste em um formulário que guiará o time de desenvolvimento na coleta das informações que sejam relevantes para estes subproblemas, dados relacionados exclusivamente ao problema, mas que serão utilizados quando for conceber a solução.

Algumas versões do Icelus foram propostas. Cada uma passou por um processo de validação. Todas as versões utilizaram o problema de especificação da IA do jogo de Starcraft como estudo de caso. Após cada uma destas validações, os comentários obtidos pelos avaliadores e pelos participantes do experimento foram considerados em um processo de revisão da literatura, permitindo assim identificar possíveis alterações no método a partir de técnicas já utilizadas em outras áreas da Computação. Desta forma, estas etapas de melhoria do método Icelus podem ser consideradas como um processo iterativo e incremental.

1.1 Objetivos

Esta tese tem como principal objetivo a especificação do método Icelus capaz de guiar um time de desenvolvimento durante a fase inicial de um projeto de software que definirá a IA para o controle dos NPCs de um jogo RTS, permitindo uma melhor compreensão deste problema. Para isto, uma análise das metodologias existentes para SMA foi realizada, onde foi comprovada que estas metodologias não dão a correta importância a esta fase inicial de compreensão do problema. Desta forma, estes métodos deixam a cargo da equipe de desenvolvimento procurar compreender o problema de forma ad hoc. Como objetivos específicos é possível citar:

- a análise das metodologias de SMA existentes e como estas lidam com o desenvolvimento destes sistemas, principalmente a fase inicial onde é necessário compreender o problema a ser solucionado;
- a análise de como outras áreas da Computação abordam a análise de problemas, identificando quais seriam os benefícios da realização da análise de um problema;

- pontuar a distinção entre os problemas da Engenharia de Software e a da Inteligência Artificial, mostrando que estes dois campos não podem tentar abordar um problema de forma similar;
- identificar a importância da análise de problemas para o campo dos Sistemas Inteligentes (SI), uma abordagem que não é tão realizada no dia a dia do desenvolvimento de SI.

1.2 Organização

O restante desta tese está organizado da seguinte forma. Primeiramente, o segundo capítulo procura abordar as diferenças entre os problemas existentes na Engenharia de Software e os que necessitam utilizar alguma solução inteligente, identificando as suas diferenças e a impossibilidade da utilização direta de técnicas da Engenharia de Software para estes outros problemas. O capítulo seguinte irá abordar como outros campos lidam com a compreensão de problemas. Os próximos quatro capítulos irão de fato apresentar o método aqui proposto, iniciando com a concepção do método, onde no quarto capítulo será apresentada a concepção de Icelus, como este método surgiu e qual o problema que estará sendo atacado. Dando sequência com a apresentação das versões do método e como este evoluiu ao longo do tempo de pesquisa. Onde no sexto capítulo, será apresentada a versão atual de Icelus. Encerrando a apresentação do método com um experimento e os resultados obtidos. Finalizando a tese com a conclusão do trabalho e possíveis trabalhos futuros dando continuidade ao projeto.

2

Um Olhar Crítico sobre a Pesquisa em SMA

Este capítulo ambientará o leitor com a diferença entre os problemas existentes no mundo da Engenharia de Software (ES) e os problemas que necessitam de soluções com Inteligência Artificial (IA). Esta distinção se faz necessária para poder compreender que uma abordagem realizada com um método puramente proveniente da ES para a análise de problemas de IA não é possível por se tratar de dois campos que necessitam lidar com informações diferentes, sendo assim necessário um método específico para cada um.

2.1 O Problema na IA

É sabido que a IA é uma ciência ainda muito nova, surgida oficialmente em 1950 com um trabalho de Turing ([TURING, 1950](#)). O que esta pouca idade resulta é em um campo de pesquisa ainda rondado por diversas incertezas, inclusive sobre o que de fato é IA ([RUSSELL; NORVIG, 2003](#)). Muitas expectativas surgiram já na origem da IA, do que poderia ser feito em alguns anos e o quanto estaríamos melhor após o fortalecimento deste campo de pesquisa. No entanto, muitas destas expectativas não foram atendidas, muito se deu ao fato destas expectativas serem muito elevadas e muitas vezes irreais ([SOMMERVILLE, 1993](#)). Mas mesmo assim, muitos campos se beneficiaram com o avanço e fortalecimento das pesquisas em IA, mostrando que é um campo ainda em amplo desenvolvimento.

Em contrapartida, a Engenharia de Software procura adaptar técnicas do campo da engenharia, conhecidas a séculos, ao desenvolvimento de software. Com a aplicação destas técnicas de engenharia ao desenvolvimento de software, passou a ser possível ter um maior controle deste processo de desenvolvimento, minimizando riscos e principalmente custos, se tornando uma saída concreta para a crise de software ocorrida nos anos 80 ([HUMPHREY, 1989](#)).

Apesar dos benefícios da IA para outros campos, ela não pode e nem deve ser vista como um processo normal de engenharia, devido a diferenças profundas entre as naturezas dos problemas destes dois campos. Um problema de engenharia (não necessariamente Engenharia de Software) envolve itens que possuem seu comportamento muito mais conhecido e previsível. Isto por possuir componentes mecânicos que transferem ou transformam energia de uma parte do

sistema para outro, ou por possuir componentes elétricos que transferem ou transformam sinais, ou seja, componentes de software que controlam estes componentes elétricos ou mecânicos e que especificam a troca de informações dentro do sistema. Estes três componentes apresentam um comportamento conhecido o que facilita o desenvolvimento do sistema, resultando inclusive, que um problema de Engenharia seja um problema que pode ser quase completamente compreendido e analisado antes de se realizar a construção de sua solução, seguindo muitas vezes um processo dividido em três etapas: (i) especificação do sistema; (ii) construção; e (iii) validação do sistema; como pode ser visualizado na figura 2.1.

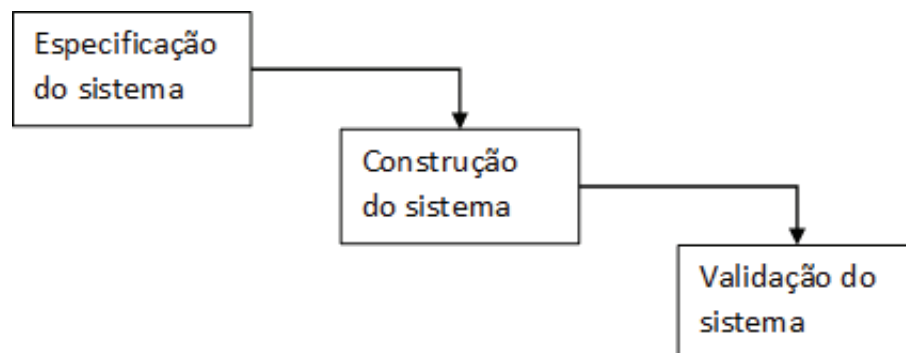


Figura 2.1: Processo de desenvolvimento de uma solução de engenharia.

Porém, apesar de ser possível visualizar, muitas vezes, o problema como um todo ainda durante a etapa de especificação, é possível que esta especificação seja realizada incompleta ou inconsistente. Esta falha na especificação ocasiona muitas vezes no desenvolvimento de soluções que não são corretas nem completas, pois esta solução foi iniciada a partir de uma especificação que não era nem completa nem correta.

Este processo de desenvolvimento é bastante útil quando o sistema for formado por componentes tangíveis, que mantêm como verdade o conhecimento prévio das partes envolvidas. No entanto, problemas que envolvam IA passam a envolver componentes complexos com um comportamento não previsível, sendo necessário que o desenvolvimento passe a seguir um processo exploratório, como visto na figura 2.2. Este processo também é dividido em três etapas, que são: (i) formular as ideias; (ii) construir um protótipo; (iii) validar a ideia e recomeçar o processo. Sendo assim, este processo também é conhecido como RUDE (Run - Understand - Debug - Edit (PARTRIDGE; WILKS, 1987)), onde primeiramente é construído um protótipo ou executado o processo real para poder observar o seu funcionamento, procurando compreender como este funciona, para então ajustar possíveis erros deste protótipo e finalmente refinar o protótipo para que este chegue mais próximo do resultado esperado.

Desta forma, passa a ser bastante difícil realizar a especificação do sistema a ser construído de forma prévia, muito devido a natureza intangível do sistema, impedindo que o analista consiga enxergá-lo antes de construí-lo. Outra dificuldade é que os sistemas de IA no geral são sistemas de controle e não um sistema em si, o que faz com que a quantidade de variáveis a serem analisadas seja muito maior do que um sistema normal.

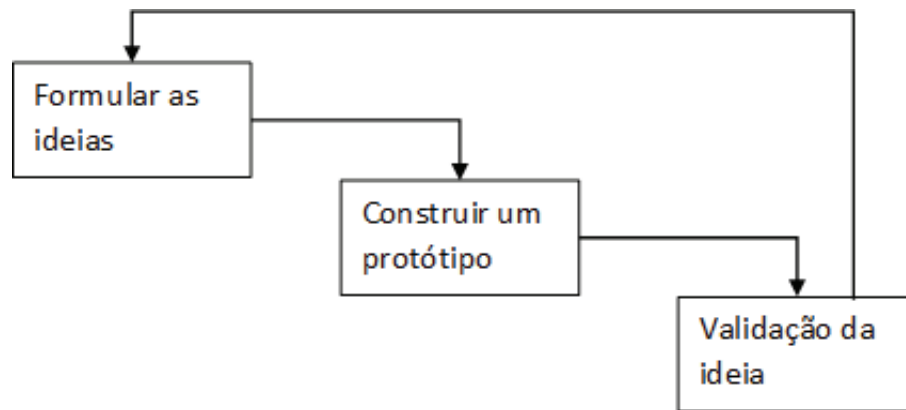


Figura 2.2: Processo de desenvolvimento de uma solução exploratória, muito utilizado na concepção de soluções que necessitam de IA.

Olhando para estes dois processos de desenvolvimento apresentados, o desenvolvimento de processos de engenharia e o processo exploratório, é possível identificar que, enquanto para o desenvolvimento exploratório é impossível responder perguntas simples como "qual sistema será construído?" e "será que o sistema foi construído de acordo com a especificação?", o desenvolvimento de sistemas de engenharia são bastante focados nestas duas perguntas. Ou seja, o processo de engenharia é bastante focado na especificação e na validação da solução, enquanto o processo exploratório não consegue garantir nenhum destes dois critérios.

No geral, os problemas da engenharia de software são mais fáceis de serem visualizados, pois lidam com artefatos concretos, como um processo existente em alguma empresa ou um sistema de controle previsível de algo já existente, enquanto os problemas da IA envolvem situações não controladas, envolvendo diversas incertezas e variáveis, tornando muito mais complexo o seu entendimento. Muitas vezes, os problemas da IA são impossíveis de serem avaliados previamente, partindo primeiramente para a experimentação, antes mesmo da análise. Este fato é verdade por não existirem métodos específicos para o campo da IA, principalmente com a capacidade de lidar com os artefatos que são importantes para a IA.

Um método desenvolvido e aprimorado para problemas da engenharia não pode ser simplesmente aplicado a um problema da IA, pois a natureza dos problemas dos dois campos são bastantes distintos (SOMMERVILLE, 1993). Segundo Partridge (PARTRIDGE, 1986), existem cinco características que diferem os problemas destes dois campos, que correspondem a: (i) tipo de resposta ao problema; (ii) dependência do contexto; (iii) modificações; (iv) especificação; e (v) definição. A primeira característica, referente ao tipo de resposta é bem simples de compreender ao olhar para dois problemas, um da engenharia como um sistema de controle de um motor de passo, onde uma sequência de sinais corretos irá fazer com que o eixo do motor gire continuamente; e um problema de IA como um jogo de xadrez, onde existem inúmeras possibilidades de se ganhar uma partida e todas elas estarão corretas.

Já o caso da segunda característica, a dependência do contexto, pode não influenciar os problemas de engenharia de software mais simples, como no caso do sistema de controle de

um motor de passo, pois independente do que ocorra com o contexto do ambiente, o sistema continuará realizando as mesmas atividades, enviando a mesma sequência de sinais. Enquanto para o problema do xadrez, qualquer mudança de contexto modificará a sequência de ações possíveis, sendo necessário realizar todas as avaliações necessárias todas as vezes que o contexto do problema sofrer alguma modificação. Da mesma forma com possíveis modificações no problema, enquanto para a engenharia de software as modificações são previsíveis e controladas, os problemas de IA são extremamente dinâmicos, impossibilitando especificações completas de um problema de IA por não ser possível alcançar sua completude. O que resulta na quarta característica, onde um problema de engenharia de software é possível ser completamente especificado enquanto um problema de IA não, devido a quantidade de incertezas e informações que só serão conhecidas durante a execução do sistema.

A quinta e última característica, que diz respeito a definição do problema, onde um problema de engenharia de software é possível ser definido de forma abstrata, sendo inclusive bastante incentivado a realização da definição e design da solução antes de realizar o desenvolvimento, enquanto problemas em IA tendem a ser definidos a partir do comportamento assumido e não de uma sequência de atividades, tornando complexo a descrição deste comportamento em uma maneira formalizada.

Desta forma, as características específicas de cada um dos problemas destes dois mundos, Engenharia de Software e IA, deixam um pouco mais claro que não se deve analisar problemas destes dois mundos de uma mesma forma. Primeiro porque a Engenharia de Software lida com características mais tangíveis à realidade, enquanto a IA lida com características comportamentais de um processo, tornando sua compreensão muito mais abstrata e o espaço de respostas possíveis muito maior e muitas vezes impossíveis de serem enumeradas. Segundo, a ES lida com um ambiente estático em que é possível ser descrito quase que por completo, muitas vezes sendo possível descrevê-lo completamente, e não tendo esta descrição invalidada ao longo da existência do software. Enquanto a IA lida com problemas difíceis de serem descritos por completo, pois o cenário quase nunca é totalmente conhecido, e mesmo quando conhecido, a quantidade de possibilidades torna esta descrição inviável devido a quantidade de variáveis e estados envolvidos. Além de ser um cenário que ao longo de sua existência estará sendo modificado, tornando muitas vezes as descrições iniciais inválidas. Assim, a descrição de um problema de IA envolve a compreensão do seu comportamento e suas possibilidades e não apenas atividades previsíveis e seus atributos, como ocorre na ES.

Voltando para o exemplo do jogo de xadrez, não importa apenas saber quais movimentos a rainha poderá realizar, é necessário também avaliar o risco de perdê-la ou perder alguma outra peça para cada possível movimento que venha a ser realizado. Além disto, para cada movimento realizado, um novo espaço de possíveis movimentos do adversário será aberto, que devem ser avaliados e levados em consideração ao escolher qual será o movimento a ser realizado. Posteriormente à realização da movimentação por parte dos dois jogadores, uma nova avaliação deverá ser realizada, possivelmente levando em consideração outros conjuntos de possibilidades

que poderá não ter muitas ações em comum com o conjunto da jogada anterior, sendo apenas um conjunto resultante a partir da jogada prévia. Enquanto para o sistema de controle de um motor de passo, o conjunto de passos a ser dado depende exclusivamente o sinal enviado anteriormente e o espaço de possibilidades não será modificado de acordo com este sinal prévio.

Enquanto tem-se a descrição de um problema de engenharia de software voltada a enumerar todas as possíveis ações, que muitas vezes são imutáveis, e suas relações com as entidades tangíveis do sistema, como é o caso do sistema de controle de motor de passo, tem-se nos problemas de IA a necessidade de uma descrição de problema que lidará muito mais com um cenário mutável a cada ação realizada e estas ações estarão relacionadas com propriedades muito menos concretas, como o risco de se perder uma peça do xadrez devido a um movimento ou com a previsão de movimentos do adversário.

Este alto nível de abstração necessário na hora de compreender um problema de IA, faz com que esta seja uma atividade difícil de ser realizada, fazendo com que muitos trabalhos de pesquisa no campo sejam realizados com os chamados '*toy problems*', que apesar de não serem relevantes na prática, são problemas muito bem conhecidos e capazes de provarem os conceitos estudados (RUSSELL; NORVIG, 2003) e assim permitirem a aplicação destes conceitos em outros problemas mais complexos. Desta forma, a grande maioria das pesquisas em IA utilizam os mesmos problemas já exaustivamente estudados previamente, para poder avaliar melhor os resultados e o desempenho dos algoritmos estudados. Como também, por ter uma descrição exata e concisa não se perde tempo na busca de descrever e analisar um problema real, já que estes problemas reais não possuem uma única descrição aceita por todos.

2.2 A Existência de Problemas Complicados na IA

Como visto, os problemas da IA lidam com características não tão simples de serem compreendidas e descritas de uma maneira formal, diferentemente dos problemas existentes em ES. Porém, este cenário ainda pode piorar. Alguns problemas da IA envolvem características que o tornam ainda mais complicados de serem analisados, pois podem envolver não apenas um único problema, mas vários. Estes casos estarão sendo considerados como problemas Multitarefas (MT).

Um problema MT envolve a realização de diversas tarefas que necessitam de um certo grau de racionalidade (nomeadas ao longo desta tese como tarefas de IA). Voltando para o problema do jogo de xadrez e a movimentação da rainha, foi visto que além de procurar identificar qual o melhor movimento da rainha a ser realizado, ainda existe a necessidade de avaliar o risco de se perder uma peça devido a próxima movimentação do adversário. Apenas esta ação de previsão já é considerada uma outra tarefa de IA, que já é bastante custosa por envolver diversas variáveis e possibilidades.

Olhando para as pesquisas acadêmicas de IA, vê-se que a maioria delas focam apenas em problemas com uma única tarefa, devido a dificuldade em relacionar o comportamento de uma

tarefa em consequência dos resultados obtidos da execução da outra tarefa (WEISS, 1999). É o que pode ser visto quando analisado o caso do xadrez e a rainha. A movimentação da rainha não está isolada apenas na sua própria movimentação, cada uma das suas possíveis movimentações irá resultar em um conjunto de possíveis movimentos do adversário que terá um risco de perder uma peça devido a este movimento. Então, avaliar o movimento da rainha não é apenas olhar se será possível atacar uma peça adversária, mas também avaliar o risco dos possíveis movimentos do adversário agregados à este movimento da rainha.

Mesmo na ES a existência de diversas tarefas em um sistema de software aumenta a complexidade do sistema (SOMMERVILLE, 1993). Porém, na ES estas tarefas se relacionam em uma forma já preestabelecida e possível de ser descrita e avaliada todas as suas possibilidades. Isto difere e muito de um problema MT em IA, pois a relação entre as tarefas pode não ser possível de ser descrita e provavelmente não será possível enumerar todas as possíveis relações entre elas.

Outra categoria de problemas que também torna ainda mais difícil a compreensão de problemas de IA, é a dos problemas que envolvem muitos agentes, denominados aqui de problemas Multiagentes (MA). Um agente é uma entidade que pode a partir da sua percepção do ambiente em que está inserido, realizar alguma ação capaz de modificar este ambiente. Geralmente, os problemas de IA envolvem agentes considerados inteligentes, ou seja, capazes de além de perceber o ambiente em que estão inseridos, realizar alguma ação racional que resulte na maximização de sua medida de desempenho, aumentando as suas chances de alcançar seu objetivo.

Estes problemas que envolvem mais de um agente inserem outra característica de difícil realização, que é o caso do relacionamento entre estes agentes (WOOLDRIDGE, 2009). Similarmente ao caso de mais de uma tarefa, os agentes passam a ter uma dependência entre suas ações, pois o resultado da ação de um agente poderá modificar a forma como um outro agente percebe este ambiente. Desta forma, será necessário um certo nível de coordenação em ambientes que possuam mais de um agente percebendo e atuando neste ambiente (WOOLDRIDGE, 2009).

No entanto, a coordenação entre agentes não é algo simples, principalmente quando aumentamos consideravelmente o número de agentes interagindo no ambiente. Cada agente em um ambiente terá seus próprios objetivos, porém alguns destes objetivos não podem ser alcançados de forma individual; neste ponto, os agentes obrigatoriamente devem se comunicar para poder em conjunto alcançar os objetivos gerais do sistema e não apenas os seus próprios objetivos específicos. Mas esta comunicação, irá por exemplo fazer com que as possibilidades de decisões se expanda de forma não controlada, permitindo ao SMA emergir seu próprio comportamento não previsível (RUSSELL; NORVIG, 2003) Esta característica de comportamento emergente não previsível em SMA, torna estes sistemas completamente diferentes dos sistemas de software mais comuns abordados pela ES, inclusive os sistemas da ES mais complexos. Pois, como visto anteriormente, os sistemas da ES possuem um comportamento previsível e estático.

Juntando estas duas categorias, os problemas MA e os MT, é encontrada uma nova

categoria com alta dificuldade em sua compreensão e principalmente em sua especificação, que são os problemas MAMT (problemas multiagentes e multitarefas). Um exemplo de problema desta categoria é a questão do gerenciamento de um time de futebol. O gerenciamento de um time de futebol passa por diversas instancias de problemas, como a gestão dos gastos, decidindo se irá comprar novos jogadores, o quanto irá gastar em cada jogador, se irá melhorar as instalações de treino, se irá melhorar as instalações do estádio, ou se irá vender algum jogador, quanto poderá pagar de salário a cada jogador, como será formada a comissão técnica, qual treinador, quais e quantos conselheiros farão parte da administração do clube, entre outros problemas. Porém, estes problemas não cabem a apenas uma única pessoa, por exemplo, a decisão de comprar um determinado jogador, passa pela indicação do técnico, para o conselho do clube, que vai para o conselho fiscal, passando pelo departamento financeiro e muitas vezes pelo responsável pelo patrocínio do clube, para averiguar o impacto da contratação às marcas associadas ao time, finalmente pelo aval do presidente e encerrando com o departamento médico. Mas o exemplo da compra de um jogador, por exemplo, não segue uma linha de raciocínio simplesmente linear. Muitas vezes algumas das instâncias envolvidas podem estar se preocupando com outros problema e a solução de um destes outros problemas podem resultar na anulação da compra do jogador, desencadeando uma nova rota de decisões possíveis para procurar avaliar se ainda é possível solucionar o problema ou simplesmente retornando para o técnico procurar um substituto para o atleta, para ser novamente avaliado. Desta forma, o controle de um clube de futebol não é uma atividade simples, sendo necessário lidar com diversos problemas e entidades responsáveis por solucioná-los.

Um outro exemplo é um jogo de estratégia em tempo real, como por exemplo Age of Empires ([EMPIRES, 1997](#)) ou Starcraft ([STARCRAFT, 1998](#)). Estes jogos envolvem alguns jogadores, humanos ou computacionais, batalhando por derrotar o inimigo e para isto realizando a coleta de recursos, gerenciando seu exército e gerenciando os confrontos diretos com o adversário em busca de destruir as unidades inimigas. Em um jogo como Starcraft, diversas atividades devem ser realizadas, como a coleta de recursos, a construção de edificações, a produção de novas unidades militares ou unidades trabalhadoras, a pesquisa de novas tecnologias, patrulha, exploração do mapa, entre outras. Por exemplo, em um determinado momento do jogo, o jogador necessitará enviar tropas militares para combater o inimigo, construir alguma edificação de defesa (como uma torre de mísseis) e também construir uma nova fabrica de tanques, enviar trabalhadores coletar novos recursos, avaliar quanto poderá gastar de recursos em cada nova edificação ou unidade, etc., e todas estas atividades devem ser realizadas ao mesmo tempo. No caso de um jogador computacional, estas atividades podem e devem ser divididas entre os diversos agentes existentes no jogo, pois facilitará a questão de coordenação das atividades em paralelo, mas a coordenação e comunicação entre as entidades existentes será uma barreira limitadora, devido a quantidade de informações necessárias de serem trocadas.

Estes dois exemplos, são claras situações de um problema MAMT, onde é possível identificar uma grande quantidade de atividades que devem ser realizadas, com suas interdependências

e além de serem problemas que deixam de certa forma clara a necessidade de mais de um agente para controlar estas atividades. Como também, é possível identificar que cada problema já é relativamente difícil de ser lidado se for tratado isoladamente. Olhando para o caso de procurar construir um sistema que lide com todos estes problemas existentes em conjunto, tem-se um acréscimo drástico da dificuldade. Isto ocorre devido a necessidade de lidar com a coordenação das atividades realizadas por cada agente e a comunicação existente entre estes agentes. Esta coordenação complica a especificação do problema em si, principalmente por não ser possível identificar um processo que seja voltado a especificação das necessidades de IA e principalmente no caso de problemas MAMT.

Mesmo sendo senso comum em outras áreas que a especificação do problema é uma atividade importante e que minimiza futuros problemas no decorrer da construção do sistema, em IA não é possível encontrar processos ou métodos que permitam lidar corretamente com a especificação de problemas de IA. Principalmente levando em consideração as questões do que de fato é importante compreender em um problema de IA e que os difere dos problemas de ES, os quais existem diversos processos e métodos para auxiliar esta atividade, mas que não podem ser aplicados diretamente para problemas de IA.

2.3 Pontos Importantes na Compreensão de um Problema de IA

Foi visto até o momento, que existem algumas diferenças entre os problemas de ES e de IA e que estas diferenças impedem que estes problemas sejam tratados da mesma forma. Mesmo que o conhecimento a respeito de compreensão de problemas seja muito mais maduro na ES, este não pode ser aplicado diretamente aos problemas que devem ser solucionados através de técnicas de IA. Como consequência, se faz necessário a existência de uma abordagem direcionada que consiga extrair as informações necessárias aos problemas de IA.

Um destes motivos é o fato de problemas de IA não lidarem diretamente com propriedades concretas e reais, e sim com a representação de comportamentos, onde não existem apenas repostas certas ou erradas. Sendo necessário que este comportamento seja modelado de alguma forma, mesmo quando está se lidando apenas com o problema em si, não pensando ainda em possíveis soluções. Para isto é indicado verificar como este problema pode ser modificado durante a tentativa de solucioná-lo, como por exemplo, se a forma de solucioná-lo deverá ser modificada devido a alguma modificação do ambiente ou devido ao tempo necessário no processamento da solução.

