**UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL**
**INSTITUTO DE INFORMÁTICA**
**PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO**

ELDER RIZZON SANTOS

# Creative Agency

Thesis presented in partial fulfillment of the requirements for the degree of Doctor of Computer Science

Prof. Dr. Rosa Maria Vicari
Advisor

Porto Alegre, December, 2010.

**CIP – CATALOGAÇÃO NA PUBLICAÇÃO**

Santos, Elder Rizzon

Creative Agency [manuscript] / Elder Rizzon Santos. – Porto Alegre: PPGC da UFRGS, 2010.

131 f.:il.

Advisor: Prof.ª Dr.ª Rosa Maria Vicari.

Thesis (Ph.D.) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação. Porto Alegre, BR – RS, 2010.

1. Computational creativity. 2. Intelligent agents. 3. Concept blending. I. Vicari, Rosa Maria. II. Título.

# AGRADECIMENTOS

Imagino que, daqui alguns anos, quando estiver contando sobre a experiência do doutorado os detalhes (trabalho, estudos, sucessos e frustações) irão desaparecer aos poucos e os bastidores humanos tornar-se-ão o personagem principal do meu relato. Muito além de uma tese e publicações, o maior impacto do doutorado é a formação do doutorando e nesta formação, pouca coisa é realizada sem a participação de outras pessoas. Em quatro anos de doutorado, bem, há muita gente nos bastidores. Com certeza, todos merecem meu muito obrigado, e para quem ler, lembrar de sua participação nos bastidores e não encontrar seu nome, tenha certeza que merece sim meus sinceros agradecimentos.

Minha querida orientadora e mãe acadêmica, Rosa Vicari, merece muitos agradecimentos pelo apoio, puxões de orelha, brainstorms em aeroportos e todas as oportunidades para eu me desenvolver e desenvolver também minha pesquisa.

Meus pais, Vilmar e Suely, possuem minha eterna gratidão pela atenção, amor e apoio incondicionais. Desde sempre ao meu lado, dedicando suas preciosas vidas para a minha educação e minha felicidade. Nenhuma agradecimento chega aos pés de toda a dedicação deles, assim, em um singelo ato de gratidão, dedico esta tese aos meus pais. Outra pessoa que está há muito tempo nos bastidores é meu irmão Marcos, a quem eu agradeço pelo exemplo de convicção e coragem para ir em direção ao que lhe faz feliz.

Tão importante quanto quem sempre esteve ao meu lado, é quem eu encontrei, me encontrou e aceitou me ajudar para que eu me encontre sempre mais. Sou privilegiado por ter uma pessoa assim em minha vida, minha querida Ciça, sempre me apoiando, muitas vezes até puxando para frente e sempre minha inspiração, fundamental para todos os momentos do doutorado, especialmente os mais difíceis. Obrigado por tudo. Agradeço também à minha segunda família, Maria Alice, Vini, Luis Fernando e Simone. A companhia e carinho de vocês é inestimável.

Os anos de trabalho na UFRGS, tanto na Informática quanto no CINTED, oportunizaram conhecer muita gente do bem, em especial meu amigo e parceiro de incontáveis tradicionais mates uruguaios, Tiago Primo, sempre disposto para bate-papo, um brainstorm ou um jogo de tênis. Agradeço também aos meus colegas e amigos Evandro, Luciano, Alexandre R., Marta, Eliane, Gilleanes, Preto, Alexandro, Kieling, Michelle e Bernardo pelos almoços descontraídos e ricas trocas de experiências.

Todos os meus amigos e companheiros do Método DeRose contribuíram imensamente para meu aprimoramento pessoal e, portanto, para o sadio desenvolvimento do meu doutorado. Em especial agradeço ao meu amigo e monitor Fabiano Gomes pelo seu exemplo de dedicação incansável e excelentes aulas práticas. Agradeço também aos meus amigos Márcio, Aline, Tanara, Pati, Maile, Maurício, Mateus, Toninho, Mônica, Carlinha, Fernanda F., Tiago e Odilon pelo companheirismo sempre alegre e descontraído.

Meus agradecimentos também à UFRGS e a todos os professores e funcionários do PPGC e CINTED, bem como às agências de fomento à pesquisa Brasileiras (CAPES, CNPq e FINEP). Finalmente, agradeço aos professores que compuseram a banca de avaliação da minha tese, profs. Rafael Bordini, Jomi Hübner, Renato Fileto, Antônio Rocha Costa e Michael Móra pelas preciosas críticas construtivas.

# CONTENTS

# LIST OF ABBREVIATIONS

| | |
|---|---|
| AI | Artificial Intelligence |
| API | Application Programming Interface |
| BDI | Belief, Desire, Intention |
| CB | Concept Blending |
| CC | Computational Creativity |
| CI | Conceptual Integration |
| DL | Description Logics |
| FOAF | Friend Of A Friend |
| HCI | Human-Computer Interface |
| IMS | Instructional Management Systems |
| JNI | Java Native Interface |
| JSON | Java Simplified Object Notation |
| KR&R | Knowledge Representation and Reasoning |
| LIP | Learner Information Package |
| LD | Learning Design |
| MAPL | Multi-Agent Planning Language |
| OBAA | Agent-Based Learning Objects |
| OWL | Web Ontology Language |
| PRS | Procedural Reasoning System |
| REST | Representational State Transfer |
| SOS | Structural Operational Semantics |
| STRIPS | Stanford Research Institute Problem Solver |

# FIGURE INDEX

# TABLE INDEX

# ABSTRACT

This PhD thesis describes an interdisciplinary research on computational creativity and cognitive agents. Our motivation to integrate these two areas is to study the human skill that uses previous experiences and knowledge to solve unpredicted problems and situations. Imbued by that motivation, our purpose is to improve the applicability of the agent's knowledge, inspired in the way that we humans understand and experience the world.