Já na avaliação de como se comportará um problema é possível verificar que o ambiente realiza um papel importante na definição dos problemas em IA. O que não é tão perceptível no caso da ES, já que no geral o ambiente dos problemas em ES são previsíveis, não exercendo muitas influências no problema, enquanto que os problemas de IA estão inseridos em um

ambiente dinâmico, exigindo que os problemas se adaptem constantemente, o que pode tornar o problema imprevisível (RUSSELL; NORVIG, 2003). Desta forma, a descrição do ambiente é uma atividade extremamente necessária e quanto melhor detalhado, mais fácil será a compreensão do problema a ser solucionado. Como por exemplo, listar os dados do ambiente que exercem alguma influência direta ou indireta relacionada ao problema a ser solucionado. Estas variáveis do ambiente permitirão delimitar o escopo do problema no ambiente e o que pode alterar a solução deste.

O tempo é uma característica que pode influenciar o problema de diversas formas, como por exemplo a necessidade de atender determinadas restrições temporais. Ou seja, um determinado problema poderá ter que ser resolvido em um tempo limite e para isto algumas possíveis soluções deverão ser descartadas por não atenderem a este critério. Sendo necessário pontuar que estas restrições temporais podem ser dinâmicas, podendo ser modificadas durante a execução do software que deverá solucionar o problema, como em um tempo t_1 o problema deverá ser solucionado em t' enquanto em um tempo t_2 deverá ser solucionado em t'' .

Outra influência temporal na solução dos problemas é com relação ao momento em que o problema deverá ser solucionado. Os problemas de IA estão no geral envolvidos em um sistema maior, onde estes problemas deverão ser executados em instantes definidos. Esta informação é importante porque, em boa parte, as soluções dos problemas de IA são formadas por algoritmos extremamente complexos que necessitam de um grande gasto computacional para serem executados, sendo um desperdício executá-las ininterruptamente. Desta forma, a identificação dos momentos temporais em que a solução do problema será processada se torna uma informação extremamente necessária para não desperdiçar poder computacional.

Além do ambiente e suas relações com os problemas de IA, também foram vistos motivos intrínsecos aos problemas de IA que os tornam de difícil compreensão, como é o caso dos problemas MAMT. Como por exemplo, o fato de ser necessário lidar com diversos subproblemas que possuem uma grande interdependência, onde as atividades necessárias para solucionar um destes subproblemas acarretará em modificações consideráveis na percepção do outro subproblema, sendo necessário realizar toda uma análise novamente nas possibilidades. Desta forma, a identificação destes subproblemas permite uma melhor avaliação do que de fato compõe um problema de IA, permitindo assim melhor avaliar possíveis soluções a serem desenvolvidas.

Além da parte multitarefas, os problemas MAMT também devem lidar com a existência de mais de um agente inserido no mesmo ambiente para alcançarem juntos um objetivo em comum. Desta forma, estes problemas que necessitem de mais de um agente envolvido deverá se preocupar com a existência da comunicação entre estes agentes o que aumentará a complexidade na coordenação das atividades realizadas por cada agente envolvido.

Desta forma, descrever um problema de IA é uma atividade diferente da atividade de descrever um problema em ES, devido a necessidade de representar algumas características intangíveis à visão do mundo em ES. Em resumo, alguns tópicos importantes na descrição de um

problema em IA são: (i) informações do ambiente que podem influenciar o problema; (ii) como o problema se comportará com as modificações no ambiente; (iii) a necessidade de se adaptar a restrições temporais; (iv) quando o problema deverá ser solucionado; (v) como lidar com a possibilidade de encontrar subproblemas envolvidos ao problema original.

2.4 Conclusão

Durante este capítulo foi mostrado que as abordagens são diferentes para os problemas em ES e da IA. Esta diferença faz com que seja necessário um método específico para poder analisar os problemas da IA. Neste ponto encontramos a proposta desta tese, um método específico para analisar os problemas que envolvam IA. Ao longo da tese será avaliada a necessidade de uma nova abordagem e Icelus será apresentado em detalhes.

3

A compreensão de problemas em diversos campos do conhecimento

De acordo com o Business Dictionary ([COMBLEY, 2011](#)) um problema seria: "uma lacuna percebida entre o estado existente e um estado desejado ou um desvio de uma norma, padrão ou status quo. Embora muitos problemas possam ter várias soluções (os meios para eliminar a lacuna ou corrigir o desvio), as dificuldades surgem onde esses meios estão, podendo não ser óbvio ou não estando imediatamente disponíveis". De fato, um problema é uma situação em que se pretende alcançar um determinado estado diferente do atual e sua dificuldade estará atrelada às atividades que devem ser realizadas (soluções) para alcançar este estado. Quanto maior a dificuldade apresentada na definição de uma solução, mais importante será o conhecimento do analista a respeito do problema.

Desta forma, para minimizar as dificuldades em definir e construir uma solução é necessário despender algum tempo buscando compreender o problema a ser abordado. Com este intuito, existem algumas atividades que permitem auxiliar e/ou guiar o analista em busca de um melhor conhecimento do problema a ser solucionado. Neste trabalho, estas atividades estarão divididas em três grandes grupos: (i) no campo conceitual, que podem e devem ser utilizadas em diversos campos de pesquisas, como é o caso das atividades de análise e modelagem; (ii) em outros campos de pesquisa, onde é possível verificar a importância da compreensão do problema e como esta atividade já está bem amadurecida com a presença de métodos e ferramentas que podem auxiliar esta tarefa de compreender o problema; e (iii) no campo em que este trabalho mais se aproxima, o campo dos SMA, onde as ferramentas ainda não são tão maduras, deixando lacunas abertas na atividade de compreender os problemas.

3.1 Campo conceitual

Em alguns casos, a solução de um problema pode ser proposta sem muitas dificuldades, mas existem outros casos em que a complexidade do problema não permite que uma solução seja proposta de forma tão direta. Estes casos mais difíceis de serem solucionados necessitam de um

maior esforço na sua compreensão, tornando crucial o estudo do que de fato seria o problema. Em contrapartida ao esforço despendido na compreensão dos problemas mais difíceis, tem-se a facilidade de encontrar e propor uma solução para os problemas bem compreendidos (PRESSMAN, 2001).

A complexidade de um problema pode estar presente de diversas formas dependendo do domínio em que estará imerso. No campo da Computação, a complexidade de um problema é mais comumente determinada de acordo com os recursos requeridos para executar um determinado algoritmo; sendo o tempo o recurso mais comum utilizado na identificação da complexidade (BIIRGISSER; CLAUSEN; SHOKROLLAHI, 1997), mas podemos ter a complexidade atrelada a outros recursos como espaço e distribuição de processamento (DASKALAKIS; GOLDBERG; PAPADIMITRIOU, 2009). Outra forma de identificarmos a complexidade de um problema computacional é referente a quantidade de propriedades transmitidas por um objeto a um observador, ou seja, a quantidade de propriedades existentes em um determinado estado; como também a quantidade de estados possíveis existentes. Já para a Engenharia de Software, a complexidade pode ser medida de acordo com as iterações existentes entre os vários elementos presentes na solução (PRESSMAN, 2001).

No geral, estas características da complexidade de um problema em Computação estão ligadas a quantidade de variáveis (tarefas, tempo, iterações, etc) que podem estar envolvidas no problema. O grau da complexidade estará diretamente relacionada a capacidade de gerenciar todas estas variáveis ao mesmo tempo e a nossa limitação em analisar dados simultaneamente (HALFORD et al., 2005). Sendo necessário, no caso destes problemas mais complexos, depender algum tempo para realizar etapas de análise do problema, o quebrando em pedaços menores e mais simples de serem compreendidos, e modelagem, realizando uma síntese das informações do problema que sejam mais relevantes. Este processo de análise e modelagem pode ser visto na figura 3.1.

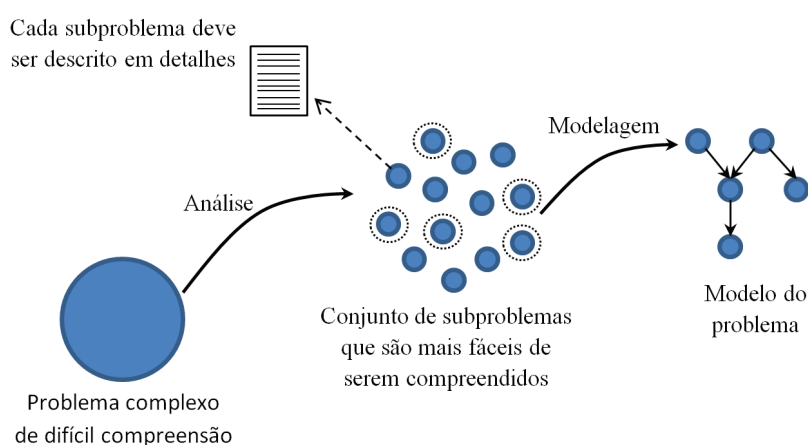


Figura 3.1: Processo de análise e modelagem de um problema.

3.1.1 Análise

Como visto, os problemas que envolvam um grau elevado de complexidade dificultam a sua compreensão, muitas vezes pelas limitações cognitivas e neuropsicológicas envolvidas no processo (HALFORD et al., 2005), passando a ser de extrema importância quebrar este problema em partes menores que facilitem a sua compreensão. Este processo de decomposição de um problema complexo em problemas menores e mais simples de serem compreendidos é conhecida como Análise.

Um processo de análise, de acordo com o Dicionário Oxford (BLACKBURN, 2008) é "um processo de quebra de um conceito em partes mais simples, para que sua estrutura lógica seja revelada. ... Uma análise filosófica irá fornecer, de forma científica, uma abordagem objetiva para problemas tradicionais. Exatamente como um matemático pode fornecer uma definição de um conceito complexo, a partir de uma sequência de termos e operações mais simples, desta forma um filósofo deve ser capaz de identificar a natureza de um conceito complexo em termos de ideias mais simples que o compunham". Seguindo este conceito, o resultado de uma análise de um problema seria a quebra deste em outras partes menores, denominadas sub-problemas, e mais fáceis de serem compreendidas e solucionadas. Cada um destes sub-problemas irão representar uma pequena parte da lacuna ou desvio entre o estado atual e o estado desejado, sendo a solução desejada a solução de parte ou todos estes sub-problemas (WHIMBEY; LOCHHEAD; NARODE, 2013).

Segundo Hillier (HILLIER; LIEBERMAN, 2001), esta análise deverá "conter uma descrição clara do problema, identificando suas metas, restrições do que pode ou não ser feito, conceitos chaves, relacionamentos entre o problema e o ambiente em que se encontra inserido e ações alternativas que podem ser feitas"; enquanto uma boa descrição do problema deverá ser clara e precisa, identificando o que será abordado, seus conceitos chaves, termos, variáveis e escopo (HERNON; SCHWARTZ, 2007). Desta forma, este processo de análise do problema irá permitir que uma solução fosse proposta de forma mais fácil, reduzindo custos e permitindo uma melhor visão do que é necessário na solução (WHIMBEY; LOCHHEAD; NARODE, 2013).

Esta descrição detalhada deverá ser o resultado deste processo de análise, permitindo uma melhor compreensão do problema pela equipe de desenvolvimento. Assumir que este processo é dispensável por gastar um tempo que poderia ser utilizado para codificar a solução é um erro bastante comum e segundo Booch, iria representar uma inconsistência no processo produtivo da equipe, pois se não há tempo para se realizar uma boa análise, sempre existirá tempo para consertar uma solução mal feita (BOOCH, 1996). Porém, este tempo gasto na correção de erros na solução é muito maior do que o que seria gasto para realizar uma boa análise.

Como também, esta descrição proveniente da análise serve como uma primeira validação do entendimento do problema. Permitindo evitar que uma solução não represente o que realmente o cliente desejaria, pois de nada adianta ter uma ótima solução, se ela não resolve o problema inicial. Neste ponto, a análise é crucial, pois permite que conceitos específicos do problema

sejam melhor descrito e compreendidos.

O principal motivo de se necessitar de um processo de análise se dá ao fato dos problemas de comunicação existentes entre os clientes e o time de desenvolvimento. Tanto que alguns autores já chegam a comentar que a fase de entendimento do problema e detalhamento dos requisitos do projeto seja o próximo gargalo que deverá ser eliminado no processo de desenvolvimento de software atual (ADZIC, 2009). A comunicação entre estes dois mundos, clientes do problema e o time de desenvolvimento, gera esta dificuldade devido a (ADZIC, 2009): (i) os requisitos imperativos impostos pelo clientes são extremamente fáceis de serem mal interpretados; (ii) mesmo as características mais óbvias não são tão óbvias e podem ser mal interpretadas; e (iii) requisitos são super-especificados, sendo expressos como sendo a solução, e focam no que deve ou não ser feito, impedindo que a equipe de desenvolvimento argumente se esta é realmente a melhor forma de alcançar o que os clientes desejam.

Para realizar este processo de análise existem algumas ferramentas, como por exemplo a árvore de problemas (SNOWDON; SCHULTZ; SWINBURN, 2008). Esta ferramenta procura quebrar o problema inicial em subproblemas menores a partir da identificação das causas que levaram ao problema analisado e das consequências deste mesmo problema. Para tanto, pode-se buscar estes novos subproblemas a partir da realização de perguntas como qual a causa que originou o problema atual e qual é a consequência deste, tornando mais fácil e intuitivo o processo de quebra do problema inicial em problemas menores e mais fáceis de serem lidados. Desta forma, a aplicação em uma etapa da árvore de problemas irá expandir o problema analisado em mais um nível, a partir da identificação de subproblemas que geraram o problema analisado como também a identificação de novos subproblemas que são as consequências da análise.

Além disto, é possível seguir com a análise, permitindo expandir a árvore de problemas em n níveis desejados. Apesar do nome, a árvore de problemas está mais próxima de um grafo do que de uma árvore, por não ter necessariamente um nó raiz e poder simplesmente ser expandida nas duas direções. Um modelo de uma árvore problemas pode ser visto na figura 3.2.

Mesmo com a identificação de diversos subproblemas relacionados ao problema inicial e com a construção da árvore de problemas que constrói uma relação de causa e consequência entre estes subproblemas, este é apenas uma das etapas da difícil tarefa de transpor as informações do problema para o restante do processo de construção da solução. Onde, por definição, a solução construída deverá representar apenas uma simplificação da solução do problema, sendo apenas uma das possíveis soluções existentes.

Desta forma, o próximo passo seria a construção de um modelo do problema, onde apenas os subproblemas que de fato importam para a solução simplificada deverão estar presentes. Um modelo é uma visão simplificada da realidade complexa do problema e sua definição permitirá que a equipe que irá trabalhar na solução se preocupe com os dados relevantes como também facilitará a comunicação entre os stakeholders do problema.

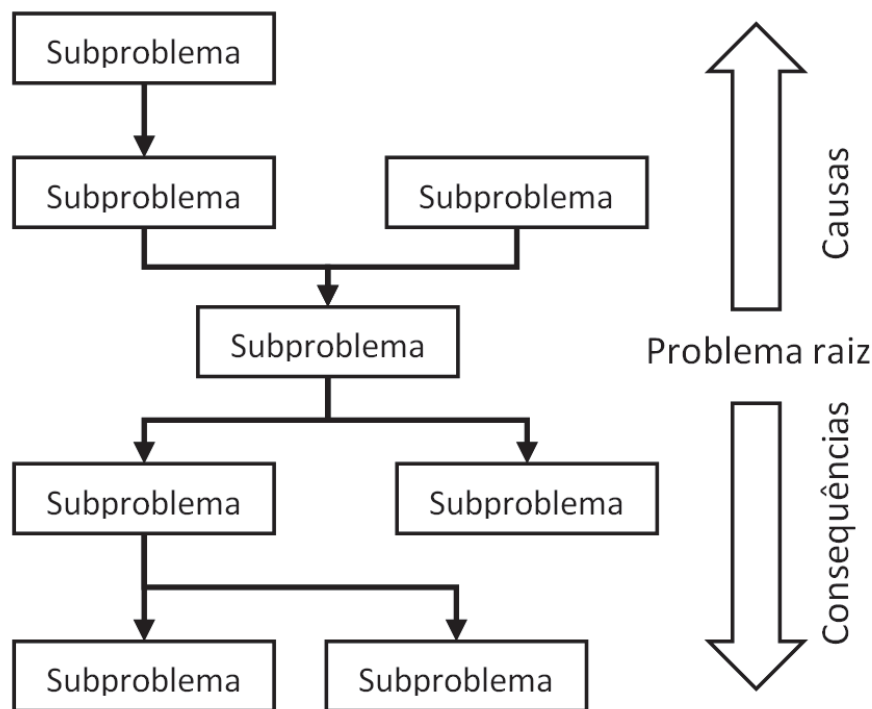


Figura 3.2: Exemplo de uma árvore de problemas, onde os nós superiores são as causas e os nós inferiores são as consequências de um determinado subproblema.

3.1.2 Modelagem

A primeira etapa, a análise, é responsável por expandir o problema, o descrevendo em partes menores; como também ser responsável por detalhar a maior quantidade de informações possíveis do objeto de estudo. A etapa a seguir, a modelagem, deve ser responsável por realizar a síntese destas informações em um modelo capaz de auxiliar a compreensão do objeto.

Um modelo deve ser utilizado para alcançar um entendimento do problema estudado, simplificando uma visão complexa da realidade (ERIKSSON; PENKER, 2000). Para isto, este modelo irá criar uma abstração, permitindo eliminar informações irrelevantes e focar nos aspectos mais importantes que devem ser analisados para construir a solução. Outra importante vantagem é a capacidade de facilitar a comunicação entre os diferentes stakeholders envolvidos. Um modelo também permitirá que a solução seja estruturada mais facilmente, permitindo que os responsáveis apresentem um maior foco na construção desta solução. Além disto, este modelo poderá ser utilizado como base para a definição dos requisitos especificados nas próximas etapas.

É possível encontrar diversas vantagens para a utilização de modelos, Stevenson (STEVENSON; SUM, 2009) lista nove destas vantagens: (i) são mais simples de serem utilizados do que a situação real; (ii) permite visualizar mais facilmente a necessidade de mais informações a respeito do problema; (iii) provê uma abordagem sistemática para a solução do problema; (iv) aumenta a compreensão do problema; (v) possibilita a análise de soluções alternativas; (vi) exige que os stakeholders sejam específicos com relação ao objetivo do problema; (vii) serve como uma ferramenta consistente para avaliação; (viii) permite usufruir das vantagens da discretização

matemática do problema; e (ix) provê um formato padronizado para analisar o problema.

Sintetizando, a definição de um modelo após a etapa de análise irá permitir que as informações mais relevantes sejam identificadas e permitirá ter uma visão mais abrangente do problema. Assim sendo, este modelo permitirá facilitar o desenvolvimento da solução de um problema mais complexo, pois conterá uma representação mais simplificada do problema inicial, como também permitirá uma primeira avaliação pelos stakeholders envolvidos no projeto.

3.2 Em outros campos da pesquisa

Como visto, a dificuldade em solucionar um problema é devido à solução não ser óbvia ou não estar imediatamente disponível (BLACKBURN, 2008). Buscando minimizar os casos em que a solução não seja óbvia, aconselha-se despende algum tempo analisando o problema, o simplificando, de forma que seja mais fácil visualizar possíveis soluções. Porém, não são todas as áreas que prestam a devida importância às etapas iniciais de análise e compreensão de um problema, mesmo com a existência de ferramentas conceituais que poderiam ser utilizadas em inúmeros campos de pesquisa. Enquanto algumas áreas, como Engenharia de Software (ES), Pesquisa Operacional (PO) e Engenharia de Conhecimento (EC), dão a devida importância à análise do problema, por reconhecer que quanto melhor especificado este for, mais fácil será definir e construir uma solução possível (SOMMERVILLE, 1995), outras áreas não dão importância como deveriam, como é o caso da área de SMA.

Olhando para as áreas da Computação que dão importância à compreensão de problemas, é possível identificar que estas áreas passaram a criar técnicas capazes de guiar as etapas de captação de informações do problema e de seu domínio. Técnicas que permitem uma visão mais clara do problema e, conseqüentemente, do que deve ser solucionado. A aplicação destas técnicas também permitirá uma análise mais simplista do problema, por permitir enxergar partes (subproblemas) e não apenas o problema como um todo. A seguir será mostrado como três áreas distintas lidam com as etapas iniciais de estudo do problema e qual a sua real importância no seguimento do processo da busca por uma solução.

3.2.1 O Campo da Engenharia de Software

Para entender como a Engenharia de Software (ES) lida com a especificação do problema é necessário primeiro compreender o que de fato é ES. Na literatura são encontradas diversas definições do que seria ES, como: (i) é a criação e utilização de princípios sólidos de Engenharia de forma a obter softwares econômicos que estejam disponíveis e funcionem de forma eficiente em máquinas reais (por Friedrich Bauer (NAUR; RANDELL, 1969)); (ii) é a aplicação de uma abordagem sistemática, disciplinada, quantificada no desenvolvimento, operação e manutenção do software (por IEEE (IEEE, 1993)); (iii) é a análise, design, construção, verificação e gerenciamento do software (por Roger Pressman (PRESSMAN, 2001)). De forma geral, a Engenharia de

Software é uma área computacional que busca melhores resultados dos códigos desenvolvidos, seja no nível de desempenho ou no nível de redução de custos ou, principalmente, no nível de qualidade do código desenvolvido.

Ao longo da evolução da ES, diversas técnicas foram aprimoradas de modo que o sistema desenvolvido seja mais fácil de ser mantido, modificado e gerenciado, permitindo que possua uma menor quantidade de erros. Com a evolução aconteceu o surgimento de alguns processos para o gerenciamento do desenvolvimento de um Sistema de Software, entre estes processos é possível encontrar o processo denominado como RUP (Rational Unified Process ([KRUCHTEN, 2003](#))). RUP possui seu foco na produção de documentação ao longo de todas as etapas do processo de desenvolvimento do software e na modelagem visual do sistema a ser desenvolvido através da utilização de UML, sendo assim considerado um método extremamente pesado ([KHAN; QURASHI; KHAN, 2011](#)). RUP é baseado em seis práticas: (i) desenvolvimento iterativo e incremental; (ii) gerenciamento de requisitos; (iii) aplicação de uma arquitetura baseada em componentes; (iv) modelagem do sistema de forma visual - UML; (v) verificar de forma contínua a qualidade do sistema em desenvolvimento; e (vi) controle de mudanças.

RUP divide o ciclo de vida de um projeto de software em quatro fases como pode ser visto na Figura 3.3. Em cada fase há a presença de até nove disciplinas, que é como RUP conceitua as atividades a serem desenvolvidas durante o ciclo do projeto. Em algumas fases certas disciplinas consomem mais tempo do que outras, tendo até algumas que nem precisam ser realizadas obrigatoriamente. A primeira disciplina é a de Modelagem de Negócio, responsável por analisar e definir um modelo matemático capaz de representar o problema a ser resolvido pelo software. Para realizar esta etapa, o analista entra em contato direto com o domínio e com o especialista do problema, procurando compreender o problema e a partir daí realizar uma especificação do mesmo e posteriormente sua modelagem em alguma linguagem matemática. Desta forma, esta etapa lida com a transposição do problema do mundo real para uma linguagem mais próxima da equipe de desenvolvimento, permitindo que os engenheiros de software compreendam o problema, com a finalidade de que possam desenvolver a solução da melhor forma possível ([VAN VLIET; VAN VLIET; VAN VLIET, 1993](#)).

A segunda disciplina, Requisitos, trata em documentar os pedidos do cliente, conhecidos como requisitos. A próxima atividade, de Análise e Design, procura realizar a modelagem visual do sistema a ser desenvolvido, procurando assegurar que os requisitos sejam atendidos e facilitar o entendimento da solução. A quarta disciplina, denominada Implementação, é utilizada para o desenvolvimento do sistema. A disciplina seguinte, Testes, procura garantir o correto funcionamento do sistema desenvolvido na etapa anterior. A sexta disciplina, Implantação, realiza o lançamento do produto na empresa do cliente, assegurando que o mesmo irá funcionar corretamente dentro do ambiente. Além destas seis disciplinas, existem outras três disciplinas de suporte organizacional, que são: (i) Gerência de Configuração e Mudanças, procura gerir e armazenar de forma adequada as alterações realizadas nos artefatos do projeto, como documentação e código; (ii) Gerência de Projeto, que lida, entre outras coisas, com as métricas, qualidades,

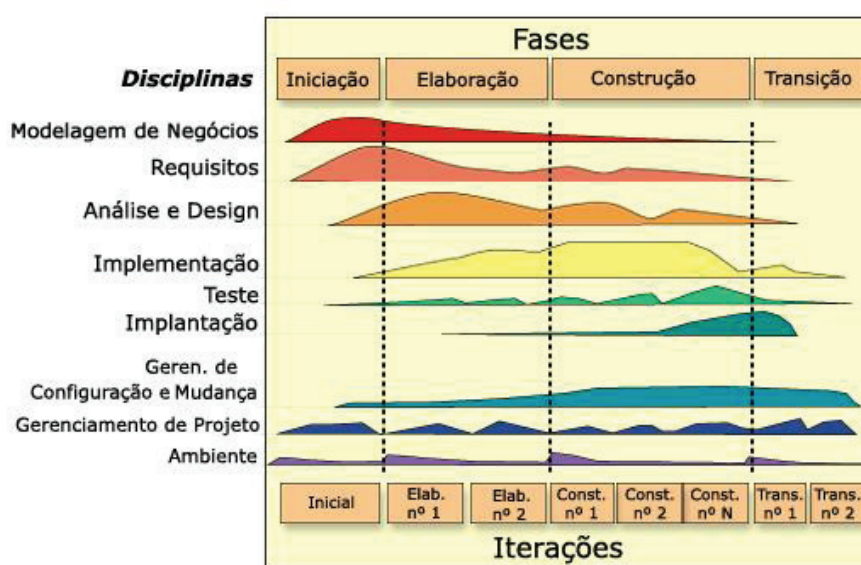


Figura 3.3: Fases e disciplinas de RUP. [Fonte: www.ibm.com/].

riscos e prazos do desenvolvimento do projeto; e (iii) Ambiente, focada em fornecer o que for necessário para reproduzir o ambiente em que o produto final será implantado, minimizando possíveis problemas durante a fase de implantação.

Porém, RUP e métodos similares são muito rígidos e mantem certa distância entre o cliente e a equipe de desenvolvimento, havendo esta aproximação no momento da entrega dos artefatos do projeto. Procurando evitar esta distância e tornar o desenvolvimento mais suscetível a modificações surgiram os métodos ágeis ([MARTIN, 2003](#); [FOWLER](#); [HIGHSMITH, 2001](#)) (Programação Extrema (Extreming Programming (XP)) ([BECK, 2000](#)), Scrum ([SCHWABER, 2004](#)) e Desenvolvimento Guiado por Testes (Test Driven Development - Desenvolvimento baseado em testes (TDD)) ([BECK, 2003](#); [JANZEN](#); [SAIEDIAN, 2005](#))). Estes métodos enfatizam uma participação mais ativa do cliente, reduzindo o tempo de descoberta de possíveis falhas no projeto ([SOMMERVILLE, 1995](#); [PRESSMAN, 2001](#)), principalmente nas fases iniciais do projeto, como compreensão do problema e proposta da solução (em RUP seriam as disciplinas de Modelagem de Negócios e Requisitos).

A primeira disciplina, Modelagem de Negócios, de RUP possui uma relevância considerável por ser responsável por transpor o problema do seu domínio original para o domínio de software, onde a solução será construída. Para isto, o conhecimento que existe a respeito do problema real deve ser formalizado, permitindo sua representação no mundo computacional e facilitando a sua compreensão pela equipe de software. Um exemplo seria a especificação do processo de saque em um caixa eletrônico. A modelagem deste processo deverá representar as atividades que são realizadas em um caixa normal do banco, para isto o analista deverá adquirir o conhecimento do processo bancário a partir de um especialista; a importância desta etapa se dá devido ao fato de que qualquer especificação errada (como não requerer do cliente do banco uma autenticação) acarretar em problemas sérios na solução e consequentemente durante

o desenvolvimento (erros identificados após esta etapa de análise serão corrigidos tardiamente, sendo necessário aumentar o tempo do projeto e consequentemente o seu custo).

Para realizar este processo de transferência de conhecimento pode ser utilizado, por exemplo, o *Business Process Management (BPM)* (OWEN; RAJ, 2003; VAN DER AALST; HOFSTEDE; WESKE, 2003), que é um método para a modelagem de processos de negócio. Antes mesmo da especificação de BPM, diversas empresas já modelavam e descreviam seus processos utilizando variadas formas e ferramentas, o que não impedia a existência de diversas falhas ou insucessos ao longo dos anos, pois os processos eram modelados e descritos sem auxílio de um método. Desta forma, o BPM foi desenvolvido na busca de padronizar e guiar a forma de proceder na modelagem dos processos de negócios.

Em BPM, O ciclo de vida de um processo é dividido em seis fases: (i) Automação, durante esta etapa é realizada a automação do processo de forma a ter um maior controle sobre o funcionamento deste, eliminando possíveis ações externas; (ii) Design, utilizando uma linguagem específica *Business Process Modeling Notation (BPMN)* (WHITE, 2004)) realiza a representação do processo em um modelo para facilitar o entendimento do usuário; (iii) Simulação, com o modelo definido e distribuída as responsabilidades aos participantes do processo, é então possível simular a execução do processo e avaliar o possível resultado; (iv) Execução, o processo estando bem definido e modelado pode então ser executado; (v) Acompanhamento, esta etapa idealmente deve ser realizada durante todas as etapas do BPM, permitindo assim averiguar se tudo está ocorrendo da melhor maneira e então identificar possíveis falhas; e (vi) Otimização, sempre que for identificado algum ponto possível de ser otimizado o mesmo deve ser realizado, procurando sempre alcançar os melhores resultados.

A utilização de BPM permite que os processos de negócio de uma empresa sejam documentados e formalizados. A grande vantagem para a empresa é ter a possibilidade de reproduzir o processo e também passar a ter métricas a fim de comparar o desempenho em cada execução deste processo, podendo assim buscar melhorias e ganho de desempenho com o intuito de deixá-lo mais ágil. Para o caso do desenvolvimento de um sistema computacional que auxilie este processo, a utilização de BPM permitirá reduzir as ambiguidades e possibilidades de não entendimento pelos engenheiros de software, já que o processo estará de certa forma, formalizado e documentado sem ambiguidades.

Desta forma, foi verificado que a primeira etapa da abordagem da ES em solucionar um problema é através da Modelagem de Negócios. Esta primeira etapa é responsável por traduzir o processo (problema) que existe no mundo real para uma linguagem matemática capaz de diminuir as possíveis ambiguidades e facilitar a sua compreensão pelo restante da equipe de desenvolvimento. Por exemplo, o processo de interagir com um caixa eletrônico de um banco e solicitar um saque em dinheiro podem ser representados em BPMN como na Figura 3.4. Com este modelo em mãos, o restante da equipe visualiza melhor o problema facilitando o restante das etapas do desenvolvimento, deis da elicitação de requisitos ao desenvolvimento e testes da solução.

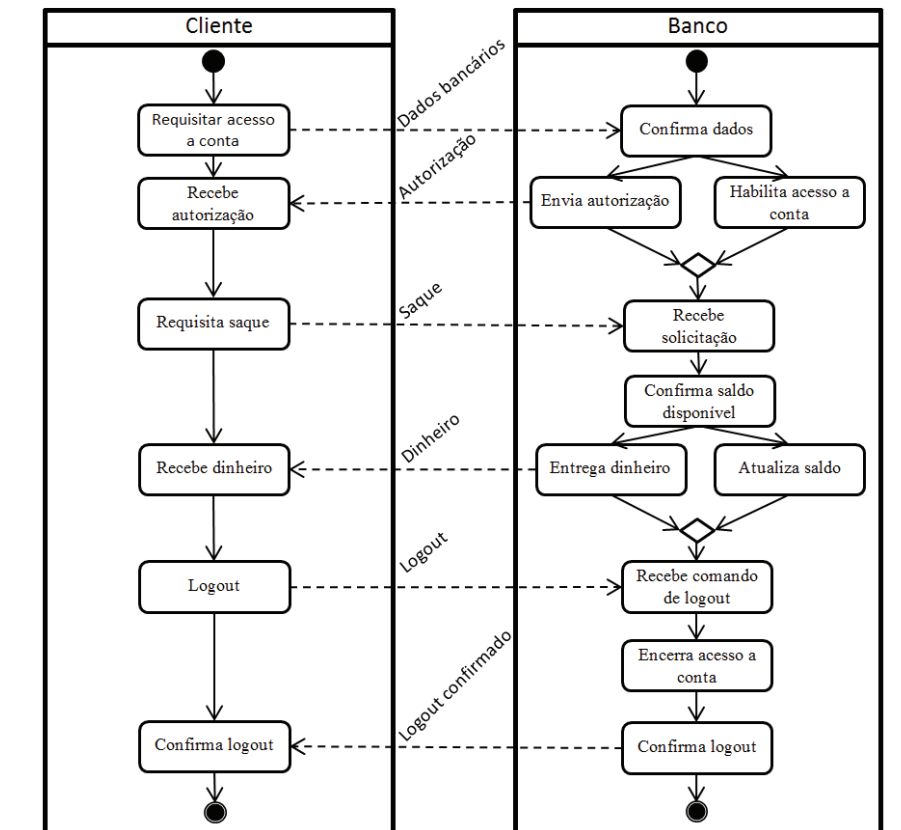


Figura 3.4: Modelo BPM para o problema de transações bancárias em um caixa eletrônico.

Com isto, para a ES, quanto mais complexos forem os problemas e mais tarefas existirem nestes, mais importante serão a especificação e sua posterior definição em um modelo matemático. Consequentemente, esta especificação e/ou modelo permitirá facilitar a continuidade do desenvolvimento da solução, reduzindo custos e tempo de projeto.

3.2.2 O Campo da Pesquisa Operacional

Outro campo da Computação que despende tempo com a compreensão do problema e sua especificação é a Pesquisa Operacional (PO). Primeiramente é necessário entender o que seria o campo da PO e como esta surgiu, para poder verificar como esta lida com a modelagem do problema. Na literatura podem-se encontrar algumas definições da área de PO, Winston ([WINSTON; GOLDBERG, 2004](#)) define PO como uma abordagem científica para sistemas de decisões, procurando determinar qual a melhor forma de operar o sistema, geralmente sendo necessário requisitar a alocação de recursos escassos. Enquanto o site web da Associação Europeia de Pesquisa Operacional (Association of European Operational Research Societies - EURO) define como "uma abordagem científica aos problemas de gerenciamento de sistemas complexos. Em um ambiente em rápida mutação e que seu entendimento irá facilitar a decisão e o desenvolvimento de soluções mais eficazes que, normalmente, pode envolver interações complexas entre as pessoas, materiais e dinheiro" ([SPERANZA, 2012](#)); Hillier ([HILLIER; LIEBERMAN, 2001](#))

define como a área de pesquisa em que busca utilizar métodos analíticos avançados para obter melhores resultados em sistemas de decisões.

Desta forma, PO utiliza conhecimentos de diversas áreas como: engenharia, gerenciamento, matemática e filosofia que em conjunto contribuem para uma maior abrangência do escopo da solução que poderá ser definida ao analisar o problema. Atualmente existem diversas variações da PO em variadas áreas da pesquisa de ciências de decisões, podendo assim ser encontrado pelo nome de Ciência de Gerenciamento (*Management Science*) e Engenharia Industrial (*Industrial Engineering*), entre outros nomes.

Historicamente, a PO surgiu como área de estudo durante a segunda guerra mundial com o intuito de conduzir e coordenar as operações (atividades) de uma organização. Anteriormente, existiam estudos isolados sobre como obter melhores decisões, mas foi durante a guerra que houve a necessidade de um estudo mais sistemático que permitisse a replicação mais facilmente em diversos outros casos. Durante a guerra, os militares necessitavam descobrir qual a melhor forma de alocar os recursos que estavam escassos para as diversas operações militares que estavam ocorrendo, como o transporte de mantimentos e armamento, entre outras atividades complexas. Foi quando diversos cientistas foram convocados para criar um modelo matemático que auxiliasse estas decisões.

Após refinarem esta versão inicial do que viria a ser PO, seis fases foram definidas: (i) definição do problema e levantamento de informações; (ii) formulação de um modelo matemático; (iii) derivar possíveis soluções matemáticas deste modelo; (iv) teste do modelo e refinar caso necessário; (v) aplicação do modelo; e (vi) implementação. Esta primeira fase é de grande importância por ser o primeiro contato do analista com o problema do mundo real. Nesta etapa, o problema deverá ser definido e todas as informações do problema devem ser coletadas, como restrições, relações, objetivos, restrições temporais para tomada de algumas decisões, entre outras. O ponto-chave desta etapa é que o analista está lidando com uma descrição do problema a partir de informações incompletas e uma análise aprofundada se faz necessários, quanto mais dados forem descobertos e utilizados, mais fácil se dará o processo de entendimento do problema por outros membros da equipe.

Um exemplo de problema solucionado por PO é referente a um estudo que ocorreu no Departamento de Polícia de São Francisco buscando aperfeiçoar o escalonamento de oficiais nas patrulhas (TAYLOR; HUXLEY, 1989). O sistema computadorizado desenvolvido chegou a reduzir US\$11 milhões anuais, tendo como princípio três itens: (i) manter um nível alto de segurança para os cidadãos; (ii) manter um nível alto de bem-estar dos oficiais; e (iii) minimizar o custo das operações. A partir destas três diretivas, as informações relevantes do processo corrente de escalonamento de policiais foram catalogadas, como também informações que poderiam representar o nível de segurança que o departamento de polícia e o governo gostariam, para finalmente o problema ser especificado.

Após esta especificação, este estudo passou para a próxima etapa da PO, a formulação de um modelo matemático. Modelos matemáticos possuem algumas vantagens quando comparado a

descrições verbais, por descreverem mais facilmente relações de causa e efeito, mostrando quais dados são mais relevantes para a análise, como também por facilitarem a busca por soluções ótimas, como maximização ou minimização de determinados resultados. Primeiramente as variáveis de decisão devem ser identificadas, que são as variáveis que se deseja encontrar os valores da solução, e então uma função objetiva deve representar o problema. Esta função matemática será o modelo do problema e a solução, neste caso do estudo com o Departamento de Polícia, será a minimização dos custos sem reduzir o grau de segurança da cidade. Para este caso, a solução buscada será minimizar o modelo matemático que é visto na equação a seguir, onde: (i) U_{ij} refere-se à escassez de oficiais em uma determinada hora i de um dia j ; (ii) B representa o grau de importância da gestão em relação a escassez de oficiais; e (iii) α sendo um fator de equilíbrio entre a escassez total e a escassez necessária para manter o nível desejado de segurança. Esta função levando em consideração a quantidade de oficiais que estejam na ativa no determinado horário do dia.

$$\sum U_{ij}^{\beta} + \alpha \sum U_{ij}$$

A partir deste modelo, inicia-se a proposta de um procedimento (geralmente computacional) para buscar possíveis soluções ótimas que solucionem o problema em questão. Quanto maior for o escopo do problema, mais dados e mais relações deverão ser especificadas no modelo, também o tornando maior. Desta forma, quanto maior for o modelo definido, maior a possibilidade de existência de falhas neste. Para minimizar a prolongação destas falhas ao longo do processo de definição e desenvolvimento da solução tem-se a quarta etapa do processo de PO, onde serão realizados testes no procedimento verificando se existem falhas nas suas relações e/ou dados que não foram incluídos no modelo do problema. Estas falhas são inevitáveis devido à dificuldade em captar todas as necessidades do problema e transportá-lo para um modelo. A quinta etapa é responsável por codificar e preparar este procedimento para ser executado, tendo a última etapa como a utilização do procedimento para encontrar a solução ótima do problema inicial.

Como visto, as duas primeiras etapas são de grande importância para o processo como um todo. A compreensão do problema e recolhimento das informações relevantes para o problema permite a definição de um modelo matemático que consiga realmente representá-lo. Com este modelo definido é possível encontrar soluções para o problema; porém, caso o modelo esteja incompleto ou errado as soluções encontradas não serão úteis ao problema que se procurava uma solução. Tendo definido corretamente o modelo matemático PO irá permitir ao analista: (i) tomar melhores decisões; (ii) melhorar o planejamento e sua precisão; e (iii) analisar melhor a utilização dos recursos e construir sistemas mais produtivos. Estas melhorias serão baseadas em: (i) dados mais completos; (ii) considerações de todas as opções existentes; (iii) predição cautelosa, levando em consideração o risco; e (iv) as últimas técnicas e ferramentas de decisão. Para auxiliar esta análise, também se tem a utilização de técnicas analíticas, como simulação,

otimização e conceitos de probabilidade e estatística. Com a expansão da utilização de PO, surgiram algumas categorias de problemas conhecidos e possíveis estilos de soluções-padrão para cada uma destas categorias, permitindo que o analista após fazer a modelagem do problema e encaixá-lo em alguma destas categorias, visualize possíveis soluções de maneira mais rápida.

Desta forma, similarmente a ES, a especificação do problema e a sua modelagem em uma linguagem que reduza as ambiguidades permite que o problema seja mais bem compreendido, facilitando a busca pela solução. Principalmente no caso da PO, o modelo do problema também é um artefato de alta importância para buscar uma solução ótima.

3.2.3 Em Engenharia do Conhecimento

No início da década de 1980, o desenvolvimento de um sistema baseado em conhecimento (*Knowledge Based System - Sistema baseado em conhecimento (KBS)*) era visto apenas como a transferência do conhecimento humano para uma base de conhecimento que seria construída. Isto, devido a teoria de que o conhecimento necessário para o KBS já existia e era de propriedade do especialista e deveria apenas ser coletado e a partir daí construir o conhecimento a ser utilizado pelo sistema. Porém, o desenvolvimento de KBS era desenvolvido de forma arbitrária e, igualmente ao desenvolvimento de software convencional, estava preste a entrar em crise (crise do software ocorrida na década de 1970 que motivou o surgimento da ES), devido à perda de prazos na entrega e o alto custo no desenvolvimento do sistema. Foi quando, seguindo o movimento ocorrido na Computação Convencional, técnicas de Engenharia foram inseridas na área de Aquisição de Conhecimento buscando tornar o processo mais disciplinado, definindo métodos, linguagens e ferramentas específicas para o desenvolvimento de KBS.

Posteriormente, estes métodos de aquisição de conhecimento passaram a ser vistos como uma tarefa de modelagem do conhecimento do especialista em um modelo computacional capaz de solucionar problemas específicos, não mais a construção do conhecimento a partir do conhecimento humano. Porém, não necessariamente modelar a forma cognitiva que o especialista tomava suas decisões, mas definir este modelo computacional capaz de obter resultados similares aos obtidos pelo especialista (STUDER; BENJAMINS; FENSEL, 1998). Estes processos para definição de um modelo computacional capaz de solucionar problemas baseados em conhecimento são de certa forma, cíclicos, permitindo o refinamento e inserção de novos dados a partir de novas observações do mundo real realizadas.

Ainda na década de 1980 Clancey realizou um estudo (CLANCEY, 1985) analisando diversos sistemas especialistas desenvolvidos para solucionar diferentes problemas. Apesar de estes sistemas estarem representados de diversas formas, utilizando técnicas diferentes, Clancey conseguiu abstrair um comportamento comum a estes sistemas especialistas. Este padrão foi denominado como Classificação Heurística, que é responsável por descrever de forma abstrata e genérica o problema, sendo assim possível a reutilização destas especificações abstratas para a solução de diversos outros problemas. Isto passa a ser possível, devido ao alto nível de abstração

inserido na especificação destes sistemas especialistas, mas que devem ser especificado com as informações do problema para cada uso realizado.

Outra iniciativa surgiu a partir de uma parceria entre as universidades do Reino Unido e dos Países Baixos. Esta parceria propôs um projeto piloto denominado Project 12 (P12), no qual o principal objetivo era o desenvolvimento de um método baseado em técnicas de engenharia que permitiria a extração e representação do conhecimento do especialista do sistema. Esta proposta foi concretização das modificações ocorridas no desenvolvimento de KBS na tentativa de minimizar os problemas existentes no seu desenvolvimento. Na verdade, este projeto P12 resultou em (BREUKER; WIELINGA, 1984, 1983; BREUKER et al., 1984): (i) uma análise sistêmica de técnicas de elicitación de conhecimento; (ii) uma primeira tentativa de formalizar modelos em nível de conhecimento do especialista; (iii) alguns casos de estudo; e (iv) um sistema computadorizado que auxiliou a análise e documentação do conhecimento. Porém, nenhum destes resultados foi capaz de chegar a um nível de formalismo que permitisse a sua utilização em ambiente corporativo. Este projeto ficou conhecido inicialmente como KADS - *Knowledge Analysis and Documentation System* (Análise de Conhecimento e Sistema de Documentação); e posteriormente *Knowledge Analysis and Design Support* (Análise de Conhecimento e Suporte à Design).

Uma das principais contribuições de KADS foi à representação do conhecimento do especialista. Esta representação é dividida em três camadas: (i) a camada do domínio, em que todo o conhecimento necessário do domínio deverá estar modelado, permitindo inclusive a reutilização deste modelo para solucionar diferentes tarefas; (ii) a camada de inferência, em que o processo de raciocínio do KBS será especificado, incluindo suas ações e regras; e (iii) a camada das tarefas, irá prover uma decomposição das tarefas em subtarefas e suas ações de inferências incluindo a especificação de suas metas. Esta separação do que seria o conhecimento específico do domínio e de como seria a inferência e tarefas do sistema solução, permitiu a KADS concretizar a ideia de reutilização de KBS. Este reuso pode ser realizado em dois níveis: (i) a descrição do domínio poderá ser reutilizada para solucionar tarefas diferentes de outros KBS; e (ii) um KBS poderá ser utilizado em outro domínio, permitindo uma nova visão da solução. Desta forma, a ideia de reuso da descrição do domínio permitiu a construção de uma coleção de modelos de problemas, onde cada modelo representaria os aspectos abstratos do KBS e do ambiente em que estará imerso. Um exemplo de modelo de inferência pode ser visto na Figura 3.5.

Porém, a falta de formalismo no processo de utilização e na forma de modelar o conhecimento, em conjunto ao fato do método não cobrir todo o ciclo de desenvolvimento de um sistema baseado em conhecimento, fez com que a primeira versão de KADS não obtivesse um sucesso duradouro. No entanto, outros projetos surgiram baseados no projeto P12 original, como o caso do CommomKADS. Diferentemente de outras abordagens de engenharia do conhecimento, CommomKADS provê uma ligação mais fácil com os conceitos modernos de OO, o que permite uma assimilação mais fácil do analista. Outro fator que permite facilitar a compreensão do

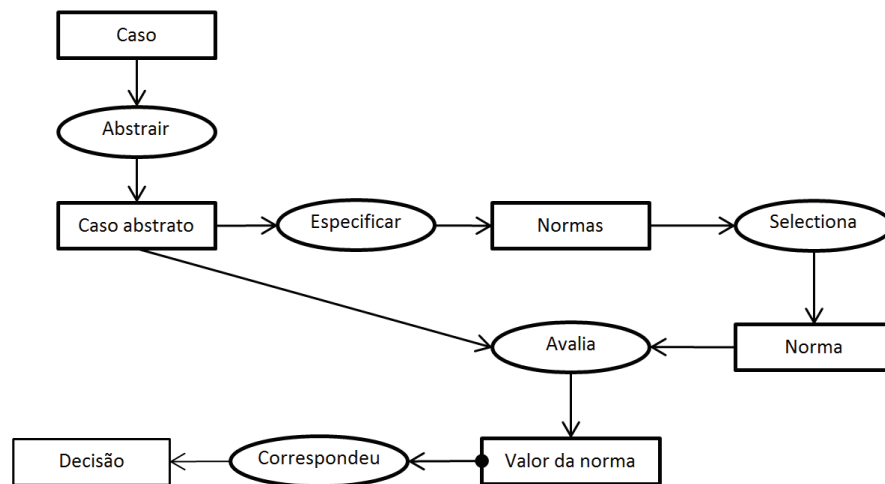


Figura 3.5: Modelo de inferência para o problema de avaliação utilizados por KADS.

método é a utilização de notações compatíveis com UML, permitindo que uma biblioteca de componentes reusáveis e configuráveis fosse proposta e que podem ser utilizados para construir um modelo especialista para um problema de maneira mais fácil (BREUKER; VELDE, 1994), de forma similar a biblioteca de padrões de projeto existentes em ES.

Em CommomKADS podem ser encontrados seis modelos para níveis diferentes de abstração do KBS: (i) o modelo da organização; (ii) o modelo de tarefa; (iii) o modelo do agente; (iv) o modelo de comunicação; (v) o modelo de competências; e (vi) o modelo de arquitetura. Onde os quatro primeiros modelos são focados na especificação e modelagem do ambiente organizacional no qual o KBS estará imerso. Detalhadamente, o modelo organizacional descreve as funcionalidades realizadas por cada unidade organizacional, como também as deficiências existentes atualmente. O modelo de tarefa especifica uma descrição hierárquica das tarefas que devem ser realizadas. Já o modelo do agente irá descrever as responsabilidades de cada agente envolvido na execução de cada tarefa. O último modelo responsável pela representação do ambiente, o modelo de comunicação, irá detalhar as diversas interações entre os agentes especificados. Já os modelos de competências e de arquitetura irão descrever aspectos funcionais e não funcionais do KBS em desenvolvimento.

Tanto para KADS como para CommomKADS a separação da especificação do problema para a definição dos modelos de inferência do conhecimento e das tarefas permitiu a criação de uma biblioteca de especificações de problemas. Desta forma, por serem de forma abstrata o suficiente para não conter informações específicas do problema, estes modelos passaram a permitir a sua reutilização, bastando apenas realizar uma instanciação do modelo o adaptando ao problema específico. Com isto, a identificação de um problema entre um destes modelos permite facilitar o processo de especificação da solução, já que também é possível encontrar possíveis soluções padrão para determinados modelos de problemas.

3.3 O campo dos SMA

Como já foi apresentado, a definição de soluções que utilizem IA não é uma tarefa fácil. Existem inúmeros fatores que devem ser analisados, os quais não são modelados facilmente, o que torna estes problemas de difícil compreensão, impedindo a especificação de uma solução de forma imediata.

Em um mundo ideal, um método seria utilizado para permitir uma especificação detalhada do problema a ser solucionado, porém no mundo real diversos projetos tem a análise do problema a ser solucionado de forma *ad hoc*. Porém, existem áreas em que uma análise é de fato realizada de forma criteriosa, por exigências da própria solução, como por exemplo soluções que procuram simular de forma mais real possível o problema solucionado, alguns exemplos destes problemas podem ser encontrados em diversas situações, iniciando em alguns jogos (conhecidos como jogos sérios (MICHAEL; CHEN, 2005)) à ferramentas militares (JADON; SINGHAL; DAWN, 2014).

Atualmente existem alguns métodos para auxiliar a especificação e o desenvolvimento de um SMA, mas focam principalmente na solução, deixando muitas vezes a etapa inicial da compreensão do problema sem ser abordada de forma tão aprofundada quando poderia. Desta forma, alguns destes métodos partem do pressuposto que o problema já está bem compreendido e já existe uma ideia inicial de uma provável solução. Apenas alguns poucos métodos procuram guiar o analista na fase inicial de identificação, análise e compreensão do problema; mas mesmo nestes casos, esta etapa ainda é realizada de forma superficial.