Our approach towards that research view is to adopt theories and results from cognitive and neural sciences as the grounding to a computational model of agents capable of acting creatively. Thus, we adopt the concept blending theory (FAUCONNIER; TURNER, 1998) – that originated from cognitive linguistics and theory of the mind – as the grounding of our model. Therefore, our proposal of creative agents integrates an implementation of concept blending into a BDI structure. In concrete terms, we use Jason's implementation of AgentSpeak to manipulate the agent's theoretical (beliefs) and practical (desires and intentions) reasoning.

Hence, the main topic of study of this research is the utilization of concept blending in a structure of intelligent agents. Consequently, we observe our contributions under two perspectives. Regarding computational creativity, we specify a model for concept blending that explicitly defines rules to represent a blending typology. Furthermore, integrating a BDI structure to the model allows the automated construction of inputs and domain information to feed the blending process.

Focusing on agents, our contribution is on the process of creative reasoning applied to supply alternative ways to use practical and theoretical knowledge. Given the blending specification defined here, it is possible to integrate different adaptation strategies to handle intention failure or other adaptation scenarios. Another feature is the possibility to work with different knowledge representations given its descriptive logics (using the OWL language) definition. The blending specification is also applied to model the reasoning of an educational recommender system.

Finally, the defined model represents an initial work towards a cognition model where blending, agency and other cognitive operations (e.g. learning) interact together to simulate different features of the human thinking.

**Keywords:** Computational creativity, intelligent agents, concept blending.

## Agência Criativa

# RESUMO

A presente tese de doutorado descreve uma pesquisa interdisciplinar nas áreas de criatividade computacional e agentes cognitivos. A motivação para a integração dessas áreas é o estudo da habilidade humana de utilizar suas experiências prévias e conhecimento geral para resolver problemas e lidar com situações a partir do momento em que as mesmas são apresentadas. Imbuídos dessa motivação, nosso propósito é ampliar a utilização do conhecimento de agentes, inspirado na forma como, nós, humanos entendemos e vivenciamos o mundo.

Nossa abordagem para concretizar essa visão de pesquisa é adotar teorias e resultados das ciências cognitivas e neurociências como fundamentação para um modelo computacional de agentes capazes de atuar criativamente. Assim sendo, adotamos a teoria do *concept blending* (fusão conceitual – tradução do autor) (FAUCONNIER; TURNER, 1998), advinda da lingüística cognitiva e teoria da mente como a fundação de nosso modelo. O modelo de agentes criativos proposto integra uma implementação da fusão conceitual em uma estrutura BDI. Concretamente, utilizamos a implementação da linguagem AgentSpeak fornecida pelo framework Jason, para manipular o raciocínio teórico (crenças) e prático (desejos, planos e intenções) do agente.

Logo, o objeto principal de estudo desta tese é a utilização da fusão conceitual em uma estrutura de agentes inteligentes visando contribuições em criatividade computacional e agentes. Considerando a área da criatividade computacional, especificamos um modelo da fusão conceitual que define explicitamente as regras necessárias para representar uma tipologia da fusão. Ademais, a integração de uma estrutura de agentes BDI ao modelo possibilita a construção automatizada das entradas e de informações de domínio para utilizar o processo de fusão.

Focando na área de agentes, nossa contribuição é caracterizada pela aplicação do processo de raciocínio criativo para fornecer alternativas de uso do conhecimento prático e teórico. Dada a especificação da fusão aqui apresentada, é possível integrar diferentes estratégias de adaptação para lidar com a falha de intenções ou outras situações que requerem adaptação. Outra funcionalidade é a capacidade de utilizar diferentes representações de conhecimento, assumindo a disponibilidade de uma definição descritiva (na linguagem OWL) da representação. O modelo de fusão conceitual também é aplicado na modelagem do raciocínio de um sistema de recomendação educacional.

Finalmente, nosso modelo de fusão representa um trabalho inicial em direção a um modelo cognitivo no qual fusão, agência e outras funções cognitivas (e.g. aprendizagem) interagem para simular diferentes funcionalidades do pensamento humano.

**Palavras-Chave:** Criatividade computacional, agentes inteligentes, fusão conceitual.

# 1 INTRODUCTION

Creative agency. As our title suggests, we are interested in applying creativity to an agent structure. However, our title can also be read backwards (agent-based creativity) and will still make sense, since the utilization of an agent structure to creative reasoning is important on its own. Agency, a conceptualization from philosophy that generally captures the human ability to decide and act autonomously, is adopted in social sciences, psychology, cognitive sciences, biology, economy and computer sciences. Inside Computer Science, agency is a research field that belongs to Artificial Intelligence (AI). Under a computational perspective, agency challenges researchers to provide theories and models of autonomy that can be executed by computers.

Creativity is related to the process undertaken by humans to generate new concepts and artifacts – in the broadest spectrum of applicability. As a process, it is mostly studied by neurosciences, cognitive sciences, psychology and philosophy. As a property (attributed to a certain thing, concrete or abstract), it is typically considered by social sciences and art appreciation. Both the process and the property views are subject of study by computational creativity researchers.

As most AI sub-fields, computational agency and computational creativity concentrate substantial research effort on the multidisciplinary links with different areas, often resulting in simulation models and insights useful for all the areas involved. Our research agenda is motivated by the idea that AI models should serve as means to aid on the challenge of understanding how the human mind works. We are especially interested on the human ability to use previous experiences and knowledge to respond to novel situations, creating new possibilities and solutions as a problem is presented.

Our approach to study this behavior is by modeling creativity as reasoning integrated to an agent structure. Considering creativity as a process modeled by computational reasoning, we are able to position it inside a traditional AI perspective of knowledge representation and reasoning (HARMELEN; LIFSCHITZ; PORTER, 2007). Most works from AI can be categorized in terms of how they represent and reason over their knowledge (also seen as static and dynamic aspects). Furthermore, each one of those categories is usually the result of studies on how we use knowledge in our minds.

For instance, early works on AI were mostly based on philosophy and mathematics, yielding one of the most successful approaches of the field: logic-based reasoning (HOFWEBER, 2009; COLMERAUER, 1985; KOWALSKI, 1986). Today, works on logic and its applications still provide important results for the field and computation in general. Recent works on logic attempt to improve the performance of reasoners while keeping or enhancing expressiveness. Beyond that, there are the works that use logic-based formalisms to represent modal, temporal, ontological, probabilistic and spatial knowledge (to name a few).

Understanding that not everything can be modeled with logic, or that logic is not adequate for every kind of knowledge, hybrid or totally different paradigms for knowledge representation were developed. For example, frames (MINSKY, 1974) and semantic networks (QUILIAN, 1968) were grounded on psychology discoveries and developed to naturally accommodate hierarchical and classification inferences. On the other hand, Bayesian networks (PEARL, 1985), which originated from statistics, received computational models that represent cause-effect and propagation of evidences in large chains of knowledge. In direct opposition to all symbolic approaches, neural networks (MCCULLOCH; PITTS, 1943) are the most prominent of sub-symbolic (connectionist) representations. Following inspiration from biology (brain studies) neural networks provide several ways to implement automated and semi-automated learning.

In this project, we follow developments from cognitive sciences on Concept Blending (CB) (FAUCONNIER; TURNER, 1998), considered to be an innate sub-conscious human skill that integrates knowledge creating novel conceptualizations. Inside computational creativity, the utilization of CB theory to specify the creative reasoning places our work under general creativity models. Specific models are developed to simulate creative activities such as composing music (MARTINS, 2004; MARTINS; MIRANDA, 2006; PEARCE; MÜLLENSIEFEN; WIGGINS, 2008), painting (COLTON, 2008), writing poetry (VEALE; HAO, 2008; HERVÁS, R. et al. 2007) and jokes (BINSTEAD, K. et al. 2006). Under general creative models, the knowledge gap we study is the consideration of practical knowledge, along with theoretical during creative reasoning. Thus, in more theoretical terms, we are trying to integrate creative reasoning inside a broader cognitive structure, allowing creativity to work with intentionality. To achieve this in a computational model, we propose the use of constructs supplied by agent theories and languages. This is what we meant by reading the title of this project backwards: agent-based creativity.

We limit our agency scope to cognitive agents implementing intentional systems. Thus, in a theoretical perspective, we consider autonomy as a property resulting from several cognitive processes and, here, we focus on the aspect of intentionality. Specifically, we adopt Bratman's Belief Desire and Intention (BDI) approach as an intentional theory to ground this work. Bratman (1987) can be considered as one of the most influential works on autonomous agents. His BDI theory has its roots on philosophy of the mind and folk psychology. BDI theory represents the agent's knowledge about the world as beliefs, desires represent how the agent wants the world to be, and intentions are desires that the agent is committed to achieve. This paradigm to specify autonomy inspired the development of several agent architectures (KUMAR; SHAPIRO, 1994; MORLEY; MYERS, 2004; D'INVERNO, M. et al. 2004), languages (DASTANI, M. et al.. 2003; RAO; GEORGEFF 1991; RAO, 1996) and also variations of the theory itself (GOVERNATORI; ROTOLO, 2008; CHOLVY, 2004; BROERSEN, J. et al. 2001). Most of these works also consider resource bounding agency, which was introduced by Bratman (1988). The philosophical and psychological grounding of the theory, together with a compatible practical (but restrictive) architecture are the main reasons for the success of the approach.

The main approach for the design of resource bounded BDI agents – Procedural Reasoning Systems (PRS) (GEORGEFF; LANSKY, 1987; RAO; GEORGEFF, 1991) use the abstraction of plans, which allow the expression of procedural knowledge inside agents (usually programmed with declarative languages). Plans specify pre-defined

recipes to handle particular world configurations (defined by the plan's pre-conditions). In this case, plans work as heuristics to reduce the search for a viable option, allowing the agent to perform practical reasoning, the reasoning towards action (WOOLDRIDGE, 1995), in a timely fashion. However, at the same time that resource-bounding action allows the agent to promptly interact with its environment, it also constrains the possibilities of action. Wooldridge (1995, 2000) argues that achieving a balance between reasoning and acting is one the main challenges of agent development and research.

Current approaches to deal with that issue follow different perspectives, such as use of emotions (JIANG; VIDAL; HUHNS, 2007; STEUNEBRINK; DASTANI; MEYER, 2007), norms (DASTANI; TINNEMEIER; MEYER, 2009; GANGEMI, 2008; CONTE; ANDRIGHETTO; CAMPENNÍ, 2009) and learning (SUBAGDJA; SONENBERG; RAHWAN, 2009; FUJITA, 2009; SEN; AIRIAU, 2007; SHOHAM; POWERS; GRENAGER, 2007; STONE, 2007). We limit our scope to approaches that increase the utilization of an agent's knowledge, without specifying an application context. Specifically, we are focused on approaches that allow the agent to adapt to situations that were not pre-programmed in the plan library. Thus, we refer to works on planning and agent learning.

Finally, our consideration of creative agency regards the use of creative reasoning to improve the applicability of the agent's knowledge (adaptation), inspired in the way that humans understand and experience the world.

## 1.1 Research Problem

Summarizing our introductory argument, the motivation for this work lies on the impact of computational models to better understand human intelligence, specifically our ability to use our experience to handle new situations. Our research is located on the intersection between computational creativity and cognitive agency. Inside computational creativity, this research is positioned on computational models of human creativity, following the CB theory. On the agent side, we place our work under BDI architectures and languages. Imbued by our motivation and context, we characterize our research question as follows:

*Q1. How can creativity support intentionality?*

This main research question is unfolded into two intermediary questions:

*Q1.1. How can creativity be computationally modeled in order to produce theoretical and practical knowledge?*

*Q1.2 How can creativity support the utilization of knowledge as a means to adapt to unforeseen situations?*

## 1.2 Research Goals

From our research questions we propose the following goal for our project:

*G1. Propose a computational model for creativity.*

*G2. Apply the model from G1 as a means to support an intentionality model.*

Given the intermediary questions we establish the following specific goals:

*G1.1. Specify a representation of the blending theory considering both theoretical and practical knowledge.*

*G2.1. Propose the utilization of the creative representation defined in G1.1 as an adaptation mechanism to support agent-based intentionality.*

## 1.3 Research Strategy

In this research project we are following the design research (also known as constructive research) strategy. According to Lukka (2003), design research is a research procedure for producing innovative constructions, intended to solve problems in the real world, and, thus, making a contribution to the field in which it is applied. Kasanen, Kari and Arto (1993) cite mathematical algorithms and theorems, artificial languages (Braille's alphabet, Morse alphabet, computer languages) and new pharmaceuticals as examples of the output of a design research approach.