A seguir serão apresentados alguns dos métodos utilizados no campo da pesquisa em SMA e que basicamente foram originados a partir dos conhecimentos já existentes no campo da Engenharia de Software e do campo da Engenharia de Conhecimento (IGLESIAS; GARRIJO; GONZALEZ, 1999; CERNUZZI; COSSENTINO; ZAMBONELLI, 2005). Da ES, alguns conceitos do paradigma Orientado a Objeto foi transportado para o campo dos agentes devido a algumas similaridades entre agentes e objetos (CERNUZZI; COSSENTINO; ZAMBONELLI, 2005; WEBER; MATEAS, 2009). Como por exemplo, Shoham (SHOHAM, 1993) indica que um agente pode ser considerado como um objeto de forma ativa, possuindo um estado mental de conhecimento. Como também a comunicação, enquanto em OO os objetos se comunicam através de trocas de mensagens, como também ocorre com agentes em SMA, a diferença nesta comunicação se dá pelo fato de um agente fazer esta troca de mensagens a partir de seu conhecimento adquirido, sendo possível analisar se uma determinada mensagem será ou não benéfica; enquanto os objetos realizam esta comunicação de uma forma muito mais mecânica, não havendo avaliações (IGLESIAS; GARRIJO; GONZALEZ, 1999). No entanto, existem diferenças marcantes entre objetos e agentes, Wooldridge (WOOLDRIDGE; JENNINGS; KINNY, 2000) relata que a grande diferença entre objetos e agentes é relacionada ao forte encapsulamento que este segundo possui. O estado interno de um agente é mais incerto do que o de um objeto e, em alguns sistemas, o comportamento de um agente frente a uma

solicitação recebida nem sempre será conhecida.

Enquanto os métodos derivados da Engenharia do Conhecimento possuem como grande vantagem a utilização dos conceitos da área para a representação do conhecimento de um agente, já que esta é uma das principais características de um SMA (IGLESIAS; GARRIJO; GONZALEZ, 1999). No entanto, estes métodos originados a partir da EC geralmente concebem um sistema centralizado, sendo então difícil a construção de um SMA (IGLESIAS; GARRIJO; GONZALEZ, 1999). Quando comparadas aos métodos originados das existentes em ES, estes métodos baseados em EC são menos extensíveis além de não permitirem uma melhor modelagem da interação entre agentes e do próprio sistema como um todo. Como também devido a falta de disseminação das técnicas de EC (IGLESIAS; GARRIJO; GONZALEZ, 1999) a utilização destes métodos precisarão de mais tempo no seu aprendizado, devido a utilizar conceitos que não estão no dia a dia da equipe de desenvolvimento.

3.3.1 Gaia

Gaia é um método para SMA que tem como objetivo permitir que o analista conseguisse detalhar o design do SMA a partir de uma lista de requisitos, este nível de detalhe apresentado pelo design do sistema deverá permitir que este fosse desenvolvido diretamente, sem a necessidade de mais análises (WOOLDRIDGE, 2009). Para isto, procura utilizar conceitos já comumente empregados pelos métodos para ES, tornando mais fácil a compreensão pela equipe que ainda não esteja acostumada com SMA. Porém, Gaia não é uma simples tentativa de adaptar as técnicas de ES, algumas alterações foram realizadas para melhor representar um sistema baseado em Agentes Inteligentes.

Este método procura a cada passo realizado diminuir a abstração do conhecimento do sistema, tornando os conceitos mais concretos. Para isto, Gaia pode ser dividida em duas partes: (i) Análise; e (ii) Design. Durante a etapa de análise o processo real é detalhado utilizando conceitos mais abstratos, que não necessariamente, terão alguma relação com a implementação. Neste detalhamento, procura-se construir um entendimento geral do problema e da sua estrutura, identificando informações como: regras, permissões, responsabilidades, protocolos, atividades e propriedades de segurança.

Já durante o design, Gaia foca na especificação concretização dos conceitos, identificando o modelo dos agentes, serviços e como ocorre a aquisição das informações pelos agentes e suas interações. Este design deverá ter um grande nível de detalhes em baixo nível, já representado partes do sistema a ser codificado, permitindo que o SMA seja codificado sem a necessidade de mais informações.

Levando em consideração o exemplo de controle da IA de um jogo RTS, tem-se o primeiro passo da etapa da análise como a identificação de quais agentes pertenceriam a organização do problema (3.6), como por exemplo, agentes para a coleta de recursos, gerenciamento das unidades de ataque e defesa, entre outros. Porém, a grande questão referente a utilização de

Gaia seria com relação à separação entre o que de fato seria problema e o que já seria solução. Agentes só existem na solução, olhando para o problema, ainda não é possível identificar o que seriam os agentes responsáveis por cada atividade ou problema a ser solucionado. Sendo esta então, uma característica predominante para excluir Gaia para a especificação do problema, pois esta já começa a sua análise pensando em como estará a disposição dos agentes; que, pela visão do autor, estão presentes apenas na solução e não no problema.

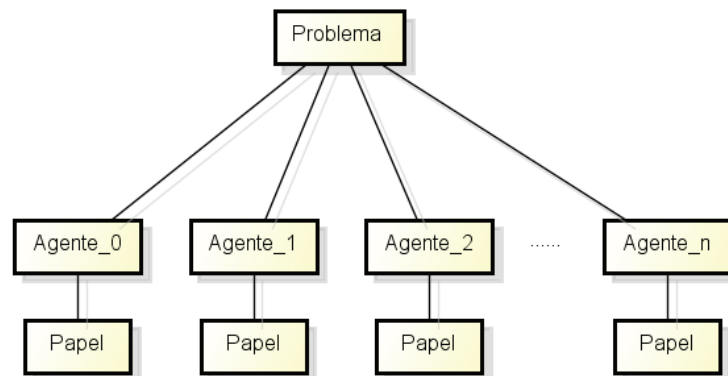


Figura 3.6: Exemplo da estrutura resultante da etapa de análise de gaia

3.3.2 MaSE

Outro método que derivou dos conceitos de ES é a *Multi-Agent System Engineering (MaSE)* (DELOACH, 2001). Este método procura guiar todo o processo de desenvolvimento de um SMA, partindo da sua caracterização até a sua implementação. Também dividida em duas fases com nomes idênticos às de Gaia: (i) Análise, que é subdividido em três etapas: identificação das metas, casos de uso (similar a UML) e refinamento das regras; e (ii) Design, subdividida em quatro etapas: diagrama de agentes que descreve os agentes e suas regras, conversação que mapeia (através de um diagrama de comunicação) a troca de informações entre agentes, arquitetura composta por suas classes, e o diagrama de *deployment*.

A primeira etapa da fase de análise de MaSE é identificação das metas, em que a partir de uma especificação inicial do sistema o analista deverá identificar um conjunto estruturado de metas que o sistema possui. Esta lista deverá ser apresentada a partir de um diagrama de metas hierárquico, como mostra a Figura 3.7. A próxima etapa seria a de construir os casos de uso, usando uma representação similar à utilizada com UML. Nesta etapa os casos de uso do sistema devem ser identificados a partir dos requisitos iniciais do sistema, os estruturando em um diagrama de sequência. Finalizando a fase de análise, a terceira etapa é realizada, onde o analista deverá transformar este diagrama hierárquico de metas em um conjunto de metas. Estas metas irão representar um bloco do SMA, sendo utilizado para definir as classes do agente. Mesmo com existindo esta fase de análise, MaSE também foca mais na descrição e elicitação dos requisitos que a solução estará baseada, muitas vezes não existindo realmente uma compreensão

do problema a ser abordado.

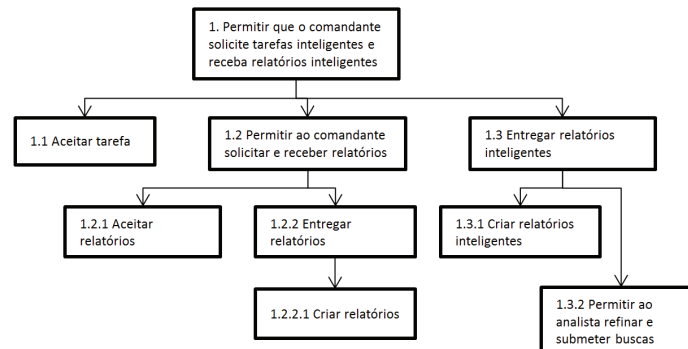


Figura 3.7: Diagrama hierárquico de metas. (Retirado e modificado a partir de (DELOACH, 2001))

Durante a fase de Design, a arquitetura do sistema a ser desenvolvido começa a tomar forma e MaSE passa a guiar o processo de desenvolvimento mais focado na codificação do sistema. A primeira etapa desta fase procura criar um diagrama de classe dos agentes, identificando quais classes deverão ser codificadas e suas comunicações. A etapa de conversação procura identificar o protocolo de comunicação existente entre os agentes. Na penúltima fase, que é muito ligada com a fase anterior e é benéfico que exista uma alternância entre estas duas etapas. Robinson (ROBINSON, 2000) descreve a construção destes agentes de forma similar a arquitetura de software baseada em componentes. A última etapa da fase de design e do método MaSE realiza a instanciação dos agentes no sistema e verifica o comportamento através de um diagrama de *deployment*.

Voltando ao exemplo da IA para controlar um jogo RTS, a primeira etapa de MaSE seria a identificação das metas deste problema, que poderia ser coletar recursos, atacar o adversário, proteger o centro de comando, produzir novas unidades e novas edificações. Posteriormente seria definido um diagrama hierárquico de acordo com a importância de cada uma destas metas e em sequência, seria definido os casos de uso de cada uma destas metas, definindo como o sistema deverá se comportar, podendo utilizar um diagrama de sequência. Ou seja, novamente já partindo para identificar como a solução deverá ser estruturada, não conseguindo distinguir a barreira entre o que seria problema e o que seria solução.

3.3.3 Tropos

Tropos (BRESCIANI et al., 2004) é um método que utiliza diversos conceitos da ES, mas que procura inserir diversos outros das áreas de Agentes e também da Engenharia do Conhecimento. Como outros métodos, Tropos utiliza a ideia de metas da área de agentes, enquanto utiliza a engenharia do conhecimento para auxiliar na forma de representar o conhecimento que o agente irá possuir. Já da ES, Tropos utiliza a subárea da Engenharia de Requisitos para permitir ao analista descrever com mais clareza o que de fato será o sistema a ser desenvolvido. Para

a aplicação do método existem cinco fases: (i) requisitos iniciais; (ii) requisitos tardios; (iii) design arquitetural; (iv) design detalhado; e (v) implementação.

A primeira fase de Tropos, denominada análise prévia de requisitos, é a única fase dos métodos apresentadas que despende algum tempo na análise e detalhamento do problema a ser abordado. Nesta fase, o ambiente em que o problema está inserido deverá ser descrito, listando dados importantes deste ambiente, como: *stakeholders* do ambiente, suas intenções que posteriormente deverão ser decompostas em nas metas e planos do sistema. Para este fim, é possível utilizar algumas perguntas como (MORANDINI et al., 2008; GIORGINI; MYLOPOULOS; SEBASTIANI, 2005): "quais são os *stakeholders* do domínio?", "quais são suas metas e como elas estão relacionadas?". Como resultado, esta primeira fase deverá gerar uma lista de *stakeholders*, suas metas e um modelo representativo destes dados. A segunda fase identifica, a partir dos dados gerados na fase anterior, quais serão realmente os requisitos do sistema a ser desenvolvido. Já na terceira fase, os atores do sistema já são descritos em detalhes, incluindo a especificação da comunicação existente. Enquanto na última fase, o sistema deverá ser codificado.

Mesmo tendo uma fase específica para levantamento dos requisitos do ambiente, Tropos faz uma análise do problema apenas de forma superficial, limitando-se ao entendimento da organização e suas primitivas. Tropos não procura se aprofundar à compreensão do problema abordado, não identificando pedaços menores do problema que permitiriam uma melhor compreensão deste e consequentemente o desenvolvimento de uma solução de forma mais fácil.

Tropos segue o mesmo caminho de Gaia, onde na primeira fase já procura definir quais seriam os atores e suas metas que estariam envolvidos no sistema, como mostra a Figura 3.8, em que está representado um problema a partir da análise realizada em *Early Requirements*. Mais uma vez olhando o caso da IA para um jogo RTS, tem-se que os trabalhadores poderiam ser um dos atores e como meta seria a coleta de recursos e construção de edificações quando solicitado. Porém, estas informações já estariam mais relacionadas à solução do que ao problema de fato, tornando Tropos um método que também não consegue lidar corretamente com o problema, isolando-o da solução.

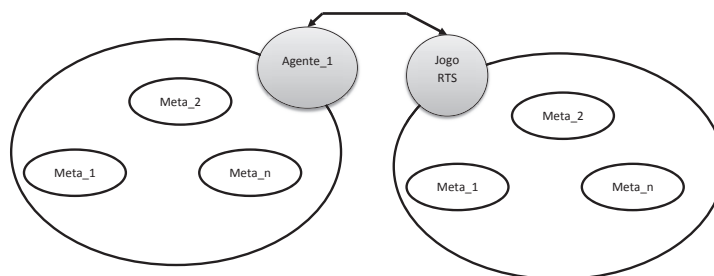


Figura 3.8: Exemplo da estrutura de *Early Requirements* de um problema modelado com Tropos

3.3.4 SADAAM

Da mesma forma que os métodos para ES foram modificadas ao passar do tempo para atender demandas de desenvolvimento mais ágeis, os métodos para SMA também seguiram esta tendência. Com isto, Surgiram alguns métodos mais novos que procuraram eliminar a rigidez de métodos mais antigos que eram mais focados na produção da documentação do SMA do que no próprio desenvolvimento. Entre estes métodos é possível citar SADAAM (CLYNCH; COLLIER, 2007), um método que utiliza técnicas derivadas dos métodos ágeis, incluindo as encontradas no Manifesto Ágil (MARTIN, 2003; FOWLER; HIGHSMITH, 2001), SCRUM (SCHWABER, 2004), XP (BECK, 2000), TDD (BECK, 2003; JANZEN; SAIEDIAN, 2005) e *Refactoring* (PRESSMAN, 2001).

O núcleo do método SADAAM é denominado de Processo de Desenvolvimento de Agentes (*Agent Development Process (ADP)*) que é composto por três fases: (i) design; (ii) implementação guiada por testes (*Test-Driven Implementation (TDI)*); e (iii) entrega e revisão. Previamente a este núcleo existe a fase de requisitos, mas SADAAM opta por tratar esta fase de forma minimalista por esta ser reconhecidamente uma etapa que demanda muito tempo e na visão de Amor (AMOR; FUENTES; VALLECILLO, 2005) ser um grande desperdício de tempo do projeto, chegando a consumir aproximadamente 64% do tempo total para especificação. Após a ADP existe ainda a fase final de finalização do projeto. Porém, existem diversos outros estudos na área de ES que o tempo gasto durante a especificação reduz possíveis erros durante o processo de desenvolvimento (BOOCH, 1996; SOMMERVILLE, 1995), o que resultaria em mais tempo gasto no retrabalho e correção do software desenvolvido.

A fase de Design, que é a primeira fase do método SADAAM, provê um processo iterativo e incremental de análise e design dos agentes autônomos, tendo como principal objetivo a tradução dos requisitos em decisões de arquitetura. A próxima fase, TDI, aplica técnicas ágeis para dar suporte na criação, teste e codificação dos agentes. Enquanto a última fase, Entrega e Revisão, lida com a entrega do código para o cliente, permitindo o cliente testar o produto solicitado. Uma das principais técnicas ágeis inseridas em SADAAM é o incentivo a realizar este processo ADP em iterações curtas e incrementais, permitindo ao cliente participar da evolução do projeto.

3.3.5 Beast

Beast (CARRERA BARROSO; SOLITARIO; IGLESIAS FERNANDEZ, ????) é um método que foi proposto na tentativa de minimizar os problemas de comunicação entre *stakeholders* e a coleta de requisitos iniciais do sistema a ser desenvolvido. Com este intuito, Beast utiliza algumas técnicas ágeis como iterações curtas e um maior contato com o cliente, procurando assim facilitar o desenvolvimento e compreensão do sistema.

Este método também foca na rastreabilidade dos requisitos, tornando cada requisito um teste de aceitação, tendo como principal benefício uma melhor compreensão dos requisitos do

sistema. Porém, a aquisição e detalhamento dos requisitos foram adaptados do Desenvolvimento Baseado em Comportamento (*Behavior Driven Development - Desenvolvimento baseado em comportamento (BDD)*), lidando os problemas que envolvam SMA como sendo qualquer outro sistema de software. Desta forma, não aborda algumas características que permitem facilitar o desenvolvimento dos agentes do problema.

Beast consiste em quatro fases: (i) especificação do comportamento do sistema; (ii) especificação do comportamento do SMA; (iii) teste a nível dos agentes; e (iv) teste do SMA. Na primeira fase, é utilizada a técnica BDD para a descrição do comportamento do sistema e facilitar a comunicação entre os clientes do produto e a equipe de desenvolvimento, utilizando para isto o conceito e a definição das histórias presentes no problema. A segunda fase utiliza o resultado da fase anterior para detalhar a arquitetura do SMA, podendo inserir, remover ou modificar comportamentos já identificados. A fase seguinte foca na identificação dos agentes e o posterior desenvolvimento destes, podendo utilizar técnicas de AOSE. A última fase, de teste do SMA que procura validar o sistema com todos os agentes interagindo e também avaliar comportamentos emergentes que possam vir a surgir.

Beast faz uma mescla das informações entre o problema e a solução, já que este define as histórias já pensando em como será desenvolvida a solução. Ou seja, não coleta apenas as informações do problema, mas já procura definir como cada história deverá ser solucionada. No caso de um jogo RTS, uma possível história seria a coleta de recursos, e além da definição da história, também estaria presente a especificação do que e como esta história seria desenvolvida.

3.3.6 MAS-CommonKADS

MAS-CommonKADS (IGLESIAS et al., 1998) é um dos métodos mais conhecidos dos que foram derivadas da EC, este método entende o padrão CommonKADS e insere alguns conceitos dos métodos da ES, como técnicas da modelagem e responsabilidades existentes no paradigma da orientação a objetos. Como também foram utilizadas técnicas para melhorar a formalização da passagem de mensagens, como a inserção de linguagens que permitam representar e especificar melhor esta comunicação. Estas modificações foram realizadas com o intuito de permitir uma melhor representação da comunicação entre agentes, tornando mais claro este processo de especificação de um SMA, minimizando possíveis ambiguidades durante a fase de desenvolvimento.

O método é composto basicamente por três fases: (i) conceituação; (ii) análise; e (iii) design. No decorrer das duas últimas fases, sete modelos serão produzidos, representando diversas entidades do sistema e seus relacionamentos. Em conjunto, estes modelos produzidos irão caracterizar a solução a ser desenvolvida.

A primeira fase, conceituação, utiliza uma abordagem centrada no usuário para identificar e documentar alguns casos de uso do sistema e assim ter uma representação da primeira descrição do sistema em alto nível, partindo da visão do usuário. Porém esta visão inicial é bastante

superficial, sem coletar dados suficientes que permitam uma visão mais abrangente do problema pelo analista.

A fase seguinte, análise, procura traduzir estes casos de uso em requisitos do sistema. Para isto, os primeiros modelos do método são desenvolvidos sendo guiados pelo risco proporcionado. Iniciando pelo modelo de agente que será responsável pela tarefa; modelo da tarefa que a decompõe em metas e dados da tarefa; modelo de coordenação que procura descrever as interações e protocolos de coordenação entre os agentes; modelo de conhecimento para representar o conhecimento do domínio, do agente, e do ambiente atrelados à tarefa; e modelo da organização procurando avaliar o impacto da utilização de um SMA no ambiente da organização. A fase de design recebe como entrada os modelos de análise construídos na fase anterior e os transforma em especificação da arquitetura do SMA, podendo iniciar a codificação do sistema.

3.3.7 Prometheus

Já o método Prometheus (PADGHAM; WINIKOFF, 2002) parte da ideia da representação do conhecimento, portanto conceitualmente facilita a representação do conhecimento de cada ator, incluindo suas metas e planos. O método é focado na geração de entregáveis ao longo do processo de especificação e desenvolvimento do SMA, permitindo assim a sua utilização em ambientes mais ágeis. A aplicação do método é dividido em três fases: (i) Especificação do sistema; (ii) Design da arquitetura; e (iii) Design detalhado.

A primeira fase de especificação do sistema procura detalhar como o SMA irá interagir com o ambiente em que estará imerso, identificando as percepções e as ações do SMA. Em paralelo, o analista deverá iniciar a descrição de quais atividades o SMA deverá realizar, olhando de forma mais genérica, quais as funcionalidades do SMA, dando uma visão de qual será o propósito do sistema. Porém, esta fase praticamente não lida com o detalhamento e compreensão do problema a ser solucionado, focando apenas na interação do SMA e o ambiente, buscando assim uma razão para a existência do sistema.

A próxima fase, design da arquitetura, irá identificar quais agentes existirão no SMA e por quais atividades cada agente será responsável. Este processo é avaliado de acordo com critérios de coerência e acoplamento existentes em ES. A última fase de detalhamento do design da arquitetura é quando é descrito a estrutura de cada agente e como estes irão realizar suas tarefas para alcançar seus objetivos. Também é nesta fase em que os dados dos agentes devem ser especificados, como crenças, desejos, e intenções, seguindo com a definição das capacidades, eventos internos e planos, finalmente seguindo para a codificação destes agentes, obtendo o SMA final.

Desta forma, Prometheus na sua fase de especificação do sistema tenta apenas listar as propriedades existentes no problema, porém já pensando nos agentes e como estes deverão se comportar. Sendo assim, esta especificação inicial já é muito próxima da descrição da solução, restando para as próximas etapas de Prometheus a organização dos agentes e quais atividades cada

um estaria responsável. No caso do exemplo de um jogo RTS, Prometheus tentaria já identificar quais agentes estariam presentes na solução e quais seriam suas características, não definindo corretamente o que seria o problema e sim já partindo para a solução.

3.4 Por que algo novo

Como visto, os métodos apresentados que lidam com o desenvolvimento de SMA não lidam com a importância que deveria a etapa inicial de compreensão do problema a ser solucionado, mas também não apresentam capacidade suficiente de lidar com a imprevisibilidade e estado parcial do ambiente presente nos problemas MAMT. Esta lacuna é exatamente o que o método Icelus proposto ao longo do restante desta tese pretende solucionar. Porém, Icelus é baseado em métodos e técnicas já existentes e utilizadas em outros campos, como também foram apresentadas, mas não é apenas uma aplicação direta de algo já existente.

A aplicação direta de algum método ou técnica existente no campo da ES, por exemplo, nos problemas de SMA, principalmente os problemas MAMT, é uma atividade que deve ser evitada. Isto se dá devido a grande diferença entre o que é compreender um problema em ES e o que é de fato compreender um problema a ser solucionado por um SMA. Como explicado no capítulo anterior, um problema de ES lida com conceitos muito mais reais e palpáveis, enquanto um problema que necessita de uma solução inteligente, irá ter a necessidade outras informações, como captar a essência das relações entre as entidades envolvidas e o ambiente, como também captar o comportamento que a solução deverá modelar, desta forma lidando com conceitos muito mais incertos e nem sempre possíveis de serem modelados de forma convencional, como ocorre em ES.

Desta forma, a proposta de Icelus tenta apresentar uma nova abordagem à compreensão de problemas a serem solucionados por SMA; uma abordagem adaptada de métodos e técnicas existentes em outros campos, como por exemplo a utilização de Árvore Causal para facilitar a quebra dos problemas mais complexos em subproblemas menores e mais simples de serem compreendidos.

3.5 Conclusão

Neste capítulo foi apresentado como diversos campos lidam com a compreensão de problemas. Partindo do campo conceitual e a teoria que envolve os conceitos de análise e modelagem de problemas, até alguns campos Computacionais. Como por exemplo, o caso da Engenharia de Software e Pesquisa Operacional que procuram lidar com detalhes os problemas a serem abordados por cada um. Enquanto o campo dos SMA não lida corretamente com a compreensão dos problemas, sempre já partindo para pensar e arquitetar a solução do problema.

Desta forma, ao longo deste capítulo uma argumentação foi formada com o intuito de defender a necessidade da definição de um novo método específico à especificação e compreensão

dos problemas. Este método foi nomeado de Icelus e será apresentado nos próximos capítulos em conjunto com os seus resultados já obtidos.

4

A Concepção de Icelus

Este capítulo irá apresentar o método Icelus, apresentando uma visão geral da pesquisa e o que foi buscado alcançar durante o desenvolvimento deste método. Além disto, também será apresentada a motivação, identificando quais foram as dificuldades enfrentadas que terminaram resultando na busca por um método que facilitasse lidar com estas; como também, listar quais são os principais pilares que Icelus se sustenta permitindo seu surgimento.

4.1 Visão Geral

Os problemas que necessitam de soluções que envolvam técnicas de IA geralmente apresentam alguma dificuldade na sua compreensão, muito se dá devido a natureza destes problemas serem diferentes das demais apresentadas em problemas mais convencionais existentes, por exemplo, na ES. Os problemas da ES lidam com conceitos e características tangíveis ao cliente, permitindo que seja mais fácil identificar o que de fato deve ser representado na solução. Porém, os problemas que lidam com soluções em IA lidam com características abstratas, como por exemplo o comportamento que a solução deverá ter e que muitas vezes não é linear, podendo sofrer mudanças drásticas devido a alguma variação do ambiente.

Esta dificuldade é aumentada consideravelmente quando os problemas classificados ao longo desta tese como MAMT são analisados. Este aumento se dá devido a inserção de outros atributos que por si só já são considerados complexos (WEISS, 1999), como: (i) a existência de diversas tarefas que devem ser realizadas para conseguir alcançar o objetivo; e (ii) a existência de diversos agentes para poder, trabalhando em conjunto, alcançar a solução de forma aceitável. O que já inclui outro atributo para aumentar a complexidade do problema, a comunicação entre os agentes existentes e que interagem com o problema. Desta forma, compreender um problema MAMT não é uma atividade corriqueira e a sua descrição é ainda mais importante, pois devido as possíveis variações, a possibilidade de ocorrer ambiguidades é muito grande, possibilitando a ocorrência de inúmeros erros durante o desenvolvimento da solução.