Hence, the new construction – central notion of the approach – is an abstract notion that can be concretized in many different ways. All human artifacts (e.g. models, diagrams, plans, commercial products and information systems) are considered to be constructions. Such constructions are characterized by the fact that they were developed, not discovered (LUKKA, 2003).

March and Smith (1995) propose four kinds of outputs for the constructive approach: consctructs, models, methods and instantiations. In this context, constructs are the conceptual vocabulary of problem domain. They arise during the conceptualization of the problem and are refined throughout the design cycle. Related to constructs, a model specifies the relationships among them. Methods provide a way to manipulate the constructs aiming at realizing a solution model. Finally, an instantiation concretizes constructs, models and methods in an environment.

Vaishnavi and Kuechler (2004) present another output called better theories. Also perceived as theory building, better theories relates to the output that improve a method or instantiation already established, such as software engineering communities devoted to improve software maintenance and reuse. An additional way to view the better theories conceptualization is through the research artifact that, when being evaluated, brings new understanding to previously established relations among constructs. Carrol and Kellog (1989) exemplify this situation on the Human-Computer Interface (HCI) where artifacts themselves constitute the most effective medium for theory development in the field. Figure 1 illustrates the relation among different levels of abstraction in research artifacts and the design research output terminology.

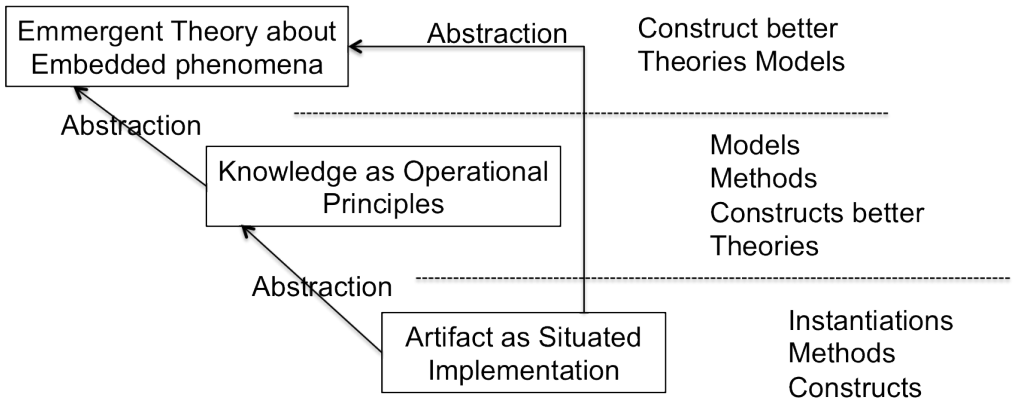

Figure 1 Outputs of design research (adapted from Vaishnavi and Kuechler, 2004)

In terms of process, Kazanen, Kari and Arto (1993) summarize the main phases of the design research approach:

1. Find a practically relevant problem which also has research potential.

2. Obtain a general and comprehensive understanding of the topic.

3. Innovate, i.e., construct a solution idea.

4. Demonstrate that the solution works.

5. Show the theoretical connections and the research contribution of the solution concept.

6. Examine the scope of applicability of the solution.

Those authors also argue that these phases may be applied in different orders, varying from case to case. Lukka (2003) goes further on the process of design research, describing the phases identified in (KAZANEN, KARI; ARTO, 1993). In addition, Lukka (2003) presents a diagram (illustrated by Figure 2) summarizing the central aspects of the constructive research approach.

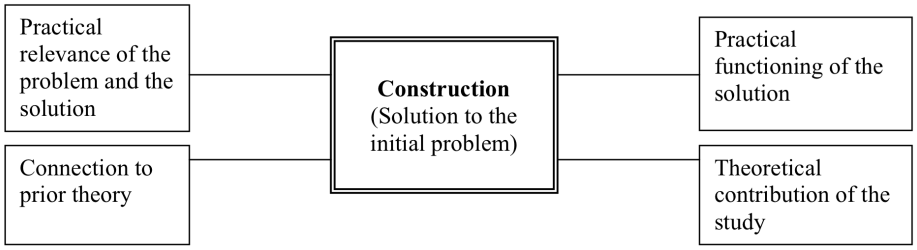


Figure 2 The central elements of the constructive research approach (LUKKA, 2003)

Positioning design research in a broad scope of research, it can be viewed as a research strategy, since it provides a structure for the research work, guiding the way that empirical evidence is collected and analyzed. Furthermore, design research is more aligned to pragmatic research since the quality of the constructed knowledge is evaluated in terms of its utility. Figure 3 presents the research onion, proposed by Saunders, Thornhill and Lewis (2006), with slight modifications to add design research and a few extra philosophies.

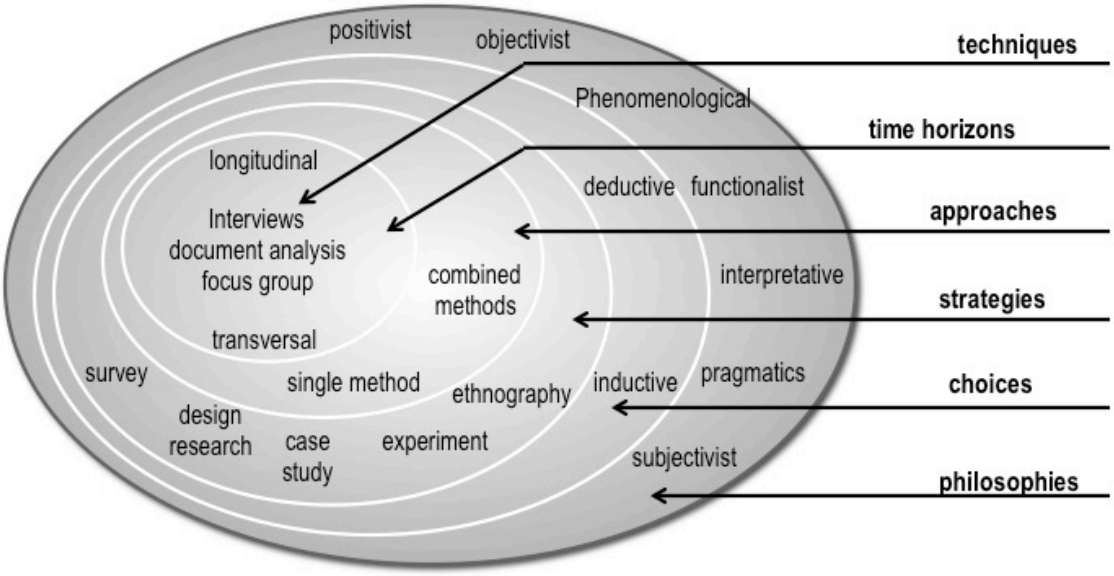


Figure 3 Research onion (adapted from (SAUNDERS; THORNHILL;LEWIS, 2006))

## 1.4 Research Process Outline

Following the constructive research strategy, we organize our work into four phases, as illustrated by Figure 4. Each phase is constituted by specific research processes (stages) and by resulting products. We also associate each phase to its respective constructive research stages (numbered bellow each phase).

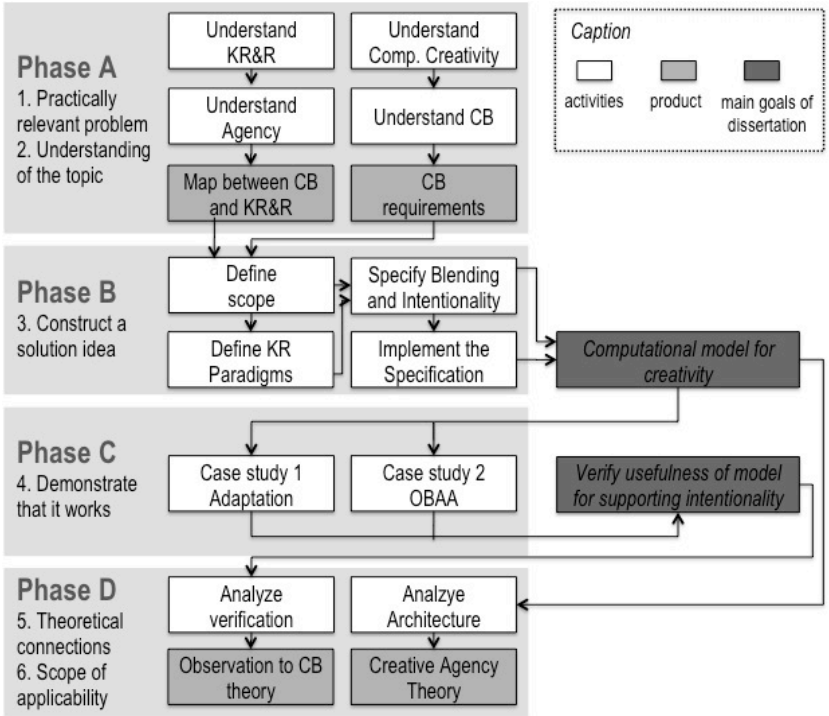


Figure 4 Research process outline following the constructive research.

Next, we describe each phase in terms of their stages and products, contextualizing its importance for the research as a whole. In addition, we briefly describe the results of each phase, describing the development of the work against the proposed stages. During

the following description we will label the stages with S.P*x*, where S regards to stage, P represents the phase and x a numerical ordering of P stages.

### 1.4.1 Phase A

The first phase regards the two initial constructive research stages (KASANEN; KARI; ARTO, 1993): identification of a problem with practical and scientifical relevance, and understanding of the investigating theme. These stages were developed mostly by bibliography research. Our main focuses of investigation through literature revision are computational creativity (S.A1), concept blending (S.A2), knowledge representation and reasoning (S.A4) and agency (S.A5). Based on the studies on CC and CB, we are able to specify a set of the requirements summarizing CB in terms of its constructs and processes (S.A3). Given an understanding of KR&R and agency we describe a mapping between the CB requirements previously established and possible ways to represent them computationally (S.A6). Such mapping was the main outcome of phase A, serving as a guide for the development of the next phase.

### 1.4.2 Phase B

This phase encompasses the third stage from constructive research: "Innovate, i.e. construct a solution idea" (KAZANEN; KARI; ARTO, 1993). In our research, this was the most important stage since it is here that our model is defined. The development began by limiting the scope of the model (S.B1). We specify some limitations due to the broad scope of CB applicability. As stated by Fauconnier and Turner (1998), CB theory aims to explain how we humans integrate known concepts to construct new ones. But, this integration can be applied to virtually anything that we are able to conceive.

As a result, a full computational account of CB would require multi-dimensional representations (e.g. symbolic conceptualization, associated emotional and sensorial perceptions, contextualization and episodic associations) and connections of a single concept. Even when considered alone, each one of those dimensions constitutes research themes on their own. Therefore we limit the design scope of our model in terms of which constructs and processes will be considered and also to which extent when relating to the original conceptualization.