Em consequência, esta tese está propondo o método Icelus com o intuito de facilitar a compreensão de problemas MAMT a partir da utilização no campo dos SMA de técnicas

já aplicadas em outros campos da Computação. Técnicas como árvores causais e captação de requisitos, onde a árvore causal é utilizada para auxiliar o desmembramento do problema em problemas menores e mais simples de serem compreendidos e a elicitación de requisitos é utilizada para extrair do cliente as informações necessariamente suficientes para descrever um subproblema; e para auxiliar esta última etapa, a concepção de Icelus gerou um documento para guiar a extração destas informações.

Porém, Icelus não foi um método que surgiu pronto, ele foi amadurecendo com o tempo e, como consequência, algumas versões do método surgiram ao longo do tempo; atualmente, Icelus encontra-se na sua terceira versão. A evolução das versões de Icelus ocorreram no decorrer da execução de um processo cíclico da aplicação de três etapas: (i) Execução, onde o método é submetido a algum experimento em busca de sua validação; (ii) Análise, onde os resultados da etapa anterior são avaliados buscando verificar se existe alguma possível modificação que possa melhorar o método; e (iii) Modificação, onde a partir da análise realizada na etapa anterior um estudo é realizado buscando encontrar embasamento científico para guiar possíveis mudanças, e estas modificações irão gerar um novo método que deverá ser submetido novamente à este mesmo processo.

Desta forma, cada nova versão de Icelus apresentada foi embasada em modificações identificadas a partir de sua utilização. Seja por verificar que os dados gerados ainda eram incompletos ou porque geravam dados em excesso, com informações já muito próximas da solução, que não devem fazer parte do escopo do método proposto. Como também, algumas modificações foram propostas com o intuito de facilitar sua aplicação, tornando-o mais prático e próximo da equipe de desenvolvimento.

4.2 Motivação

A ideia do método Icelus surgiu a partir da observação da dificuldade apresentada pelos alunos em propor e desenvolver o projeto de fim de semestre de uma disciplina de Agentes Inteligentes do curso de Ciência da Computação na Universidade Federal de Pernambuco. Este projeto consiste em propor e posteriormente codificar um SMA para controlar os exércitos no jogo Starcraft Broodwar ([STARCRAFT, 1998](#)), jogo classificado como RTS, através da interface de programação BroodwarAPI - BWAPI. Entre estas dificuldades, pode-se citar a não compreensão dos alunos em como construir uma linha de raciocínio capaz de englobar todas as atividades que devem ser executadas pelas entidades existentes no jogo, como também, foram presenciadas muitas soluções propostas pelos alunos que não apresentavam um grau aceitável de completude, muitas vezes, apresentando soluções de diversos problemas pontuais, mas que a solução como um todo não conseguia se comunicar corretamente.

Porém, um sistema de controle para um jogo RTS é tipicamente um problema MAMT, onde é possível encontrar inúmeras atividades, como coleta de recursos, defesa do centro de comando, ataque ao centro de comando adversário, exploração do mapa, ataque às unidades

adversárias, entre outras; e que no geral são divididas por diversos agentes que deverão estar trabalhando em conjunto em busca do objetivo comum que é derrotar o adversário. Além desses critérios básicos de um problema MAMT, um jogo RTS ainda possui uma característica que torna este problema ainda mais especial, tudo ocorre em tempo real modificando constantemente o ambiente e sendo necessário uma verificação constante de como o problema será solucionado.

Devido a estas dificuldades, muitos alunos apresentavam uma grande dificuldade inicial na concepção do projeto, não conseguindo lidar com a quantidade de tarefas existentes nem conseguindo muitas vezes montar uma linha de raciocínio de uma possível solução. De fato, os jogos RTS apresentam diversas tarefas e muitas delas necessitam de uma soluções baseada em técnicas de IA dificultando a identificação e coordenação destas tarefas; tarefa como: predição (LAIRD, 2001; WEBER; MATEAS, 2009), planejamento (PÉREZ, 2011), gestão de recursos (CHURCHILL; BURO, 2011) , reconhecimento de padrão (SHARIFI; ZHAO; SZAFRON, 2010), etc.

Os primeiros projetos entregues pelas primeiras turmas apresentavam um alto nível de simplificação, onde trabalhavam apenas com um grupo muito reduzido destas tarefas, primeiro por não conseguirem identificar todas as tarefas existentes e segundo por preferir tratar com apenas as que eles mesmos achavam mais relevantes. Esta redução drástica do escopo do problema inicial foi a forma possível que os alunos encontravam de lidar com esse problema complexo. Porém, muitas vezes tarefas importantes eram deixadas de lado, ou até mesmo nem eram identificadas, demonstrando que os alunos não haviam realmente compreendido a profundidade do problema. Devido a este cenário, alternativas tinham que ser identificadas para permitir que os alunos passassem a enxergar melhor o que de fato compunha o problema, como a identificação de mais tarefas envolvidas e como conseguir lidar com elas de forma coordenada.

A primeira alternativa foi procurar métodos, processos ou técnicas no campo dos SMA que pudessem nortear os alunos em como lidar com estes problemas, mas os casos encontrados partiam direto para o gerenciamento do desenvolvimento da solução, não se aprofundando na compreensão e estruturação do problema a ser solucionado. Sendo então necessário aumentar a área de pesquisa, quando finalmente foram encontrados estudos, como BPM que passou a lidar um pouco melhor com a análise do problema a ser solucionado. Porém, no geral, os problemas eram problemas concretos e mais fáceis de serem descritos, muitas vezes não envolvendo as dificuldades que são encontradas em um problema MAMT. Como por exemplo, as técnicas existentes em BPM não poderem ser aplicadas diretamente em SMA por não estarem preparadas para lidar com as necessidades de Agentes Inteligentes, como a representação do conhecimento dos agentes e sua racionalidade.

Como consequência, foi necessário realizar um estudo a respeito das técnicas existentes em outros campos da Computação e como estas poderiam ser aplicadas em problemas que envolvam IA. Desta forma, foi idealizado um método que permitirá ao time de desenvolvimento (neste caso em específico, os alunos) analisar e compreenderem melhor o problema a serem solucionado, identificando os possíveis subproblemas existentes, suas informações e como estes

se relacionariam.

4.3 Icelus

Ao longo deste estudo, foi identificada a dificuldade em encontrar um método que seja capaz de auxiliar o time de desenvolvimento na análise do problema a ser solucionado. Desta forma, Icelus está sendo proposto justamente na tentativa de atender esta demanda existente mas que não é muito valorada. Assim sendo, a definição de Icelus foi baseada em estudos realizados em como outras áreas da Computação lidam com seus problemas, como por exemplo como a ES lida com a especificação e compreensão dos problemas a serem resolvidos.

Tendo isto em mente, a definição de Icelus resultou em um método que é basicamente estabelecido sobre dois pilares: **(i)** coleta de informações do problema e sua descrição; e **(ii)** expansão do problema. Ambos os pilares procuram se basear em técnicas já utilizadas nestes outros campos da Computação.

4.3.1 Pilares

A própria ES possui diversas técnicas de como lidar com o cliente especialista no problema e desta forma extrair as informações importantes de como representá-lo no campo digital (SOMMERVILLE, 1995). Inúmeras técnicas podem ser utilizadas, o mais importante é estabelecer um canal de comunicação entre o cliente e a equipe de desenvolvimento, para assim fluir a troca de informações, permitindo que a solução reflita da melhor forma o que o cliente deseja (SOMMERVILLE, 1995). Diversas técnicas podem ser utilizadas para coletar os requisitos. Não existe uma regra de qual técnica deverá ser utilizada de acordo com o projeto, na verdade, estará muito mais relacionada a quão maduro é o cliente sobre o projeto e a experiência do time de desenvolvimento. Algumas técnicas podem ser listadas, como (SOMMERVILLE, 1995): (i) entrevistas; (ii) experimentação; (iii) experimentação direcionada, através de casos de uso; (iv) questionários; entre outras.

O mais importante desta etapa é estabelecer a comunicação inicial entre o cliente (stakeholder especialista responsável por conter as informações do problema a ser abordado) e o time de desenvolvimento, podendo ser uma comunicação mais formal ou não, cabendo ao analista identificar qual a melhor forma de lidar com o cliente, tornando-o mais seguro do que ele realmente quer como solução. Desta forma, Icelus procura ter um dos seus pilares em como extrair os requisitos do problema a partir do conhecimento do cliente, para isto incentiva a utilização de entrevistas ou experimentos direcionados com o cliente sejam realizados para facilitar esta comunicação.

Para melhor guiar esta etapa de coleta de informações do problema, um formulário para a captura destes dados foi definido. Este documento, não apenas poderá guiar como a conversa com o especialista ocorrerá, mas também manterá o foco desta conversa inicial na busca por

informações necessárias ao problema. Informações, estas que muitas vezes não são coletadas e que irão resultar horas posteriores de retrabalho.

O segundo ponto importante de Icelus é a forma como expandir o problema, procurando identificar problemas menores e mais fáceis de serem compreendidos e consequentemente mais fáceis de serem explicados pelo cliente. Para isto, Icelus procurou verificar técnicas já existentes que guiassem a quebra de problemas em itens menores que poderão tanto, e principalmente, facilitar a compreensão destes itens mais simples, como também já representarão um fluxo das atividades envolvidas na solução do problema.

Entre as ferramentas avaliadas, a árvore causal apresentou as melhores características que permitissem atender as necessidades de Icelus. Desta forma, a partir de um problema inicial, procura-se identificar outros subproblemas menores que podem ter sido a causa deste problema ocorrer ou podem ser gerados como consequência deste. Com esta árvore causal fica mais fácil expandir o problema pois passa a ter uma linha causal ligando os possíveis subproblemas envolvidos e com estes problemas se tornando visíveis, é mais fácil identificar quais de fato devem fazer parte da solução a ser desenvolvida.

4.3.1.1 Métodos de expansão de problemas

Existe algumas ferramentas que buscam realizar uma expansão do universo em que o problema se encontra, seja buscando novos problemas, seja procurando encontrar possíveis relações que permitam sair do problema atual e alcançar a meta desejada. Algumas destas ferramentas serão detalhadas na sequência desta seção.

Uma destas ferramentas existentes para a expansão de um problema e assim buscar uma sequência lógica de ações que permita solucioná-lo, é o STRIPS (FIKES; NILSSON, 1972). STRIPS é uma ferramenta que procura criar um planejamento de ações que permitam sair de um estado inicial para o estado final. Este planejamento é realizado a partir da decomposição de ações encontradas a partir de um problema inicial. Desta forma, para o estado a ser analisado deve-se identificar uma meta e a partir desta meta, buscar formalizar uma tarefa com seus parâmetros, consequências e pré-condições que permitam levar este estado a alcançar a meta desejada.

Mais focado na busca dos operadores possíveis (tarefas) que permitam levar o estado atual ao estado desejado, STRIPS não auxilia muito na identificação dos possíveis estados que podem ser alcançados a partir do estado atual. Desta forma, pode-se verificar que este método procura detalhar muito mais informações do que apenas buscar quais deveriam ser as próximas preocupações do problema inicial. Focando na busca das possíveis soluções do que de fato apenas expandindo o problema inicial. Tornando-o inválido para a necessidade de Icelus, que busca apenas um método capaz de expandir o problema inicial em outros problemas que estejam relacionados e permitam uma melhor compreensão do que deverá ser solucionado.

Outra ferramenta existente é o HTN (*Hierarchical Hierarchical Task-network* (EROL; HENDLER; NAU, 1994; EROL, 1996)). HTN foi criado inicialmente para solucionar algumas

falhas existentes entre o planejamento utilizando STRIPS e as necessidades encontradas em Pesquisa Operacional. Por ser uma derivação do STRIPS, as necessidades de Icelus que não foram supridas por STRIPS ainda permanecem sem serem atendidas por HTN, pois ainda foca mais no planejamento de ações do que simplesmente a expansão do problema inicial.

Da mesma forma de STRIPS, HTN é um método para definição de planos entre o estado atual e o estado alvo que deseja-se alcançar. Desta forma, além do estado atual, se faz necessário o conhecimento do estado desejado, o que dificulta a expansão desejada pelo método de expansão de problemas buscado por Icelus. Icelus necessita expandir o conhecimento do problema a partir de um problema inicial sem necessariamente ter conhecimento de onde deve-se chegar ao término da expansão deste problema.

Similar a STRIPS e a HTN, também é possível encontrar GOAP (*Goal-Oriented Action Planning* ([ORKIN, 2004](#); [WEBER; MATEAS; JHALA, 2010](#))). GOAP mantém o mesmo foco de STRIPS e HTN, procurando identificar a sequência de ações necessárias a serem realizadas para sair do estado inicial e alcançar o estado final já conhecido. GOAP procura mapear cada uma das metas (estados finais) aos seus estados iniciais e no momento em que um estado inicial é alcançado, o sistema passa a realizar o planejamento necessário para alcançar seu estado final.

GOAP é uma técnica bastante utilizada para planejamento de ações de Agentes Inteligentes, devido a sua dinamicidade. Porém, é mais um método que busca criar um planejamento entre o estado inicial e o estado final já conhecido, não se adequando as necessidades de Icelus.

Enquanto, a Árvore de Problemas, ou também conhecida como Árvore Causal ([SNOW-DON; SCHULTZ; SWINBURN, 2008](#)), busca criar relações entre o estado atual e estados futuros ou passados a partir da relação de causa e consequência. Sendo esta ligação com estados passados um grande diferencial quando comparada a STRIPS, HTN e GOAP, pois utilizando uma Árvore de Problemas é possível encontrar tanto estados que são metas do estado atual, como também identificar estados em que sua meta é alcançar o estado atual. Desta forma, não é necessário ter conhecimento prévio de outros estados que não seja o atual, permitindo realizar a expansão do grafo de estados até alcançar a abrangência desejada, ou esgotar as possibilidades existentes. Assim sendo, a utilização da Árvore de Problemas é muito mais voltada para a expansão do conhecimento a respeito de um universo do que a busca pelo planejamento de como realizar a movimentação do estado atual e alcançar o estado desejado.

Devido as características listadas dos métodos STRIPS, HTN, GOAP e Árvore de Problemas, foi decidido a utilização de uma Árvore de Problemas para permitir a expansão do conhecimento do problema em Icelus.

4.3.2 Método

Desta forma, Icelus é um método que procura especificar o problema a ser solucionado por um sistema inteligente. Surgido a partir da observação da dificuldade em alunos iniciarem projetos que envolvam sistemas com diversos subproblemas e inúmeros agentes inteligentes,

como é o caso da IA para controlar um exército no jogo Starcraft. Icelus procurará definir uma documentação descritiva do problema que permitirá um melhor planejamento da solução a ser desenvolvida.

Desta forma, Icelus procura utilizar algumas técnicas já existentes em outros campos do conhecimento para a realização da captação de informações do problema. Esta coleta de informações poderá ser guiada como uma coleta de requisitos, uma atividade muito bem conhecida no campo da Engenharia de Software. Para suportar esta atividade, Icelus define um formulário que pontuará quais os tipos de informações do problema são as mais importantes e devem ser coletadas.

Outro ponto chave de Icelus é a utilização de uma Árvore de Problemas que permite auxiliar a expansão do problema atual em problemas mais simples que poderão ser compreendidos mais facilmente. A utilização desta ferramenta permite a expansão da completude da análise o que permitirá uma solução que atenderá de forma mais completa o problema inicial.

Uma documentação detalhada do problema é considerada como resultado final da utilização de Icelus. Esta documentação servirá como entrada para o restante do processo de desenvolvimento da solução a ser construída. Permitindo assim, que o restante da equipe já tenha uma ideia mais completa do que de fato é o problema a ser solucionado já no início do processo de definição da solução.

4.4 Conclusão

Este capítulo realizou uma apresentação superficial do que seria Icelus, mostrando sua motivação e algumas escolhas realizadas durante a pesquisa realizada para a sua proposta. Como principal motivação, foram apresentadas as dificuldades enfrentadas pelos alunos em turmas de graduação, no momento em que foram solicitados à desenvolver o projeto de final de uma disciplina, que consistia na descrição e implementação de uma IA para controlar um jogo RTS, em específico o jogo Starcraft.

A partir desta motivação, Icelus foi sendo concebido (melhor detalhado no próximo capítulo), sendo focado principalmente na coleta de informações relacionadas ao problema a ser solucionado e na quebra do problema inicial em subproblemas mais simples de serem solucionados e compreendidos. Enquanto a primeira é suportada por técnicas de coleta de requisitos existentes em Engenharia de Software e na utilização de um formulário definido por Icelus, a segunda é suportada pela utilização da técnica de Árvore de Problemas, facilitando o processo de expansão do problema inicial. No sexto capítulo, a versão atual de Icelus será melhor explanada, permitindo uma melhor compreensão de como utilizá-lo.

Desta forma, no próximo capítulo, serão apresentadas as várias versões de Icelus e como este evoluiu durante o tempo, como também, será mostrado como ocorreu este processo evolutivo. Enquanto no Capítulo 6, a última versão de Icelus será apresentada e também como esta versão deve ser utilizada.

5

A Evolução de Icelus

Icelus é um método que precisou ser evoluído ao longo do tempo, para isto seguiu um processo evolutivo incremental que será detalhado melhor ao longo deste capítulo. Como também as versões intermediárias do método serão apresentadas, mostrando os resultados obtidos e quais foram as principais diferenças entre cada uma destas versões.

5.1 Refinamento

Como visto, Icelus surgiu a partir da percepção da dificuldade dos alunos em desenvolver um SMA capaz de controlar um exército em um jogo RTS. Todo o processo de compreensão do problema, especificação e desenvolvimento da solução era realizado de forma ad hoc pelos alunos, o que não é apenas uma prática da academia, uma abordagem ad hoc também pode ser encontrada na indústria, onde o desenvolvimento de SMA chega a ser realizado partindo diretamente para a prototipação da solução, sem ao menos ter despendido algum tempo na interpretação do problema abordado.

Desta forma, a ideia de Icelus partiu do princípio já consolidado em ES que a utilização de um método pode, e deve, melhorar o desempenho da equipe de desenvolvimento ([PRESSMAN, 2001](#)). Com este princípio, um levantamento bibliográfico foi realizado buscando informações do que poderia ser feito para guiar os alunos nesta árdua tarefa de desenvolver um SMA tão complexo. Porém, para poder avaliar de fato o que seria ou não importante para uma primeira versão de um método, foi necessário observar como os alunos se comportavam tendo que realizar esta tarefa de definir um SMA para um jogo RTS. A partir da execução e observação deste experimento, foi possível levantar as primeiras hipóteses do que poderia ser útil na construção do método Icelus. A partir deste conjunto de hipóteses, foi possível apresentar uma primeira versão de Icelus.

Este processo de experimentação realizado para a definição da primeira versão de Icelus foi aplicado outras vezes permitindo a sua evolução. Em cada novo experimento realizado, novas hipóteses foram levantadas e serviram como guia para o estudo do que poderia ser modificado em Icelus buscando validar as hipóteses. Para cada conjunto de modificações realizadas, novos

experimentos seriam realizados, buscando validar as modificações; e, conseqüentemente, novas hipóteses seriam definidas.

Desta forma, cada versão apresentada passou um processo de validação, como por exemplo, a execução de um experimento controlado com um grupo de participantes. Este experimento foi executado duas vezes, com a segunda e a atual terceira versão. Onde o grupo de participantes foi dividido em duas metades e para ambas foi solicitado que descrevessem a IA para controlar um jogo RTS, porém para uma das metades seria apresentado o método Icelus e para a outra metade não, iria realizar a descrição da IA de forma ad hoc. As duas metades foram divididas em duplas e todos tiveram o mesmo tempo disponível para finalizar a atividade.

Com o resultado em mãos, a segunda etapa do processo evolutivo de Icelus dá-se início, onde estes resultados deveriam ser submetidos a uma avaliação às cegas por dois avaliadores externos com experiência em Jogos, IA e SMA, com o intuito de quantificar os problemas identificados por cada grupo e qualificar as descrições realizadas para cada decisão. O capítulo 7 apresentará melhor os últimos resultados obtidos.

A última etapa deste processo evolutivo é iniciado com as observações realizadas pelos avaliadores, onde a partir destas observações, um estudo da literatura era realizado buscando sustentação para as modificações à serem realizadas em Icelus. Com estas alterações implementadas, um pequeno estudo de caso era realizado para fazer uma primeira validação da nova versão. Este estudo de caso era a utilização do método para especificar uma pequena parte da IA para um RTS, podendo assim avaliar previamente se as modificações chegaram a surtir algum efeito ou não. Com esta primeira validação realizada, uma nova versão de Icelus era estabelecida e deveria iniciar uma nova iteração do processo evolutivo, submetendo esta nova versão a um novo experimento controlado e conseqüentemente sua posterior avaliação.

5.2 Versões

Com este processo de refinamento definido, Icelus foi amadurecendo a cada nova versão apresentada. No momento, existem três versões do método já experimentadas e avaliadas; e uma nova versão que surgiu após o último experimento, porém esta nova versão não sofreu grandes modificações, havendo apenas modificações no processo de sua aplicação. Na tabela 5.1 é possível ter uma visão geral das características de cada uma das versões definidas no processo que chegou a definir Icelus. Logo em seguir, as duas primeiras versões serão apresentadas detalhadamente, com seus respectivos resultados obtidos a partir dos experimentos realizados.

5.2.1 Icelus v1

Icelus em sua versão inicial foi proposto baseado na coleta de informações dos subproblemas que deveriam ser solucionados durante a solução do problema. Com estes dados, o analista passa a ter uma melhor visão e compreensão destes subproblemas que serão solucionados, permi-

Tabela 5.1: Comparativo entre as características de Icelus em suas diferentes versões.

Versão	Identificação de subproblemas	Descrição dos subproblemas	Dificuldades identificadas	Modificações
v1	ad hoc	Questionário simples	Coleta de informações que não pertenciam ao escopo do problema e sim da solução	---
v2	ad hoc	Formulário de dados específico	Necessidade de abranger a identificação dos subproblemas	Definição de um formulário melhor estruturado para a coleta específica dos dados do problema
v3	Árvore causal	Formulário de dados específico	Melhor especificação do método	Utilização de uma árvore causal para guiar a identificação do subproblemas envolvidos

Tabela 5.2: Exemplo de planilha da aplicação da primeira versão de Icelus.

Id	Subproblema	Quando	Variáveis	Como	Quem	Protocolo
1	Deve minerar	Trabalhador inativo	Há outra prioridade (estratégia)	Regras	Unidade mineradora	Central
2	Onde minerar	Depende de #1	Distância para o Centro de Comando	Regras	Unidade mineradora	Central e distribuído
			Acessibilidade da mina			
			Ocupação da mina			

tindo identificar qual seria a melhor solução para o caso especificado. Para auxiliar e organizar estes dados, foi proposto uma planilha indicando quais informações deveriam ser coletadas para cada uma dos subproblemas.

Sendo guiado pelas informações requisitadas pela planilha, o analista deveria identificar para cada subproblema: (i) o momento de solucionar o subproblema; (ii) as variáveis que deveriam ser consideradas na solução; (iii) como seria a ideia da solução, se utilizaria regras ou uma função; (iv) como seria a questão da responsabilidade do subproblema, se a solução seria centralizada ou distribuída; e (v) como seria realizada esta solução. Um exemplo de preenchimento da planilha pode ser visualizado na Tabela 5.2. Simplificadamente, esta versão inicial do método Icelus era resumida na aplicação desta planilha, a utilizando como guia na extração das informações do problema.

Esta planilha serve de base para um questionamento ao especialista do problema e pode ser preenchida a partir da realização de algumas perguntas, como: (i) "O que deve ser feito?", para identificar qual a decisão que será detalhada; (ii) "Quando tomar a decisão?", para identificar o momento em que a decisão deverá ser processada; (iii) "Quais informações do ambiente são importantes para a decisão", tentando identificar as variáveis do problema que deverão ser consideradas ao processar a decisão; (iv) "Como a decisão será decidida?", com o intuito de identificar se a decisão será tomada a partir de um conjunto de regras ou a partir de uma função utilidade; e (v) "Quem decide?", para identificar a responsabilidade da decisão, atribuindo a decisão a uma entidade que provavelmente será representada por um agente inteligente. A partir destas informações, o analista pode preencher o último campo de acordo com sua experiência, já identificando se esta decisão seria tomada de forma centralizada, por um único agente, ou se seria de forma distribuída, sendo processador por um conjunto de agentes.

Icelus é centrado no conjunto de subproblemas que compõem o problema inicial, permitindo ao analista descrever cada um destes subproblemas identificados que conseqüentemente torna o problema principal mais fácil de ser compreendido. Porém, mesmo com a presença deste guia de perguntas que orienta quais informações são relevantes para a especificação de um subproblema, a identificação destes subproblemas ainda era de total responsabilidade do analista. Neste momento, o único auxílio de Icelus na identificação de subproblemas era o questionamento do que deveria ser feito para solucionar o problema, não existindo um guia de como expandir este questionamento e aumentar a abrangência desta análise.

Esta primeira versão v1, mesmo ainda muito imatura, foi apresentada e utilizada como base na disciplina de SMA, obtendo ótimos resultados preliminares. Com a sua utilização, os alunos passaram a descrever melhor a solução que propuseram como projeto, além dos professores da disciplina identificarem que o código desenvolvido para o projeto apresentava uma melhor estruturação. Como base de comparação, foram utilizados os projetos dos semestres anteriores onde os alunos realizavam este projeto de forma *ad hoc*. Outro resultado relevante encontrado com a utilização desta primeira versão de Icelus foi uma modelagem inicial dos problemas existentes em um jogo RTS, Figuras 5.1 e 5.2. Esta modelagem foi definida a partir da utilização da primeira versão de Icelus por um grupo de alunos do curso de graduação, tendo sido utilizada como base do desenvolvimento do projeto de fim de semestre a também apresentada e descrita no relatório entregue como parte da nota da disciplina. Este modelo representa a sequência de decisões a serem realizadas ao planejar a IA para Starcraft, mostrando as decisões quais outras decisões serão desencadeadas quando algumas destas decisões sejam realizadas. Mais informações destes resultados obtidos podem ser visto em (ROCHA et al., 2012).

Analisando os resultados obtidos durante esta fase de especificação do problema, foi possível identificar possíveis melhorias do método. Algumas informações solicitadas na planilha não diziam respeito ao problema e sim à solução e que ainda não deveria fazer parte da análise. Com isto, algumas modificações foram realizadas a partir de estudos realizados. Estas alterações buscaram isolar quais informações são de importância para um problema e quais já estavam

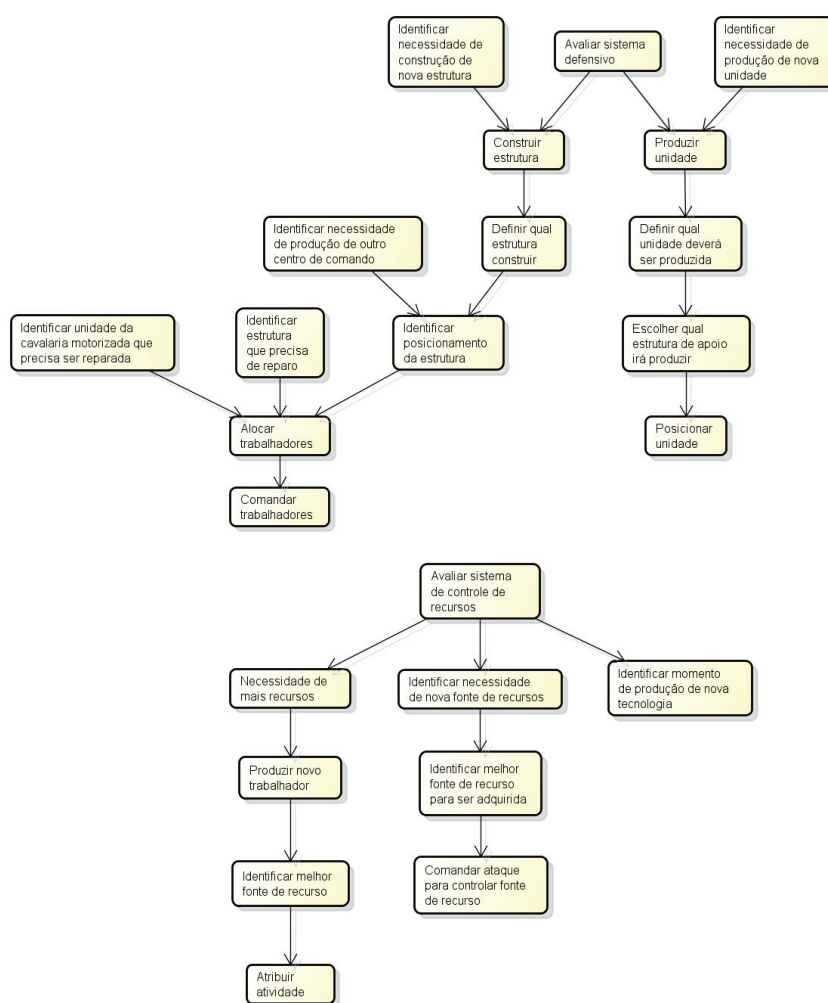


Figura 5.1: Modelo de problemas identificados em Starcraft a partir do primeiro experimento utilizando Icelus.

relacionadas à solução. Esta separação foi necessária, pois neste momento as informações de uma possível solução não são importantes e não devem ser levadas em consideração.

5.2.2 Icelus v2

A primeira versão de Icelus guiava o analista na coleta de diversas informações, mas algumas destas já estariam relacionadas à solução e não apenas ao problema, característica que não é o intuito do método. Icelus é um método apenas para análise do problema, não devendo limitar a solução. Desta forma, foi necessária a realização de algumas modificações neste método inicial. Estas alterações foram realizadas na estrutura da planilha que guia o analista na coleta das informações.

As modificações foram realizadas após um estudo sistemático da literatura existente em outras áreas da Computação sobre especificação de problemas, como por exemplo, a literatura sobre **BPM!** (BPM!). Outra área que também foi abordada durante os estudos foi à área de

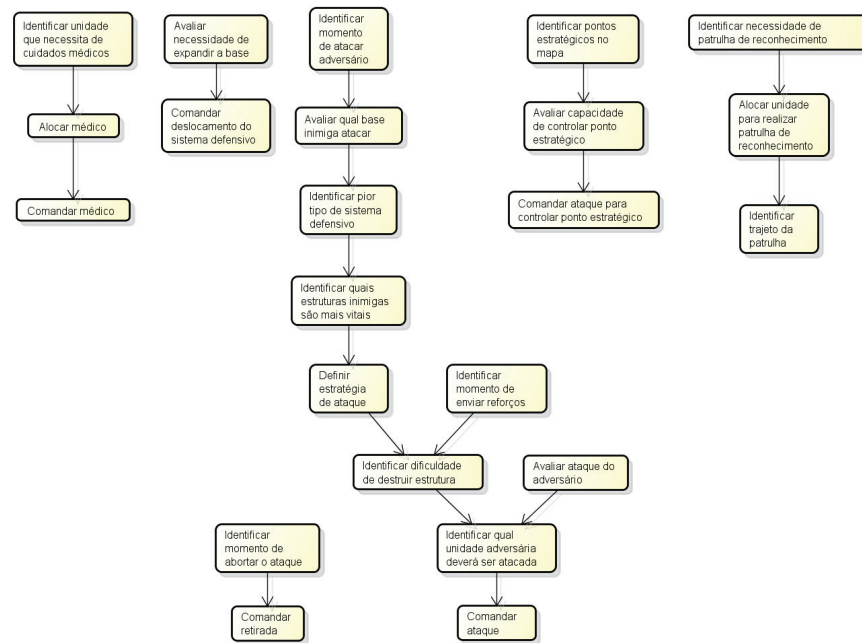


Figura 5.2: Modelo de problemas identificados em Starcraft a partir do primeiro experimento utilizando Icelus (continuação).

SMA, em especial estudos que utilizaram jogos RTS como suporte. Esta delimitação do escopo na área de SMA se deu devido a utilização de um jogo RTS como estudo de caso da proposta do método Icelus. No entanto, Icelus está sendo desenvolvido com o intuito de ser capaz de lidar com qualquer tipo de problema MAMT, independente do contexto.

Durante o período em que foram realizadas estas modificações, algumas versões de Icelus foram geradas e para cada uma destas versões, foram realizadas algumas avaliações superficiais para averiguar o impacto da mudança. Para isto, estas versões foram utilizadas na especificação de uma parte do problema existente em controlar a IA de um jogo RTS, foi o caso do sistema de defesa do centro de comando de Starcraft. Após a identificação dos subproblemas existentes em defender o centro de comando, foi realizada a especificação destes subproblemas, tendo em seguida a avaliação da documentação gerada. Esta avaliação foi realizada por dois professores com experiência em SMA e jogos, buscando identificar a relevância dos dados coletados e se estes dados estavam relacionados exclusivamente com o problema. Este processo evolutivo da primeira para a segunda versão foi realizado de forma iterativa e incremental (Figura 5.3), onde Icelus foi submetido a esta avaliação para cada conjunto de modificações realizadas, tendo como resultado de cada avaliação um relatório sobre a relevância dos dados coletados. Este relatório posteriormente guiaria os estudos em busca de possíveis modificações em Icelus que novamente seria avaliado, fechando uma iteração.

Após algumas iterações, foi definida uma nova versão estável de Icelus. As modificações realizadas tiveram como impacto a modificação da estrutura do documento de coleta de dados dos subproblemas, deixando de ser uma planilha para seguir os modelos de formulários

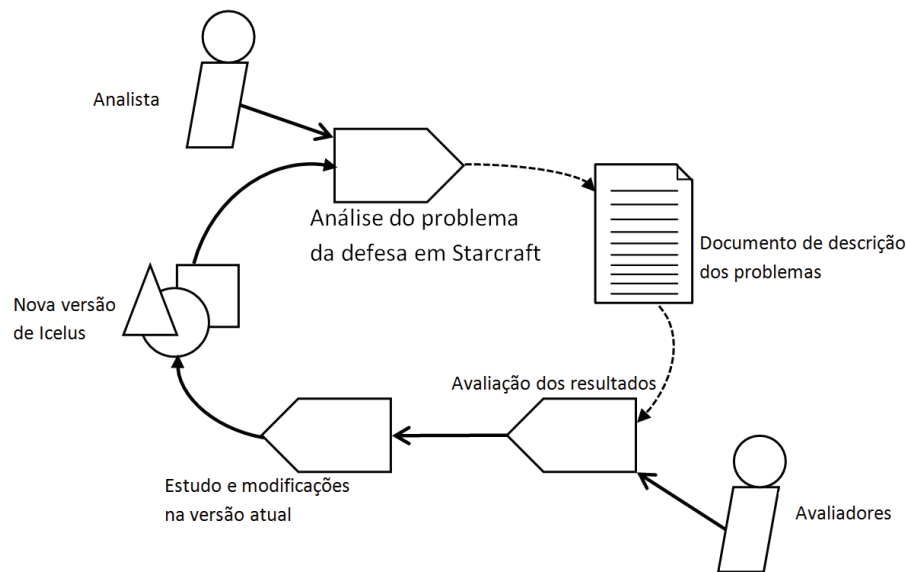


Figura 5.3: Processo iterativo e incremental em que Icelus foi submetido durante sua evolução.

existentes no campo da Engenharia de Requisitos. Outra característica adquirida da ES foi a utilização de identificadores para cada um dos subproblemas identificados, permitindo ao time de desenvolvimento rastrear o problema e sua solução correspondente durante todo o ciclo de produção do sistema, sendo possível realizar apenas modificações pontuais em caso de falhas de entendimento (GOTEL et al., 2012). Desta forma, estes subproblemas podem ser relacionados a outros documentos durante todo o processo de desenvolvimento da solução. Outras modificações foram realizadas diretamente nos dados coletados, tentando limitá-los apenas às informações do problema. Os dados coletados pelo formulário da segunda versão de Icelus podem ser visto a seguir:

SBP_01 : Subproblema o identificador do subproblema que permitirá o rastreamento por todo processo de desenvolvimento.

- 1. Descrição** uma descrição clara e breve do que realmente significa o subproblema, permitindo que o subproblema em questão seja compreendido em qualquer momento do ciclo de desenvolvimento.
- 2. Objetivo** identificar qual é o objetivo na solução deste subproblema, sendo possível perceber a razão de este subproblema ser considerado para a solução.
- 3. Variáveis** lista de informações do ambiente que são significantes para o subproblema, passando a ser dados que devam ser considerados no momento de codificar a solução.
- 4. Contexto do ambiente** um contexto representa um estado (conjunto de variáveis, ações, situações) do ambiente em que o problema estará inserido. Dependendo deste contexto o problema poderá ser influenciado, sendo necessário considerar outras informações na solução deste subproblema. Um exemplo seria a decisão de construir uma edificação em um jogo RTS, caso o contexto atual esteja favorável a realizar um ataque ao inimigo, o

ideal é construir alguma edificação que permita produzir unidades mais fortes ou que permita a melhoria da tecnologia do exército, mas se o contexto for de defesa, o ideal é focar em edificações de suporte a defesa do centro de comando, como a construção de bunkers e torres de mísseis.

- 5. Influência temporal** procurar avaliar se o tempo gasto no processamento da solução poderá influenciar na escolha da solução, caso da produção de uma nova unidade militar, em determinados momentos é mais vantajoso gastar mais tempo e produzir unidades mais fortes do que produzir unidades mais fracas, mas que necessitam de menos tempo para serem produzidas; em outros momentos não, é mais importante produzir unidades mais fracas e mais rapidamente.
- 6. Gatilhos** identificar qual o momento em que o problema será requisitado para ser solucionado, podendo ocorrer a partir de algum temporizador interno ao sistema, a partir do resultado de outros subproblemas ou a partir de eventos externos.
- 7. Nível do subproblema** identificar qual o nível de conhecimento que este subproblema necessitará do restante do ambiente, identificando se terá apenas conhecimento das informações locais ou se será necessário informações mais gerais do ambiente.

Também ocorreu uma alteração significativa na utilização do método em relação à primeira versão para esta segunda versão de Icelus. Primeiramente pela formalização através da proposta de um processo para guiar a utilização e segundo por ter sido inserida algumas etapas neste processo (Figura 5.4). Primeiramente a utilização de Icelus deve começar pela identificação se caso o problema seja suscetível a variações de contexto do ambiente, listando estes possíveis contextos. Posteriormente, na segunda etapa a identificação dos subproblemas existentes deve ser realizada. Porém, ainda nesta versão de Icelus, esta etapa não havia sido detalhada corretamente, deixando-a a cargo do analista que deveria identificar os subproblemas de forma ad hoc. Para cada um destes subproblemas, o analista deve preencher o formulário de Icelus com as informações do problema, utilizando a lista de contexto identificada no primeiro passo. Como resultado, um conjunto de documentação referente aos subproblemas que foram identificados, que facilitarão a proposta da solução e sua posterior codificação.

Esta versão foi submetida a um experimento similar ao realizado com a primeira versão de Icelus. Porém, este novo experimento foi realizado na forma de um teste AB, separando os participantes em dois grupos, onde um utilizaria Icelus enquanto o outro não, permitindo comparar o impacto da utilização de Icelus frente ao outro grupo que não utilizou o método. O grupo de participantes do experimento foi formado por um conjunto de 25 alunos de graduação em seu último ano, alunos que já possuíam experiência em IA por terem no mínimo cursado uma disciplina sobre IA. Por ser uma disciplina de fim de curso e possuir outras disciplinas que envolviam IA e Sistemas Inteligentes (SI) como pré-requisito, os alunos já possuíam uma certa experiência, no mínimo acadêmica, sobre o assunto de SMAs. Procurando avaliar o perfil de jogador dos alunos, foi realizado um questionário e estas informações podem ser vistas na

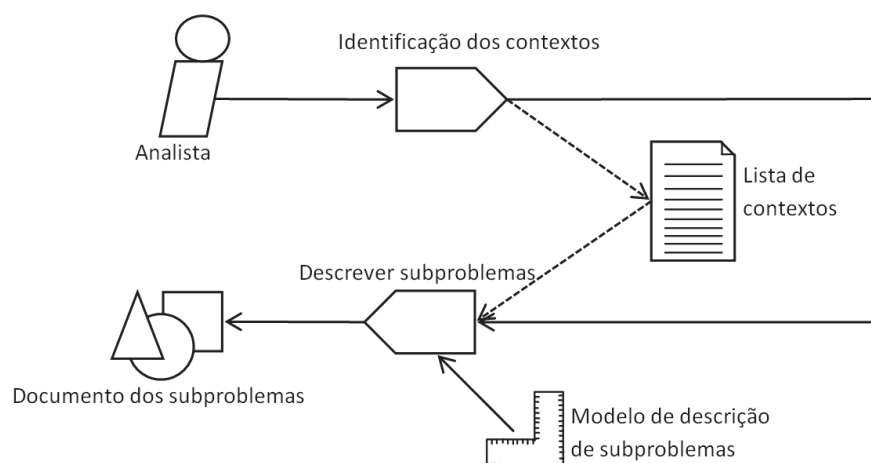


Figura 5.4: Processo de aplicação desta segunda versão de Icelus.

Figura 5.5. Com isto, foi possível identificar que além da experiência com SMAs e Sistemas Inteligentes (SI), os alunos também tinham experiências com jogos RTS, onde cerca de metade dos participantes já haviam jogado jogos RTS por um tempo considerável (mais de 100h), como também tinham contato com partidas online.

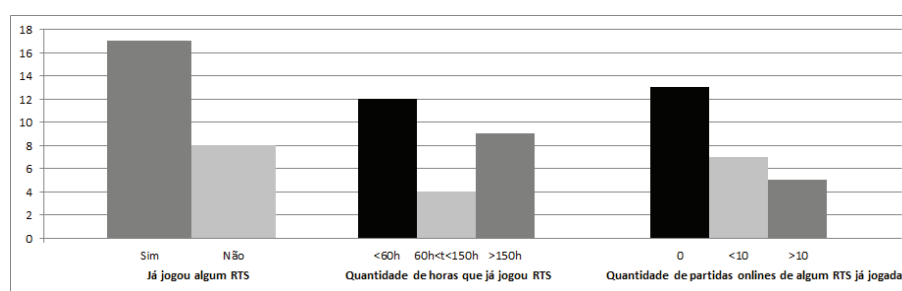


Figura 5.5: Perfil de jogador dos alunos que participaram do experimento.

Estes 25 participantes foram divididos em dois grandes grupos; denominados como grupo Icelus os que receberam treinamento na utilização de Icelus e iriam utilizar o método durante a atividade; e como grupo *ad hoc* os que não iriam utilizar a metodologia na atividade. Finalmente, estes dois grupos se organizaram em duplas para realizar a especificação da IA para Starcraft. Estes dois grupos foram separados em dois ambientes distintos, tendo os integrantes do grupo Icelus recebido inicialmente um treinamento na utilização do método durante 15 minutos iniciais, enquanto o grupo *ad hoc* não teve nenhum treinamento adicional. Posteriormente, foi solicitado que durante uma hora, ambos os grupos realizassem a atividade de listar os subproblemas existentes na IA para Starcraft e especificá-los.

Estes documentos de descrição da IA de Starcraft definidos pelos dois grupos (Icelus e *Ad hoc*) foram submetidos a duas avaliações cegas realizadas por dois professores especialistas em Jogos e em SMA. As duas avaliações foram realizadas no âmbito quantitativo, avaliando a quantidade de subproblemas identificados, e no âmbito qualitativo, avaliando a qualidade da descrição e o quanto ficaria mais fácil de propor uma possível solução. Os resultados obtidos

Tabela 5.3: Resultados obtidos da avaliação das seis equipes que compuseram o grupo de participantes que utilizaram o método Icelus na atividade de descrição da IA de Starcraft.

Grupo Icelus	Quantitativa		Qualitativa	
	Avaliador 1	Avaliador 2	Avaliador 1	Avaliador 2
Equipe 1	3	3	4	5
Equipe 4	2	2	4	3
Equipe 6	5	5	5	5
Equipe 7	4	4	4	5
Equipe 9	3	3	5	5
Equipe 11	3	3	4	4
Média	3,33	3,33	4,33	4,5

Tabela 5.4: Resultados obtidos da avaliação das seis equipes que compuseram o grupo de participantes que não tiveram contato com o método Icelus, realizando a atividade de descrição da IA de Starcraft de forma ad hoc.

Grupos <i>Ad hoc</i>	Quantitativa		Qualitativa	
	Avaliador 1	Avaliador 2	Avaliador 1	Avaliador 2
Equipe 2	5	5	3	4
Equipe 3	5	5	1	1
Equipe 5	1	1	2	1
Equipe 8	1	1	2	1
Equipe 10	5	4	2	2
Equipe 12	3	2	1	1
Média	3,33	3	1,83	1,67

pelo grupo Icelus podem ser vistos na Tabela 5.3 e os resultados do grupo *ad hoc* na Tabela 5.4.

Como pôde ser visto a utilização de Icelus não causou impacto significativo na média de identificação dos subproblemas, tanto os grupos de participantes que utilizaram Icelus como os que não utilizaram obtiveram classificação similar pelos avaliadores. No entanto, a avaliação qualitativa apresentou que os participantes que utilizaram Icelus descreveram melhor estes subproblemas do que os participantes que não receberam.

De forma espontânea, os avaliadores chegaram a classificar os documentos das equipes participantes do experimento em três casos distintos: (i) equipes que apresentaram uma maior visão do problema inicial; (ii) equipes que não tiveram uma visão completa do problema, descrevendo os subproblemas encontrados de forma bastante superficial; e (iii) equipes que descreveram os subproblemas encontrados ao nível de script. As equipes 1, 6, 7, 9 e 11 (todos do grupo que utilizou Icelus) demonstraram uma descrição mais detalhada, sendo classificados como pertencentes ao primeiro caso. A equipe 4 que também utilizou o método Icelus, foi classificada como pertencente ao caso (iii), enquanto todas as equipes *ad hoc* foram classificadas como apenas uma descrição superficial dos subproblemas, caso (ii). Esta avaliação qualitativa realizada de forma espontânea pelos avaliadores reforçara o impacto positivo da utilização de

Icelus como método de análise de problemas.

Para finalizar este experimento, foi realizado um levantamento do sentimento dos participantes a respeito da utilização de Icelus. As respostas coletadas confirmaram algumas impressões dos avaliadores (Figura 5.6); (i) a utilização de Icelus fez com que fosse necessário um pouco mais de tempo na especificação do problema, (ii) mas torna a descrição do problema mais fácil, (iii) levando em consideração mais variáveis de ambiente e (iv) resultando em uma especificação mais fácil de ser codificada. Em outras palavras, Icelus gera um *overhead* temporal na análise do problema, mas minimiza possíveis erros nas fases seguintes, inclusive minimizando a dificuldade na codificação da solução.

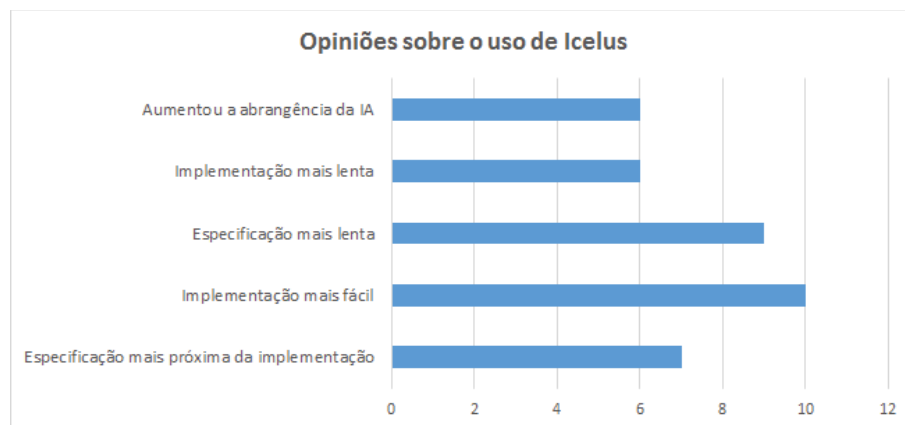


Figura 5.6: Opiniões dos participantes do experimento a respeito da utilização de Icelus.

Mesmo após a análise destes dados e a confirmação que Icelus já apresentara bons resultados, novas modificações foram realizadas com o intuito de melhorar a identificação de subproblemas, tornando o método ainda mais amplo nesta fase de análise. Estas modificações geraram a terceira versão de Icelus e a necessidade de novos experimentos.

5.3 Conclusão

Este capítulo apresentou o processo evolutivo de Icelus, iniciando com seu surgimento ainda de forma bastante simplória, baseando-se apenas em uma planilha até a sua segunda versão, já mais formalizada com um processo e com planilhas guiando a fase de descrição dos subproblemas. Durante o capítulo, também foi apresentado o processo evolutivo, mostrando como de fato ocorreu a evolução de uma versão para outra.

Nos próximos capítulos serão apresentados a última versão de Icelus e como fazer seu uso e posteriormente os resultados obtidos com um último experimento realizado.

6

Icelus

A análise de um problema basicamente reflete a quebra deste em subproblemas menores e mais simples de serem compreendidos, e o posterior estudo destes subproblemas garantindo sua compreensão. A versão anterior de Icelus já obteve bons resultados na descrição destes subproblemas, mais ainda não apresentava nenhum auxílio na fase inicial de quebra do problema principal. Desta forma, um estudo foi realizado com o intuito de realizar algumas modificações no processo de aplicação de Icelus, permitindo um melhor auxílio na quebra do problema principal.

Para melhorar a identificação dos subproblemas foi incorporada a Icelus a teoria da árvore de problemas, também conhecida como árvore causal (SNOWDON; SCHULTZ; SWINBURN, 2008). Uma árvore de problemas é uma ferramenta para mapear outros problemas ligados a um problema principal, a partir de suas causas e consequências. Este mapeamento permite uma melhor análise do escopo em que o problema original está inserido, permitindo visualizar vários subproblemas e suas correlações facilitando a identificação de quais destes são realmente significantes para a solução que será proposta.

Como toda árvore computacional, uma árvore de problemas inicia por um subproblema qualquer como raiz. Seus nós filhos irão ser representados por outros subproblemas que serão identificados a partir das consequências ou causas do anterior. Para cada novo subproblema identificado, é possível buscar novas causas e consequências, expandindo a árvore com novos subproblemas.

Com a incorporação da árvore de problemas como ferramenta para auxiliar a identificação dos possíveis subproblemas ligados ao problema principal, Icelus passou a dar suporte a todo o processo de análise de um problema. Começando com a quebra de um problema complexo em subproblemas mais simples e encerrando com a aplicação do formulário capaz de expandir o conhecimento a respeito de um subproblema em específico. Salientando que apesar de parecer um método muito rígido para ser utilizado em ambientes mais ágeis, é possível utilizar Icelus apenas para uma parte do sistema a ser desenvolvido. Como por exemplo, no caso de um projeto guiado por uma adaptação de SCRUM, é possível utilizar Icelus apenas para alguns dos itens existentes no Product Backlog. Desta forma, Icelus permitirá expandir estes itens existentes

no backlog podendo assim aumentar a precisão da estimativa do sprint por passar a lidar com atividades mais simples de serem compreendidas, como também a utilização do restante do método Icelus permitirá uma melhor compreensão do problema, aumentando a capacidade de propor a solução do item e a posterior codificação desta solução.

Porém, Icelus também é possível que seja utilizado de forma iterativa e incremental, onde a cada nova iteração o problema inicial pode ser expandido em apenas mais um nível. Desta forma, não é necessário se preocupar em fazer uma análise aprofundada do problema de uma única vez, permitindo que se obtenha resultados melhores por estar expandindo o problema de forma mais controlada, de acordo com a necessidade do projeto.