Closely related to our scope limitations is the definition of the representation paradigms (S.B2) that will be used to specify the model. This stage is based on the mapping provided by S.A4, further specifying the role of the representation in the whole model.

Next, we specified our model – given our limitations and knowledge paradigm – to represent conceptual blending integrated to an intentionality structure (S.B3). In this level, the model define how blending is represented and how the blending process will handle BDI-based intentionality structures (also represented). Given our focus on adaptation, our specification provides additional structures (specific frames and template networks) and triggers to allow adaptation. The specification is written in traditional Structural Operational Semantics (SOS) (PLOTKIN, 1981, 2004).

Finally, we implemented the specification (S.B4) using the Jason (BORDINI; WOOLDRIDGE; HüBNER, 2007) framework and the OWL language as syntax for the blending engine. This stage was partially developed since we applied more time on the development of the specification and only descriptive representations (OWL) are considered in the implementation. Specific KR&R necessary for CB is integrated through Application Programming Interfaces (API) – when available – and, if not, we

adopt alternatives such as the Java Native Interface (JNI) or Java Simplified Object Notation (JSON).

In consequence of all phase B stages, the outcome is an agent architecture with an integrated concept blending mechanism. Regarding this outcome, we consider that the developed specification (S.B3) together with the models from phase C partially results in an agent architecture. The remaining part would be technical details from the implementation that could be applied to further specify the model (e.g. specific language details).

By specifying how intentional structures can be used during blending (S.B5), even in the specification level, we give an important step toward goals G1, G1.1 and G2. The foundations to achieve Goal G2.2 are also supplied by the model and partial architecture from phase B.

### 1.4.3 Phase C

Here, we cover the remaining aspects to achieve the proposed goals. This phase develops the fourth stage of constructive research, which is to demonstrate that the solution works. Although the partial implementation from S.B5 and the specification from S.B3 already demonstrate that at least parts of the solution work, we view that both implementation and model as structures that need to be filled with content in order to make sense. Therefore, our initial idea for phase C was to implement two case studies to consider the practical aspects of the architecture. In practice, we did not developed two case studies, rather we developed two studies with the goal to test the constructs specified on phase B. Still, we will present our original idea of the case studies and the respective limitation to test studies.

Our first case study aimed at studying how our architecture would behave during adaptation tasks (S.C1). Since the blending mechanism is implemented on the architectural level of Jason, we are able to test it with already developed agents, with minimum configuration effort. The integration of blending into an agent architecture were developed using the Jason API. Our stage S.C1 focused on the specification with operational semantics of a blend-based adaptation, its implementation, and on functionality tests with one example agent. Upon completion of this study, goal G2.1 is achieved, since we specify how blending can be applied to adaptation tasks.

Next, we applied the developed model to specify an agent that recommends educational resources based on the student's history and on the meta-data specification of the resources (S.C2). A complete study over recommendation systems require at least a statistical analysis of recommendations' success and a comparison to algorithms implementing the same kind of recommendation. Such study is not part of our research project. Here, we are interested in verifying the utilization of the creative mechanism as language primitives (implemented as internal actions in Jason by our architecture). In the context of the application, which is a research project to provide learning content on Digital TV (DTV), mobile and web platforms, creative reasoning is applied to surprise the user or to establish relations among content following a an association pattern different from the traditional approaches (based on deduction and generalization).

Both studies represent a proof of concept (S.C3) for the architecture and model, providing content so that the creative reasoning mechanism can be tested. Thus, the product of phase C concludes the developmental part of our research as goals G1, G1.1 and G2 are studied through the cases.

### 1.4.4 Phase D

On the last phase of the research project the link between the model and the theoretical reference is presented. In addition, the applicability scope of the solution is examined (KAZANEN; KARI; ARTO, 1993). Hence, the two first stages of phase D propose an analysis of our model (S.B6) and proofs (S.C3) given a background of computational creativity, agency and lastly, an integrative perspective of the fields. The first stage focuses on the model (S.D1) while the second studies aspects shown by the tests (S.D2). One of the products of this phase is the observations and discussion to CB theory triggered by the development of the model (S.D3). We believe that our model can contribute especially on regard to the representation and utilization of intentionality – considered as a vital relation inside blending theory (Section 2.3). The last product of our research positions our results as the initial steps towards a theory of creative agency (S.D4).

## 1.5 Research Chronogram

Next we present our research chronogram describing the development of this work from the beginning to its conclusion. Besides the stages defined in Section 1.4, the chronogram illustrated by Table 1 contains activities that were part of the PhD but do not provide a direct contribution to the research itself.

Table 1 Research Chronogram

| Stages | Year / Semester |
|---|---|
| ☑ Obtaining credits in disciplines<br>☑ P1. Paper publishing (AAMAS, PROMAS)<br>☑ S.A5 Understanding agency – BDI approaches | 2007 /1 |
| ☑ Obtaining credits in disciplines<br>☑ S.A5 Understanding agency – AI and Education<br>☑ P2. Paper publishing (ITS, ICALT, CAEPIA, PRIMA) | 2007 /2 |
| ☑ Qualification Exam – KR&R, specific theme: Ontologies<br>☑ S.A4 Understanding KR&R – Survey<br>☑ French Proficiency Test | 2008 /1 |
| ☑ S.A5 Understanding KR&R – Concept Phil., Symbol Grounding<br>☑ S.A5 Understanding Agency – AgentSpeak (L, DL), Jason<br>☑ OBAA project (proposal and devlopment) | 2008 /2 |
| ☑ P3. Paper publishing (WCCE – 2 papers)<br>☑ S.A2 Understanding Concept Blending<br>☑ S.A3 CB Requirements<br>☑ S.A6 Mapping of CB and KR&R approaches | 2009 /1 |
| ☑ S. A2 Understanding Concept Blending<br>☑ S.A3 CB Requirements<br>☑ S.A6 Mapping of CB and KR&R approaches<br>☑ P4. Paper publishing (EPIA)<br>☑ S.A1 Understanding Conceptual Creativity<br>☑ S.B1 Define Model Scope<br>☑ S.B2 Define KR&R paradigms<br>☑ S.B3 Specify the Model<br>☑ S.C2 Recommendation Case Study | 2009 /2 |

| | |
|---|---|
| ☑ P5. Paper publishing (WEBMEDIA, not accepted)<br>☑ S.B3 Specify the Model<br>☑ Thesis writing<br>☑ S.B4 Implement the Specification<br>☑ S.B5 CB Agent Architecture<br>☑ Thesis writing<br>✓ S.C1 Adaptation Case Study<br>✓ P5. Paper publishing (Int. J. K-Based Systems) | 2010 /1 |
| ✓ S.D1 Analysis of the Architecture<br>☑ S.D2 Analysis of the Conceptual Proofs<br>☑ S.D3 Observations to CB Theory<br>☑ S.D4 Theory of Creative Agency<br>✓ P6. Paper publishing (Int. J. AAMAS)<br>☑ P7. Paper publishing (RENOTE)<br>✓ P8. Paper publishing (ICCC 2011)<br>☑ PhD Thesis conclusion and defense | 2010 /2 |

**Caption:**

☑ Completed.
✓ Partially developed

# 2 RELATED WORK

In this section we describe the works that most affect our goals. As a result, this section does not illustrate a complete survey of the fields, instead, it presents sufficient works to contextualize our contributions. Each section begins with a brief contextualization on how the described area affects our work. Our presentation of related work also focuses more on creativity than agency. This decision was made because, for us, creativity and its computational counterpart were a less explored topic than agency. The remaining part of the section is organized as follows: section 2.1 presents a theoretical perspective on creativity, while section 2.2 presents a computational perspective, introducing computational creativity; next, section 2.3 describes the theory of concept blending; in section 2.4 we present Divago (a computational implementation of CB); section 2.5 describes algebraic semiotics (a partial formalization of CB); finally, section 2.6 presents works on agent adaptation.

## 2.1 Creativity

This section introduces the topic of creativity from a theoretical perspective. The concepts and views given here are fundamental for a discussion about the applicability of our model in general studies of creativity (research stages S.D3 and S.D4). Our ambitious view is that we may contribute on the research about how humans are able to create new ideas and also use this ability on our daily lives. Furthermore, based on the concepts presented here we also position our research in terms of general perspective on creative behavior (research goal G1).

As with any conceptualization representing something that humans do, but cannot explain how, creativity does not have its precise definition – no news for the AI researcher here. For the purposes of this work, we consider creativity as a process that generates new ideas or concepts. This line of thought follows the theoretical stand by Boden (2004), which goes further on the definition:

> Creativity is the ability to come up with ideas or artefacts that are new, surprising and valuable. (BODEN, 2004, p.1)

Boden begins her definition by relating creativity to the ability to generate ideas. Her intuition is to view creativity as a generic act of creation, so, considering ideas, it might be a new thought, concept, poem, music, mathematical theorem, cooking recipe, joke, and so on. She also mentions structures and artifacts which positions creativity in the realm of concrete things, like paintings, sculptures, origami and vacuum cleaners. In summary, Boden views creativity as the ability to generate ideas and artifacts, in the broadest context possible (BODEN, 2004).

To describe novelty, Boden (1998, 2004) introduces two kinds of creativity: Psychological and Historical – (P and H creativity, respectably). P-creativity

contextualizes novelty inside the individual or system itself (e.g. children often come up with ideas that are new to them, but that have been on textbooks for years). If someone comes up with an idea that no one else has had it before, then the idea is considered H-creative. Boden (2004) argues that H-creativity is a special case of P-creativity and, hence, the study of how creativity happens is in the realm of P-creativity. Dorin and Korb (2009) and Saunders and Gero (2002) present the application of creativity to artificial life. Both works define creativity in similar ways, based on Boden's H-Creativity approach.

Still looking at Boden's definition, she also characterizes a creative idea as being surprising – with regard to three interpretations. The first specifies surprise as its unfamiliarity or the improbability of its occurrence (e.g. winning the lottery). An alternative interpretation of a surprising idea is the one that unexpectedly fits into a paradigm (style of thinking) that you already have. Here, the surprise is to realize that this new idea is part of an already established paradigm – e.g. "how did I not think of that first?" In addition, an idea might surprise because of its apparent impossibility, it breaks paradigms and even evokes more ideas considered impossible and that, now, on the shed of this new idea, might become reality.

Finally, the last term in the definition of creativity refers to the value of the new idea. Defining value, or what is valuable – or interesting, useful, beautiful, … – is, to say the least, subjective and dependent on the situation where value is being assessed. As stated by Boden (2004) herself, her notion of new has two meanings and her notion of surprising, three; but the notion of valuable, no one is able to tell how many meanings it has, and might have. Although context-dependent and potentially controversial, having value inside the definition of creativity actually reflects our general perception of creativity. I might hear a Beatles' song and consider it highly creative but at same time, you might hear the same song and consider it a mere evolution of British rock and roll with influences from the hippie culture. While value remains a subjective perception, we will not be able to settle on a scientific theory for creativity. However it does not restrain research on the processes that generate creative ideas (BODEN, 1998, 2004).

Focusing on how creativity occurs, Boden (2004) distinguishes three main kinds of creativity: combinatorial, exploratory and transformational. Combinatorial creativity combines familiar ideas with unfamiliar connections (e.g. poetic imagery, collage in painting or textile art, and analogies). Exploratory and transformational creativities are properly explained under the perspective of conceptual spaces, which Boden consider to be structured styles of thought. These styles are part of a culture or a group and are not originated by one individual mind (e.g. ways of writing poetry, styles of painting or music and culinary schools).