Devido a estas modificações na utilização do método, foi necessário reorganizar o processo de aplicação de Icelus, onde foi inseridas as etapas de coleta de informações e estruturação da árvore de problemas, este novo processo pode ser visualizado na Figura 6.1. Esta versão de Icelus foi dividida em quatro atividades: (i) Coletar dados do ambiente; (ii) Gerar árvore de problemas; (iii) Selecionar tarefas; e (iv) Descrever tarefas.

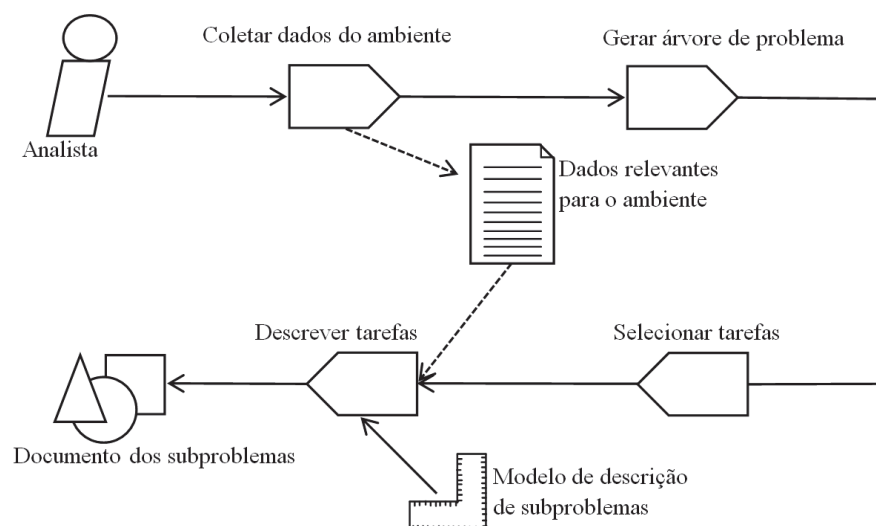


Figura 6.1: Processo de utilização da terceira versão de Icelus.

A seguir, cada uma destas fases serão abordadas, permitindo uma melhor compreensão sobre o processo de utilização desta versão de Icelus. Ao término de cada seção, será destacado o que se espera que seja entregue após realizar cada uma das fases.

6.1 Coletar dados do ambiente

A primeira atividade de Icelus é o levantamento das informações conhecidas do ambiente e do problema abordado. Estas informações serão úteis na delimitação do escopo do problema, diminuindo o risco de utilizar termos desconhecidos durante as etapas posteriores.

A importância desta fase vai além de apenas listar estas informações, o significado destas deve estar nivelado entre todos os *stakeholders* envolvidos. Este nivelamento ajudará a

no momento das comunicações posteriores, ficando mais fácil a troca de informações entre os clientes e o time de desenvolvimento. Como também, permitirá que erros de codificação por má interpretação da descrição sejam reduzidos.

Estes dados podem ser coletados de diversas formas, seja por uma entrevista direta com o cliente ou com o especialista do sistema (*stakeholders* responsáveis por deter o conhecimento a respeito do problema a ser solucionado) ou através de experimentações ou algum outro método que a equipe de desenvolvimento esteja familiarizada. No caso de um jogo RTS como Starcraft, é importante que todos os integrantes do time de desenvolvimento tenham conhecimento de quais entidades fazem parte do jogo e a importância de cada uma delas, como por exemplo, saber quais as entidades de recurso estão disponíveis e que de fato o controle das áreas que possuem determinados recursos são primordiais para um bom desempenho em uma partida do jogo. Como também, saber quais são as edificações existentes no jogo e saber, que por exemplo, o centro de comando é uma edificação vital e que deverá ser protegida a todo custo.

6.1.1 Entrega

Lista com as informações coletadas que são consideradas importantes e a descrição destas informações. Desta forma, um pequeno exemplo da coleta dos dados para a IA de Starcraft está estruturada a seguir:

Raças As possíveis escolhas do usuário de seus exércitos, existindo três raças: (i) *Terran*; (ii) *Protoss*; e (iii) *Zerg*. Cada raça possui suas particularidades, sendo melhores em algumas estratégias e necessitando de mais ou menos dos recursos existentes.

Minérios Um dos recursos disponíveis no jogo, muito utilizado pelos *Terrans* e *Zergs*.

Gás Vespeno O outro recurso disponível no jogo, sendo essencial para os *Protoss* e sendo necessário para as outras raças apenas para a construções e pesquisas mais avançadas.

Centro de comando Uma das edificações existentes no jogo, sendo a principal. Nesta edificação são produzidos os trabalhadores e também é responsável receber e armazenar os recursos disponíveis.

Trabalhador A unidade mais básica. Responsável por coletar os recursos e construir novas edificações.

6.2 Gerar árvore de problemas

Esta atividade irá auxiliar a identificação dos subproblemas relacionados ao problema principal. Para isto, é necessário selecionar um nó raiz para a árvore, que poderá ser um subproblema qualquer. Em Icelus, a árvore de problemas é expandida a partir das causas do estado inicial e das consequências do estado final de um subproblema qualquer.

Como por exemplo a produção de uma nova unidade de infantaria tem como estado inicial a necessidade de recursos, identificado como um novo subproblema; e possui como estado

final a existência de uma nova unidade, logo é necessário movê-la para algum ponto no mapa para ser posicionada ou simplesmente para atacar uma unidade adversária. Como visto neste exemplo, a partir de um problema raiz, identifica-se o estado inicial do subproblema e buscam-se as causas que levaram a este estado inicial (qual seria a necessidade que o ocasiona), como também o estado final do subproblema (o que acontece quando o subproblema é solucionado) e quais seriam as consequências por ter alcançado este estado final. Para cada nova causa ou consequência, um novo nó da árvore é gerado.

Após expandir a árvore em um nível identificando as causas e as consequências do problema inicial, é possível prosseguir encontrando novos nós a partir das causas e consequências destes novos subproblemas, aumentando o nível da árvore a cada nova rodada de expansão realizada.

Sendo esta necessidade de expandir o problema nas duas direções (origem e destino) o principal fato da escolha pela Árvore de Problemas, e não outras técnicas de planejamento como HTN, GOAP ou STRIPS.

6.2.1 Entrega

Árvore de problemas expandida a partir de um problema inicial. Onde um exemplo derivado do problema anteriormente exposto está representado na Figura 6.2. Em que, tem-se a produção de uma unidade de infantaria como nó raiz, podendo-se montar a árvore de problemas expandida como mostrado na figura.

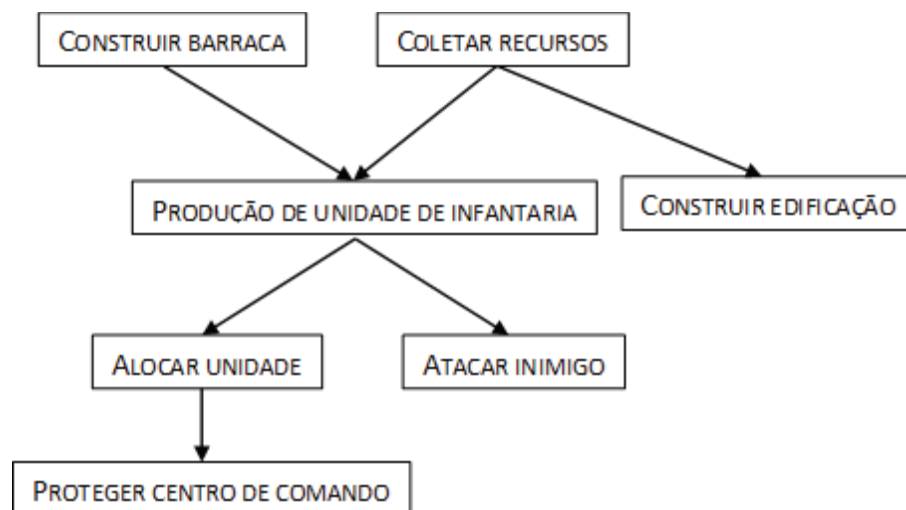


Figura 6.2: Exemplo de uma árvore de problemas de Starcraft, utilizando o subproblema da produção de uma unidade de infantaria como raiz.

Como apresentado na figura 6.2, é possível após expandir um nível, como com a identificação do subproblema da *coleta de recursos*, é possível expandir este nó, identificando novos subproblemas, como o que ocorreu com a identificação do subproblema *Construir edificação*. Ou por exemplo, com o caso de *Alocar unidade*, em que o subproblema *Proteger centro de comando*.

6.3 Selecionar tarefas

A partir da árvore montada, o time de desenvolvimento poderá selecionar os subproblemas mais importantes e mais complexos que necessitem de um melhor detalhamento para facilitar a especificação da solução. Esta escolha deverá ser realizada a partir da experiência do time e da descrição inicial dada pelo cliente, podendo selecionar apenas alguns ou até mesmo selecionar todos os nós da árvore para serem descritos.

Geralmente, a árvore de problemas irá conter subproblemas mais simples de serem compreendidos, não sendo necessário utilizar estes subproblemas nas próximas etapas. Mas também conterà subproblemas que estão presentes no problema inicial, mas que podem não ser considerados no momento em que for se pensar na solução, seja por simplificação necessária para atender possíveis restrições do problema, ou seja por, de fato, não terem uma significância considerável na solução a ser desenvolvida.

6.3.1 Entrega

Lista de subproblemas consideráveis para a abordagem de uma possível solução a ser desenvolvida posteriormente. Onde, olhando para o problema exemplo utilizado neste capítulo, tem-se a lista inicial de subproblemas como: **(i)** construir barracas; **(ii)** coletar recursos; **(iii)** produção de unidade de infantaria; **(iv)** construir edificação; **(v)** alocar unidade; **(vi)** atacar inimigo; e **(vii)** proteger centro de comando. Entre estes subproblemas, pode-se selecionar um subconjunto que contenha apenas os subproblemas considerados mais complexos e que necessitem de mais detalhes para serem compreendidos, como por exemplo: *coletar recursos* e *atacar inimigo*.

6.4 Descrever tarefas

Após iniciar a utilização de Icelus, listando os dados do problema, expandindo este problema inicial em subproblemas menores e mais simples de serem compreendidos, identificar quais destes subproblemas irão compor a possível solução que será desenvolvida posteriormente, chega o momento da última etapa, a descrição destes subproblemas. Para isto Icelus provê um formulário capaz de guiar o analista sobre quais informações de fato são necessárias para compreender realmente o problema.

Neste ponto, as informações coletadas na primeira fase passam a ter uma importância ainda maior, quanto mais a descrição realizada nesta fase for delimitada pelas informações coletadas na primeira fase, melhor, pois garantirá que todos os *stakeholders* tenham pleno conhecimento a respeito do problema.

6.4.1 Entrega

Lista de documentos descritivos dos subproblemas selecionados na fase anterior, onde os documentos são baseados no formulário padrão fornecido por Icelus. Desta forma, com a lista de subproblemas identificados na fase anterior, é possível utilizar o formulário de Icelus e realizar a descrição destes. Um exemplo de descrição está em sequência:

Coletar minério

Descrição Decidir quando designar operários para coletar minérios e de qual mina deve ser coletado.

Variáveis de ambiente (i) Número de trabalhadores; (ii) número de minas de minério; (iii) distância do centro de comando; (iv) número de trabalhadores coletando na mina.

Contexto Não aplicável

Tempo Não aplicável

Gatilhos (i) Início do jogo; (ii) produção de novo trabalhador.

Objetivo Otimizar a eficiência da coleta de minérios.

Nível Decisão será tomada por um nível mais gerencial e com influência em todo o jogo.

Este documento de descrição é considerado como saída de Icelus que poderá ser utilizado como entrada no desenvolvimento da solução ao utilizar outros métodos já existentes e abordados no capítulo 3 desta tese. Com a descrição realizada, a equipe de desenvolvimento possui um maior conhecimento a respeito do problema, permitindo propor mais facilmente uma possível solução.

6.5 Conclusão

É esperado que com a utilização desta nova versão de Icelus a quantidade de subproblemas identificados aumente de forma bastante considerável, permitindo uma melhor visão do problema como todo. Desta forma, quando comparado a uma abordagem não utilizando Icelus, é esperado que o número de subproblemas identificados pela utilização de Icelus seja muito maior, como também a qualidade da descrição permaneça com um bom nível de qualidade.

No próximo capítulo serão apresentados os dados de um experimento realizado com esta versão, procurando validar as modificações realizadas.

7

Icelus - Resultados

Alguns experimentos foram realizados buscando validar o método Icelus. Este capítulo irá apresentar o último experimento realizado, que foi executado com a última versão de Icelus.

7.1 Experimento

Com as últimas alterações realizadas na segunda versão de Icelus, uma terceira versão foi definida e esta deveria ser experimentada para poder obter o seu comportamento diante de problemas muito próximos dos problemas reais obtidos por uma equipe de desenvolvimento no dia a dia. Desta forma, um experimento foi conduzido onde Icelus foi submetido a um ambiente controlado e comparado com a não utilização de outros métodos.

Este experimento foi conduzido com um grupo de alunos de graduação na tentativa de caracterizar o impacto da utilização de Icelus quando solicitada a especificação da IA de um problema que pode ser caracterizado como MAMT. Estes alunos já se encontravam no fim do curso de Ciências da Computação, onde mais da metade dos participantes já possuíam experiências com o desenvolvimento comercial de ferramentas que envolvessem SMAs. Sendo assim, participantes que comprovadamente possuem um nível considerável de experiência, seja na indústria, seja na teoria.

Enquanto a avaliação dos resultados foram avaliados pelos professores da disciplina, os quais possuem larga experiência no campo de IA, SMA e desenvolvimento de jogos. Ressaltando apenas, que a avaliação foi realizada a partir de um processo as cegas, onde os avaliadores não tinham conhecimento a respeito de quais equipes haviam utilizado Icelus ou de forma *ad hoc*, como também não tinham ideia de quais alunos estavam em quais grupos, pois a documentação era apresentada aos avaliadores anonimamente.

É importante deixar claro que a experimentação que está sendo realizada é frente a utilização de Icelus contra a utilização de nenhum outro método, caso mais comum encontrado na indústria, onde a equipe de desenvolvimento se utiliza apenas da própria experiência para analisar o problema e criar uma especificação baseada apenas em prototipação. Desta forma, estando sujeita a muitas ambiguidades durante este processo de especificação inicial, mas que

podem ser identificadas rapidamente devido as provas realizadas com os protótipos iniciais.

Porém, este experimento não procura avaliar a utilização de Icelus durante o restante do processo de desenvolvimento devido a inserção de inúmeras outras variáveis inseridas as quais não fazem parte do escopo de controle de Icelus. Como por exemplo, técnicas de IA escolhidas para o desenvolvimento e quais tecnologias serão utilizadas para o desenvolvimento.

7.1.1 Definição das metas

Durante o experimento, dois critérios estarão sendo acompanhados: **(i) Quantidade**, onde por ter um tempo fixo para a realização da tarefa, a quantidade de subproblemas identificados é que é variável, então esta métrica deverá ser acompanhada para ser possível identificar o impacto da utilização de um método contra um processo mais rígido; e **(ii) Qualidade**, onde esta será a métrica de fato mais importante, pois os avaliadores irão atribuir notas para cada uma das descrições apresentadas avaliando o quão bem descrito foi um subproblema, principalmente o quão fácil estaria esta descrição de ser interpretada por um time de desenvolvimento.

7.1.2 Planejamento do experimento

O experimento que está sendo apresentado teve como principal motivação a avaliação de dois critérios bem pontuais: **(i)** quantidade de subproblemas identificadas ($QtSbP$); e **(ii)** a qualidade da descrição destes subproblemas ($QlSbP$). Para cada um destes critérios existem duas variações: **(i)** a abordagem utilizando Icelus; e **(ii)** a abordagem *ad hoc*, onde não foi utilizado nenhum método de controle.

A principal hipótese deste experimento é a hipótese nula, em que não existirá diferença entre as abordagens utilizando Icelus ou não utilizando nenhum processo. No entanto, este experimento tenta na verdade invalidar esta hipótese inicial; esperando verificar, que de fato, a utilização de Icelus representará alguma melhora nesta atividade de quebrar um problema e na posterior descrição deste.

Desta forma, tem-se duas hipóteses nulas:

- $H0_0: QtSbP_{Icelus} \cong QtSbP_{ad hoc}$
- $H0_1: QlSbP_{Icelus} \cong QlSbP_{ad hoc}$

No entanto, outras hipóteses alternativas serão definidas para o momento em que a hipótese nula forem rejeitadas. De fato, existem duas hipóteses alternativas:

Hipótese alternativa 1 $H1_{0..1}$: A quantidade de subproblemas identificados e a qualidade da descrição destes subproblemas utilizando Icelus e uma abordagem *ad hoc* são diferentes:

- $H1_0: QtSbP_{Icelus} \neq QtSbP_{ad hoc}$
- $H1_1: QlSbP_{Icelus} \neq QlSbP_{ad hoc}$

Hipótese alternativa 2 $H_{20..1}$: A quantidade de subproblemas identificados é maior quando utilizado Icelus e a qualidade da descrição destes subproblemas serão melhores quando utilizando Icelus em uma comparação com o caso de utilizar uma abordagem *ad hoc*:

- H_{20} : $QtSbP_{Icelus} > QtSbP_{ad hoc}$
- H_{21} : $QlSbP_{Icelus} > QlSbP_{ad hoc}$

Para poder validar o impacto na quantidade e qualidade do resultado de uma abordagem de análise de problemas utilizando o método Icelus aqui proposto contra o caso de uma abordagem *ad hoc*, se faz necessário que o objeto de controle seja a quantidade de subproblemas identificados e a qualidade da descrição. Desta forma, o tempo disponível para cada uma das abordagens serão o mesmo, como também os requisitos da atividade também serão os mesmos, apenas a descrição da IA de controle de um jogo RTS, em específico, Starcraft BroodWar. Não sendo necessário avançar esta comparação no processo de desenvolvimento, por passar a inserir outras características que não podem ser controladas e passam a interferir no resultado do início do processo de análise de problemas.

Para realizar este experimento, os participantes foram divididos em dois grupos em que um utilizará a abordagem Icelus, enquanto o outro grupo irá utilizar uma abordagem *ad hoc*. Levando em consideração que quem utilizará Icelus não utilizará uma abordagem *ad hoc*, passamos a apenas ter um único fator de estudo, a diferença entre a utilização de Icelus diante a utilização de uma abordagem *ad hoc*.

7.1.3 Design do Experimento

O experimento foi composto por 14 participantes de uma turma de fim de curso denominada de Agentes Autônomos do curso de Ciência da Computação da UFPE. Os alunos que integram esta turma, já cursaram obrigatoriamente disciplinas de Engenharia de Software e IA, inclusive muitos já possuem experiência de mercado.

Estes 14 participantes foram divididos em 7 equipes com dois integrantes cada e foram divididos em: (i) 3 equipes *ad hoc*; e (ii) 4 equipes que utilizaram uma abordagem com o processo Icelus. Durante o experimento, as equipes *ad hoc* e Icelus foram isoladas, permanecendo em salas separadas para não existir a troca de informações entre equipes que utilizassem métodos diferentes.

Todos os grupos tiveram disponível o mesmo tempo para realizar a atividade solicitada, porém o grupo que iria utilizar a abordagem Icelus, teve 15 minutos de treinamento no método antes da contabilização do tempo ser iniciada. A atividade solicitada foi dividida em duas etapas: (i) identificação dos subproblemas, em que durante 20 minutos os grupos focaram apenas em listar os subproblemas que pudessem identificar na IA que controlaria Starcraft; e (ii) descrição de um subproblema, onde os grupos tiveram disponíveis mais 20 minutos para descrever um dos subproblemas identificados.

7.1.4 Validação

O grande foco deste experimento é avaliar o quanto o problema estará bem descrito após este experimento, pois partimos do princípio já existente em ES que quanto melhor a especificação de um problema, melhor será o resultado do seu desenvolvimento. Para isto, os resultados obtidos devem ser avaliados por dois avaliadores externos ao experimento, onde a avaliação deverá ser conduzida em um processo as cegas, onde os avaliadores não deverão poder identificar quais os resultados que foram utilizados Icelus ou os que não foram utilizados nenhum método.

Porém, a utilização de Icelus é baseada no uso de um formulário para descrever os subproblemas, o que torna fácil perceber quando uma descrição foi ou não construída a partir do método. Desta forma, foi necessário a realização de um procedimento que procurasse encobrir os vestígios da utilização de Icelus. Como por exemplo, capturar a descrição realizada a partir de tópicos presentes no formulário e colocá-la de maneira mais descritiva e frasal. A seguir serão apresentadas um exemplo da versão original de uma descrição utilizando o método Icelus e qual o resultado desta descrição após sua manipulação.

SBP_01 : Minerar

1. **Descrição** Avaliar quando deve ou não minerar.
2. **Objetivo** Ter mais recursos.
3. **Variáveis** Quantidade de recursos disponíveis no mapa, e quantidade de recursos já coletados..
4. **Contexto do ambiente** Se tiver sendo atacado, pare a coleta e proteja os coletores.
5. **Influência temporal** Sim, pois com o tempo os minérios e o gás acabam, e novos pontos de extração precisam ser buscados.
6. **Gatilhos** Sempre.
7. **Nível do subproblema** Operacional.

E após a manipulação do resultado obtido a partir de Icelus, tem-se:

1. A ação de Minerar deve levar em consideração a quantidade de recursos disponíveis no mapa e quantidade de recursos já coletados, sendo uma decisão operacional. Se tiver sendo atacado, pare a coleta e proteja os coletores. O tempo irá influenciar na decisão, pois com o tempo os minérios e o gás acabam, e novos pontos de extração precisam ser buscados. Esta decisão deverá ser realizada sempre e terá como consequência mais recursos.

Desta forma, reduz a possibilidade dos avaliadores identificarem quais das descrições apresentadas foram realizadas através do método Icelus ou quais foram realizadas sem a utilização de nenhum método.

7.2 Análise dos Resultados

A análise do estudo irá comparar as avaliações dos dados coletados durante o experimento para avaliar dois critérios: (i) a quantidade de subproblemas identificados; e (ii) a qualidade da descrição realizada.

Para realizar a comparação, será feito uso de um teste de t (JURISTO; MORENO, 2010). O teste de t é um teste estatístico que compara a média entre dois conjuntos de valores e avalia se a diferença entre estas médias possui alguma significância. Com o teste de t será realizada uma análise da rejeição da hipótese nula, em que não haverá diferença entre a abordagem utilizando Icelus ou *ad hoc*.

O primeiro passo é definir a média dos valores obtidos, então considerando $\chi_1, \chi_2, \dots, \chi_n$ as notas referentes a quantidade de subproblemas identificados para as amostras que utilizaram Icelus adquiridas durante o experimento e $\gamma_1, \gamma_2, \dots, \gamma_n$ as notas dadas pelos avaliadores as equipes que utilizaram uma abordagem *ad hoc*, tem-se:

$$\bar{\chi}' = \frac{1}{n} \sum_{i=1}^n \chi_i'$$

$$\bar{\gamma}' = \frac{1}{m} \sum_{i=1}^m \gamma_i'$$

De forma similar, pode se calcular as médias para as notas referentes à qualidade da descrição dadas às equipes que utilizaram Icelus ou uma abordagem *ad hoc*. Seguindo o teste de t, o próximo passo é calcular a distribuição t_0 dos valores das amostras, assim sendo, tem-se:

$$t_0 = \frac{\bar{\chi} - \bar{\gamma}}{S_p \sqrt{\frac{1}{n} + \frac{1}{m}}} \quad \text{onde} \quad S_p = \sqrt{\frac{(n-1)S_{\chi}^2 + (m-1)S_{\gamma}^2}{n+m-2}}$$

Onde S_{χ}^2 e S_{γ}^2 são as variâncias das amostras coletadas no experimento:

$$S_{\chi}^2 = \frac{(\sum_{i=1}^n \chi_i^2) - n\bar{\chi}^2}{n-1}$$

$$S_{\gamma}^2 = \frac{(\sum_{i=1}^m \gamma_i^2) - m\bar{\gamma}^2}{m-1}$$

Desta forma, o teste consiste em comparar as médias e verificar se existe alguma diferença, rejeitando a hipótese nula. Para isso, o teste checa se $|t_0| > t_{\alpha, f}$, onde $t_{\alpha, f}$ é a distribuição em uma determinada porcentagem ou nível de significância, usando f como margem de comparação, onde $f = n + m - 2$ (WOHLIN et al., 2000). Como o experimento foi realizado com poucos participantes, assumimos uma margem de comparação de 95%, o que significa que estamos trabalhando com a probabilidade de 95% dos valores serem próximos das médias.

Tabela 7.1: Quantidade de subproblemas relevantes identificados durante o experimento pelos grupos Icelus.

Grupo Icelus	Quantidade de problemas
Equipe 2	9
Equipe 3	7
Equipe 6	8
Equipe 7	15
Média	9.75

Tabela 7.2: Quantidade de subproblemas relevantes identificados durante o experimento pelos grupos com uma abordagem Ad hoc.

Grupo <i>Ad hoc</i>	Quantidade de problemas
Equipe 1	6
Equipe 4	14
Equipe 5	4
Média	8

7.3 Execução

O Experimento foi executado com uma turma de fim de graduação da UFPE. Esta turma já possuía experiência suficiente para lidar com os problemas de IA como também ter capacidade de construir documentação técnica de desenvolvimento de sistemas.

Essa turma foi formada por 14 alunos que foram divididos em 7 equipes e distribuídas em 2 grupos onde um dos grupos iria utilizar uma abordagem ad hoc, enquanto a outra teria contato com o método Icelus. Para ambos os grupos foi solicitado as mesmas atividades: (i) listar os subproblemas que fossem identificados durante 20 minutos; e (ii) descrever um dos subproblemas.

Com estas informações coletadas, elas foram submetidas a um processo como descrito previamente para eliminar possíveis traços na descrição que identificassem quais grupos cada equipe participava. Após este processo, os dados foram submetidos para uma avaliação as cegas por dois professores com experiência no campo de jogos, IA e SMA. Estes avaliadores atribuíram notas de 1 a 5 para o critério de qualidade dos dados que foram apresentados. Além disto, a lista de subproblemas foram avaliadas e a quantidade de subproblemas relevantes e não repetitivos foram contabilizadas.

7.3.1 Dados Resultantes

A quantidade de subproblemas relevantes e não repetidos estão listados nas tabelas 7.1 e 7.2

Tabela 7.3: Resultados obtidos da avaliação das quatro equipes que compuseram o grupo de participantes que utilizaram a terceira versão do método Icelus na atividade de descrição da IA de Starcraft.

Grupo Icelus	Qualitativa	
	Avaliador 1	Avaliador 2
Equipe 2	3	3
Equipe 3	2	4
Equipe 6	2	1
Equipe 7	3	4
Média	2,50	3

Tabela 7.4: Resultados obtidos da avaliação das quatro equipes que compuseram o grupo de participantes que utilizaram uma abordagem *Ad hoc* na atividade de descrição da IA de Starcraft.

Grupo Ad hoc	Qualitativa	
	Avaliador 1	Avaliador 2
Equipe 1	4	1
Equipe 4	4	5
Equipe 5	5	1
Média	4,33	2,33

As notas apresentadas pelos avaliadores com relação a qualidade das descrições podem ser visualizadas nas tabelas 7.3 e 7.4.

7.3.2 Análise Estatística

Com estes dados é possível fazer uma avaliação utilizando o *teste de t* procurando rejeitar a hipótese nula. Como descrito nas seções anteriores, o teste consiste em checar se a diferença média entre os valores obtidos é significativa ou não. Para isto, deve-se calcular o valor de $|t_0|$ da distribuição e verificar se este valor é maior do que o valor tabelado $t_{\alpha, f}$, onde $f = n + m - 2$. No caso específico deste experimento, $f = 4 + 3 - 2$ e $\alpha = 0.05$. Então deve-se comparar o valor de $|t_0|$ com o valor da distribuição de $t_{0.05, 5}$ que é 2.5706. Assim, na tabela 7.5 tem-se a avaliação das hipóteses nulas do experimento.

Desta forma, o resultado das hipóteses nulas foram para a avaliação quantitativa e qualitativa para cada avaliador, respectivamente: (i) 0.53, que por ser um valor menor do que

Tabela 7.5: Valores de $|t_0|$ para a avaliação das hipóteses nulas do experimento.

Quantitativa	Qualitativa	
	Avaliador 1	Avaliador 2
VERDADEIRA	FALSA	VERDADEIRA

o valor de $|t_0|$ calculado, resulta na rejeição da hipótese nula, que indica que não ocorreram mudanças significativas na quantidade de problemas identificados pelo método *ad hoc* e por Icelus; (ii) 4.05, que ficou a cima do valor de $|t_0|$ o que aceita a hipótese nula, identificando que existe uma diferença significativa entre a qualidade das descrições entre os grupos Icelus e os *Ad hoc* pela avaliação realizada pelo primeiro avaliador; e (iii) 0.55, que por ser um valor a baixo, igualmente ao primeiro caso, também rejeita a hipótese nula, concluindo que pela visão do segundo avaliador, não houve melhorias significativas na descrição dos problemas quando é comparado os grupos *ad hoc* e Icelus.

7.4 Conclusão

Durante este capítulo, foi apresentado o experimento realizado com a última versão de Icelus em conjunto com 14 participantes. Com os resultados obtidos foram realizados um teste estatístico para identificar se os resultados eram relevantes ou não. Como avaliação destes resultados, pode ser comprovado que estatisticamente não há diferenças comprovadas entre a quantidade de subproblemas identificados pelos grupos que utilizaram ou não o método Icelus. Porém, com relação a qualidade da descrição, um avaliador rejeitou a hipótese nula, enquanto o outro não.

Desta forma, não é possível ter muitas conclusões a partir do resultado do experimento. Principalmente devido a a ter sido realizado com um grupo muito pequeno de participantes.

8

Conclusão

Ao longo desta tese foi apresentado o método Icelus, responsável por auxiliar a equipe de desenvolvimento em analisar e melhor detalhar o problema que será solucionado. No caso específico de Icelus, o foco é o grupo de problemas que necessitam de soluções ditas inteligentes (possuem algum nível de racionalidade), mas especificamente soluções que envolverão diversas tarefas que serão solucionadas por técnicas de IA e que também envolvam diversos Agentes Inteligentes. Sendo ainda mais específico, Icelus ao longo desta tese utiliza o controle da IA de um jogo de RTS como principal suporte, tendo sido todo desenvolvido baseando-se neste caso.

Inicialmente nesta tese, foram abordadas as dificuldades em compreender um problema que necessita de uma solução inteligente. Problemas que não podem e nem devem ser abordados da mesma forma que um problema que necessita de uma solução inteligente. Porém, a comunidade de SI não é muito voltada para a especificação do problema, estando mais preocupada com a solução, seja por esta ser mais difícil de ser definida e especificada, seja devido a dificuldade de codificar uma solução. Desta forma, a primeira contribuição desta tese foi mostrar que de fato é necessário que problemas que necessitam de soluções inteligentes também sejam especificados, pois esta análise e sua posterior especificação irá facilitar a compreensão do problema e consequentemente a definição da solução e sua codificação. Porém, esta discussão pode e deve ser estendida identificando o que ainda pode ser feito para tentar organizar melhor o desenvolvimento de SMA, porém analisando até que ponto deve-se ir para não engessar esta atividade, prejudicando o seu desenvolvimento.

A segunda etapa da tese foi de fato a apresentação do método Icelus, apresentando a motivação que foi a partir da observação da dificuldade de alunos de uma disciplina conseguir desenvolver SMAs mais complexos, que envolvam a execução de diversas tarefas inteligentes e agentes. Mesmo com várias argumentações durante as aulas de acompanhamento, muitos projetos apenas lidavam com alguns problemas pontuais, não fazendo de fato um SMA que fosse capaz de lidar com a grande quantidade de subproblemas envolvidos. A tese segue com a apresentação de como o método Icelus foi evoluído, mostrando como foi definida cada uma das versões existentes e como estas foram avaliadas. Por último foram apresentados os resultados encontrados nos últimos experimentos e mostrando uma análise destes dados.

A primeira versão de Icelus foi proposta a partir desta observação da dificuldade dos alunos conseguirem lidar com um problema que envolva muitos agentes e tarefas. Esta primeira versão de Icelus passou por uma validação, onde algumas mudanças foram identificadas, permitindo que uma nova versão fosse proposta, fazendo com que este processo de evolução pudesse ser repetido algumas vezes, sempre evoluindo Icelus em cada uma destas iterações. Em cada iteração resultados foram obtidos e apresentados ao longo desta tese. Principalmente o caso do último experimento realizado com a última versão de Icelus, onde foi apresentado os resultados de uma análise estatística utilizando um teste-t. Porém, o teste que fez um comparativo entre a utilização de Icelus com a não utilização de nenhum outro método, seguindo a definição de forma ad hoc, não conseguiu de fato comprovar as hipóteses, já que o resultado de um dos avaliadores do experimento rejeitou a hipótese nula referente a qualidade da descrição, enquanto o outro aceitou a hipótese.

Com estes resultados obtidos não foi possível ter um veredicto a respeito da qualidade e capacidade do método. Sendo este um passo importante para que Icelus de fato seja avaliado com a profundidade que se faz necessária. Para isto, é necessário novos experimentos, experimentos que deverão ser conduzidos com participantes mais experientes e que, se possível, já estejam inseridos na indústria. Para esta tese, a realização de experimentos como este tipo de participantes não foi uma tarefa possível devido a dificuldade de organizá-lo com integrantes das empresas do polo tecnológico de Recife, seja devido as necessidades e urgências dos projetos de cada empresa, seja devido a pequena quantidade de integrantes dispostos a realizar este experimento. Por causa disto, a necessidade de ter conduzido os experimentos de validação que foram apresentados envolveram apenas alunos de graduação em Ciência da Computação da UFPE.

Porém, apesar de não ter sido realizado um experimento com integrantes experientes, o método foi apresentado à algumas pessoas em uma turma da pós-graduação, o que envolveu alguns poucos participantes já inseridos na indústria. A primeira impressão apresentada foi a rejeição devido a inserção de um método em uma etapa realizada superficialmente de forma ad hoc, porém focada na prototipação da solução. O que, de fato, é um comportamento esperado devido a inserção de um processo, mas que no final poderá ajudar na forma em como a tarefa já era executada.

Devido a esta observação, se faz necessário o emprego de um esforço futuro para tornar Icelus cada vez mais ágil, permitindo-o que seja mais fácil de ser aplicado e possivelmente aceito mais facilmente por profissionais já inseridos na indústria. Este esforço deve ser realizado em conjunto com um estudo aprofundado em como tornar a sequência de atividades realizadas por Icelus em atividades mais ágeis, englobando escopos menores e mais dinâmicos e que sejam mais fáceis de serem adaptadas para qualquer organização que venha utilizar o método. Como também, é possível estender Icelus para realizar melhor uma modelagem do problema abordado e, focar principalmente em como agregar a saída de Icelus como um artefato de entrada nos outros métodos existentes para SMA, mas que apenas lidam com a solução e mas não com o problema.

Outro esforço necessário é a avaliação da capacidade de Icelus em lidar com outros suportes, não apenas com jogos RTS. Para isto, uma avaliação de novos escopos devem ser realizados, como também um estudo comparativo com o escopo já abordado por Icelus, levantando as diferenças e possíveis adaptações que seriam necessárias. Com estas modificações, alguns experimentos deverão ser conduzidos para poder validá-las, sendo então possível ter um método mais abrangente e que tenha a capacidade comprovada de lidar com escopos mais abrangentes e não apenas a como é o caso atual em que é um método específico para lidar com jogos RTS.

Desta forma, Icelus ao longo desta tese foi apresentado e validado, mostrando qual seria a sua necessidade e como este evoluiu ao longo dos últimos 4 anos. Durante este período de evolução algumas validações foram realizadas e seus benefícios foram comprovados, não de forma unânime (vide os resultados do último experimento), se mostrando eficaz na análise de problemas para jogos RTS. Mas de forma mais abrangente, o método ainda necessita de melhorias que devem ser realizadas em trabalhos futuros.

Referências

- ADZIC, G. **Bridging the communication gap**: specification by example and agile acceptance testing. [S.l.]: Lulu. com, 2009.
- AMOR, M.; FUENTES, L.; VALLECILLO, A. Bridging the gap between agent-oriented design and implementation using MDA. In: **Agent-Oriented Software Engineering V**. [S.l.]: Springer, 2005. p.93–108.
- BECK, K. **Extreme programming explained**: embrace change. [S.l.]: Addison-Wesley Professional, 2000.
- BECK, K. **Test-driven development**: by example. [S.l.]: Addison-Wesley Professional, 2003.
- BIIRGISSER, P.; CLAUSEN, M.; SHOKROLLAHI, M. A. Algebraic complexity theory. **Grundlehren der mathematischen Wissenschaften**, [S.l.], v.315, 1997.
- BLACKBURN, S. **The Oxford dictionary of philosophy**. [S.l.: s.n.], 2008.
- BOOCH, G. **Object solutions managing the object**. [S.l.: s.n.], 1996.
- BRESCIANI, P. et al. TROPOS: an agent-oriented software development methodology. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.8, n.3, p.203–236, May 2004.
- BREUKER, J. A.; WIELINGA, B. J. **Techniques for knowledge elicitation and analysis**. [S.l.]: University of Amsterdam, Department of Social Science Informatics and Laboratory for Experimental Psychology, 1984.
- BREUKER, J. et al. **Interpretation of verbal data for knowledge acquisition**. [S.l.]: University of Amsterdam, 1984.
- BREUKER, J.; VELDE, W. Van de. **CommonKADS library for expertise modelling**: reusable problem solving components. [S.l.]: IOS press, 1994. v.21.
- BREUKER, J.; WIELINGA, B. J. **Analysis techniques for knowledge based systems**. [S.l.]: University of Amsterdam, 1983.
- BURO, M.; FURTAK, T. RTS games and real-time AI research. In: BEHAVIOR REPRESENTATION IN MODELING AND SIMULATION CONFERENCE (BRIMS). **Proceedings...** [S.l.: s.n.], 2004. v.6370.
- CARRERA BARROSO, A.; SOLITARIO, J. J.; IGLESIAS FERNANDEZ, C. A. Behaviour driven development for multi-agent systems. In: INFRASTRUCTURES AND TOOLS FOR MULTIAGENT SYSTEMS (ITMAS-2012). **Anais...** [S.l.: s.n.].
- CERNUZZI, L.; COSSENTINO, M.; ZAMBONELLI, F. Process models for agent-based development. **Eng. Appl. of AI**, [S.l.], v.18, n.2, p.205–222, 2005.
- CHURCHILL, D.; BURO, M. Build Order Optimization in StarCraft. In: AIIDE. **Anais...** [S.l.: s.n.], 2011.

- CLANCEY, W. J. Heuristic classification. **Artificial intelligence**, [S.l.], v.27, n.3, p.289–350, 1985.
- CLYNCH, N.; COLLIER, R. Sadaam: software agent development-an agile methodology. In: WORKSHOP OF LANGUAGES, METHODOLOGIES, AND DEVELOPMENT TOOLS FOR MULTI-AGENT SYSTEMS (LADS'007), DURHAM, UK. **Proceedings...** [S.l.: s.n.], 2007.
- COMBLEY, R. **Cambridge Business English Dictionary**. [S.l.]: Cambridge University Press, 2011.
- DANIELSIEK, H. et al. Intelligent moving of groups in real-time strategy games. In: CIG. **Anais...** IEEE, 2008. p.71–78.
- DASKALAKIS, C.; GOLDBERG, P. W.; PAPADIMITRIOU, C. H. The complexity of computing a Nash equilibrium. **SIAM Journal on Computing**, [S.l.], v.39, n.1, p.195–259, 2009.
- DELOACH, S. A. **Analysis and Design using MaSE and agentTool**. [S.l.]: DTIC Document, 2001.
- EL-NASR, M. S. Interaction, narrative, and drama: creating an adaptive interactive narrative using performance arts theories. **Interaction Studies**, [S.l.], v.8, n.2, p.209–240, 2007.
- EMPIRES, A. of. **Microsoft**. 1997.
- ERIKSSON, H.-E.; PENKER, M. Business modeling with UML. **Business Patterns at Work**, John Wiley & Sons, New York, USA, [S.l.], 2000.
- EROL, K. Hierarchical task network planning: formalization, analysis, and implementation. In: IEEE. **Anais...** [S.l.: s.n.], 1996.
- EROL, K.; HENDLER, J. A.; NAU, D. S. UMCP: a sound and complete procedure for hierarchical task-network planning. In: AIPS. **Anais...** [S.l.: s.n.], 1994. v.94, p.249–254.
- EVSLIN, B. **Gods, Demigods and Demons: a handbook of greek mythology**. [S.l.]: Harvard Common Press, 2006.
- FIKES, R. E.; NILSSON, N. J. STRIPS: a new approach to the application of theorem proving to problem solving. **Artificial intelligence**, [S.l.], v.2, n.3, p.189–208, 1972.
- FOWLER, M.; HIGHSMITH, J. The agile manifesto. **Software Development**, [S.l.], v.9, n.8, p.28–35, 2001.
- GIORGINI, P.; MYLOPOULOS, J.; SEBASTIANI, R. Goal-oriented requirements analysis and reasoning in the tropos methodology. **Engineering Applications of Artificial Intelligence**, [S.l.], v.18, n.2, p.159–171, 2005.
- GOTEL, O. et al. Traceability fundamentals. In: **Software and Systems Traceability**. [S.l.]: Springer, 2012. p.3–22.
- HALFORD, G. S. et al. How many variables can humans process? **Psychological science**, [S.l.], v.16, n.1, p.70–76, 2005.

HERNON, P.; SCHWARTZ, C. What is a problem statement? **Library & Information Science Research**, [S.l.], v.29, n.3, p.307–309, 2007.

HILLIER, F. S.; LIEBERMAN, G. J. **Introduction to operations research**. [S.l.]: Tata McGraw-Hill Education, 2001.

HUMPHREY, W. S. Managing the software process (Hardcover). **Addison-Wesley Professional. Humphrey, WS, & Curtis, B.(1991). Comments on a critical look at software capability evaluations. Software, IEEE**, [S.l.], v.8, n.4, p.42–46, 1989.

IEEE. **IEEE Standards Collection: software engineering**. 1993.

IGLESIAS, C. A. et al. Analysis and design of multiagent systems using MAS-CommonKADS. In: **Anais...** Springer-Verlag, 1998. p.313–326.

IGLESIAS, C.; GARRIJO, M.; GONZALEZ, J. A Survey of Agent-Oriented Methodologies. In: INTERNATIONAL WORKSHOP ON INTELLIGENT AGENTS V : AGENT THEORIES, ARCHITECTURES, AND LANGUAGES (ATAL-98), 5. **Proceedings...** Springer-Verlag: Heidelberg: Germany, 1999. v.1555, p.317–330.

JADON, S.; SINGHAL, A.; DAWN, S. Military Simulator-A Case Study of Behaviour Tree and Unity based architecture. **arXiv preprint arXiv:1405.7944**, [S.l.], 2014.

JANZEN, D.; SAIEDIAN, H. Test-driven development: concepts, taxonomy, and future direction. **Computer**, [S.l.], n.9, p.43–50, 2005.

JURISTO, N.; MORENO, A. M. **Basics of software engineering experimentation**. [S.l.]: Springer Publishing Company, Incorporated, 2010.

KHAN, A. I.; QURASHI, R. J.; KHAN, U. A. A Comprehensive Study of Commonly Practiced Heavy and Light Weight Software Methodologies. **CoRR**, [S.l.], v.abs/1111.3001, 2011.

KRUCHTEN, P. **The Rational Unified Process: an introduction**. 3.ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.

LAIRD, J. E. It knows what you're going to do: adding anticipation to a quakebot. In: AUTONOMOUS AGENTS. **Proceedings...** [S.l.: s.n.], 2001. p.385–392.

LUCAS, S. M. et al. **Artificial and Computational Intelligence in Games**. [S.l.]: Schloss Dagstuhl-Leibniz-Zentrum für Informatik GmbH, 2013.

MARTIN, R. C. **Agile software development: principles, patterns, and practices**. [S.l.]: Prentice Hall PTR, 2003.

MCLEOD, L.; MACDONELL, S. G. Factors that affect software systems development project outcomes: a survey of research. **ACM Computing Surveys (CSUR)**, [S.l.], v.43, n.4, p.24, 2011.

MICHAEL, D. R.; CHEN, S. L. **Serious games: games that educate, train, and inform**. [S.l.]: Muska & Lipman/Premier-Trade, 2005.

MORANDINI, M. et al. Tool-supported development with tropos: the conference management system case study. In: **Agent-Oriented Software Engineering VIII**. [S.l.]: Springer, 2008. p.182–196.

NAUR, P.; RANDELL, B. Software Engineering: report of a conference sponsored by the nato science committee, garmisch, germany, 7-11 oct. 1968, brussels, scientific affairs division, nato. In: OF THE FIFTH INTERNATIONAL CONFERENCE ON . **Anais...** [S.l.: s.n.], 1969.

ORKIN, J. Applying goal-oriented action planning to games. **AI Game Programming Wisdom**, [S.l.], v.2, n.2004, p.217–227, 2004.

OWEN, M.; RAJ, J. BPMN and Business Process Management. Introduction to the New Business Process Modeling Standard. **Popkin Software 2003**, [S.l.], 2003.

PADGHAM, L.; WINIKOFF, M. Prometheus: a methodology for developing intelligent agents. In: AOSE. **Anais...** Springer, 2002. p.174–185. (Lecture Notes in Computer Science, v.2585).

PARTRIDGE, D. Engineering artificial intelligence software. **Artificial Intelligence Review**, [S.l.], v.1, n.1, p.27–41, 1986.

PARTRIDGE, D.; WILKS, Y. Does AI have a methodology which is different from software engineering? **Artificial intelligence review**, [S.l.], v.1, n.2, p.111–120, 1987.

PÉREZ, A. U. Multi-Reactive Planning for Real-Time Strategy Games. **M. Sc. Report. Universit Autònoma de Barcelona, Spain**, [S.l.], 2011.

PRESSMAN, R. S. **Software Engineering**: a practitioner's approach. 5th.ed. [S.l.]: McGraw-Hill Higher Education, 2001.

ROBINSON, D. J. **A component based approach to agent specification**. [S.l.]: DTIC Document, 2000.

ROCHA, F. et al. Towards a Method for Modeling AI in Games: a case study on real-time strategy games. In: XI SBGAMES. **Proceedings...** [S.l.: s.n.], 2012. p.41–48.

RUSSELL, S. J.; NORVIG, P. **Artificial Intelligence**: a modern approach. 2.ed. [S.l.]: Pearson Education, 2003.

SCHWABER, K. **Agile project management with Scrum**. [S.l.]: Microsoft Press, 2004.

SHARIFI, A.; ZHAO, R.; SZAFRON, D. Learning Companion Behaviors Using Reinforcement Learning in Games. In: AIIDE. **Anais...** [S.l.: s.n.], 2010.

SHOHAM, Y. Agent-oriented programming. **Artif. Intell.**, Essex, UK, v.60, n.1, p.51–92, Mar. 1993.

SNOWDON, W.; SCHULTZ, J.; SWINBURN, B. Problem and solution trees: a practical approach for identifying potential interventions to improve population nutrition. **Health promotion international**, [S.l.], v.23, n.4, p.345–353, 2008.

SOMMERVILLE, I. Artificial Intelligence and Systems Engineering. In: PROSPECTS FOR ARTIFICIAL INTELLIGENCE: PROCEEDINGS OF AISB93, THE NINTH BIENNIAL CONFERENCE OF THE SOCIETY FOR THE STUDY OF ARTIFICIAL INTELLIGENCE AND SIMULATION OF BEHAVIOUR, 29 MARCH-2 APRIL 1993, THE UNIVERSITY OF BIRMINGHAM. **Anais...** [S.l.: s.n.], 1993. p.48.

SOMMERVILLE, I. **Software engineering (5th ed.)**. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1995.

- SOTTILARE, R. A.; GILBERT, S. **Considerations for adaptive tutoring within serious games**: authoring cognitive models and game interfaces. [S.l.]: DTIC Document, 2011.
- SPERANZA, M. G. Association of European Operational Research Societies. **Wiley Encyclopedia of Operations Research and Management Science**, [S.l.], 2012.
- STARCRAFT. **Blizzard**. 1998.
- STEVENSON, W. J.; SUM, C. C. **Operations management**. [S.l.]: McGraw-Hill/Irwin Boston, MA, 2009. v.8.
- STUDER, R.; BENJAMINS, V. R.; FENSEL, D. Knowledge engineering: principles and methods. **Data & knowledge engineering**, [S.l.], v.25, n.1, p.161–197, 1998.
- TAYLOR, P. E.; HUXLEY, S. J. A break from tradition for the San Francisco police: patrol officer scheduling using an optimization-based decision support system. **Interfaces**, [S.l.], v.19, n.1, p.4–24, 1989.
- TURING, A. M. **Computing Machinery and Intelligence**. One of the most influential papers in the history of the cognitive sciences: <http://cogsci.umn.edu/millennium/final.html>.
- VAN DER AALST, W. M. P.; HOFSTEDE, A. H. M. T.; WESKE, M. Business process management: a survey. In: BUSINESS PROCESS MANAGEMENT, 2003., Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2003. p.1–12. (BPM'03).
- VAN VLIET, H.; VAN VLIET, H.; VAN VLIET, J. **Software engineering**: principles and practice. [S.l.]: Wiley, 1993. v.3.
- WEBER, B. G.; MATEAS, M. A data mining approach to strategy prediction. In: COMPUTATIONAL INTELLIGENCE AND GAMES, 2009. CIG 2009. IEEE SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2009. p.140–147.
- WEBER, B. G.; MATEAS, M.; JHALA, A. Applying Goal-Driven Autonomy to StarCraft. In: AIIDE. **Anais...** [S.l.: s.n.], 2010.
- WEISS, G. (Ed.). **Multiagent systems**: a modern approach to distributed artificial intelligence. Cambridge, MA, USA: MIT Press, 1999.
- WHIMBEY, A.; LOCHHEAD, J.; NARODE, R. **Problem solving & comprehension**. [S.l.]: Routledge, 2013.
- WHITE, S. A. Introduction to BPMN. 2004. **IBM Corporation**, [S.l.], 2004.
- WIELINGA, B. J.; SCHREIBER, A. T.; BREUKER, J. A. KADS: a modelling approach to knowledge engineering. **Knowledge acquisition**, [S.l.], v.4, n.1, p.5–53, 1992.
- WINSTON, W. L.; GOLDBERG, J. B. **Operations research**: applications and algorithms. [S.l.]: Duxbury press Boston, 2004. v.3.
- WOHLIN, C. et al. **Experimentation in software engineering**: an introduction. 2000. [S.l.]: Kluwer Academic Publishers, 2000.
- WOOLDRIDGE, M. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

WOOLDRIDGE, M.; CIANCARINI, P. Agent-oriented software engineering: the state of the art. In: FIRST INTERNATIONAL WORKSHOP, AOSE 2000 ON AGENT-ORIENTED SOFTWARE ENGINEERING, Secaucus, NJ, USA. **Anais...** Springer-Verlag New York: Inc., 2001. p.1–28.

WOOLDRIDGE, M.; JENNINGS, N. R.; KINNY, D. The Gaia Methodology for Agent-Oriented Analysis and Design. **Autonomous Agents and Multi-Agent Systems**, Hingham, MA, USA, v.3, n.3, p.285–312, Sept. 2000.

YU, E. S.; MYLOPOULOS, J. Understanding “why” in software process modelling, analysis, and design. In: SOFTWARE ENGINEERING, 16. **Proceedings...** [S.l.: s.n.], 1994. p.159–168.