Boden's view of conceptual spaces is not definitive, works influenced by neural and cognitive sciences (FAUCONNIER; TURNER, 2002, 2008) tend to consider spaces as packages of concepts dynamically constructed by our brains as we interact with the world (making coffe, writing a thesis, driving a car, trees, cars, among others). Still, Boden's exploratory and transformational creativities make sense in either view of conceptual spaces. Exploratory creativity generates new ideas by exploring hidden possibilities inside a previously established conceptual space, for example, the "thinking outside the box" idea that makes perfect sense – it fits in the space – but was not explicit in the current paradigm.

Moving to a more surprising kind of creativity, the transformational process result in impossible, unthinkable ideas, relative to an established conceptual space. It is that kind of idea that breaks, or at least changes current paradigms, welcoming new thoughts that were previously inconceivable.

To balance Boden's theoretical (philosophy) discussion on creativity, we present the view proposed by Duch and Pilichowski (2007), where the authors follow discoveries on neurosciences to contextualize creativity. Hence, according to Duch and Pilichowski, the process of creativity is defined as follows:

> Creativity involves neural processes that are realized in the space of neural activities reflecting relations in some domain (in the case of words, knowledge about morphological structures), with two essential components: 1) distributed chaotic (fluctuating) neural activity constrained by the strength of associations between subnetworks coding different words or concepts, responsible for imagination, and 2) filtering of interesting results, amplifying certain associations, discovering partial solutions that may be useful in view of the set goals. (DUCH; PILICHOWSKI, 2007, p.126)

In that definition, filtering is viewed as a process that relies on several features, like previous expectations, current associations and arousing emotions. Duch and Pilichowski discuss that creativity requires prior knowledge, imagination and filtering of the results. According to his view, imagination is constrained by the possible compositions of elementary operations (activations of neurons and neural-networks). Duch and Pilichowski propose that filtering imagination by domain, forming conceptual spaces (reflecting different neural configurations) is also an important aspect for creativity.

## 2.2 Computational Creativity

Shifting the focus to computer science, this section presents an overview on computational creativity. Far from a survey, our aim here is to describe the main approaches of the field, allowing us to characterize our contribution on computational approaches to creativity (research stages S.A1, S.D3, S.D4).

We present a few conceptualizations of the field, its main perspectives and a categorization scheme. All the concepts discussed so far regard (represent) Boden's theoretical creativity approach – although a bias towards computational models can be noted on the descriptions of creative processes. When considering computational creativity, Boden (2004) notes that computational approaches can help on investigating new hypotheses of how the mind works, but she does not commit to a definition for it. Wiggins (2006), proposes a set-theory-based formalization of Boden's conceptualizations, in an attempt to further specify her theory. Along with the formalization, Wiggins also constrains Boden's definition of creativity, seeing it through the perceptive perspective:

> The performance of tasks which, if performed by a human, would be deemed creative. (WIGGINS, 2006, p.451)

Wiggins assumes that creativity is too subjective to define, thus he proposes a definition grounded on the perception of creativity, in the sense that, although we are not able to precisely define it, we know it when we see it. Since Wiggin's purpose is to further specify Boden's view, after his definition of creativity, he conceptualizes computational creativity as follows:

> The study and support, through computational means and methods, of behavior exhibited by natural and artificial systems, which would be deemed creative if exhibited by humans. (WIGGINS, 2006, p.451)

Moreover, Wiggins also characterizes a creative system as a collection of processes, natural or automatic, which is capable of achieving or simulating behavior that would be considered creative in humans. A similar definition of computational creativity is presented by Franová and Kodratoff (2009) where it is considered to be the whole set of methods by which a computer may simulate creativity. We consider computational creativity as a field of study that has the potential to help us understand how creativity occurs in humans. Thus, we define computational creativity as the study of creativity through (based on) computational methods. The ways that the study can be conducted actually represent the main streams of work under computational creativity.

One stream researches ways to model the process of creativity generically, formalizing specific properties of creative reasoning and then contextualizing them into a broader framework of cognition (CARDOSO, A. et al.. 2001; PEREIRA; CARDOSO, 2006; PEREIRA, 2007; BLAIN, 2007; SAUNDERS, 2006; WIGGINS 2006; COLTON, 2008). A second line is represented by more specific works that attempt to mimic human creativity in specific domains, like music composition and improvisation (MÁNTARAS, 2006; PEARCE; MÜLLENSIEFEN; WIGGINS, 2008), joke (BINSTED, K. et al.. 2006), poetry (PILICHOWSKI; DUCH, 2008), story (ZHU; HARREL, 2008), painting generation (COLTON; MÁNTARAS; STOCK, 2008), and theater interpretation (MORAES, 2004). These works not necessarily commit to a formal creative model, rather, they use specific techniques (e.g. neural networks, Bayesian networks and genetic algorithms) leading to context-specific creative behavior.

Most of these works also consider, implicitly, the output of a creative system as an action contextualized by the environment and also by other actors (an audience, other musicians, other actors, etc.). According to our perspective, this line differs from the one followed in this research in terms of the abstraction of action. In our context, we observe action and practical knowledge in a higher abstraction level, considering a model of practical reasoning itself. The specific creative systems in general have outputs that can be perceived as actions, but are not modeled into an explicit practical reasoning framework. Thus, we do not position our work on this line of computational creativity. Finally, a third line groups systems and techniques to support creative work (SAUNDERS, 2009; HÉLIE; SUN, 2008). Regarding this division of computational creativity, we position our work on the first stream (computational models of creativity).

Recalling Boden's theory, she proposes a distinction between Psychological and Historical creativities. This division actually represents the main stands (positions) on creativity, P represents a focus on the phenomenon of creation while H represents a focus on the result of the phenomena, a view that positions the creation inside a context (societal, cultural, historical, …).

Our view focuses on the P side, since we want to empower the agents with a kind of reasoning that allow them to further use what they know or what they can do. Moving to a multi-agent perspective, H-creativity also present opportunities for research, given that a societal and normative context is present and influentiates the agent behavior. A general account of the H view is presented by Csikszentmihalyi (1988). Figure 5

illustrates that view, examining creativity from systemic perspective (boxes represent systems and ellipses, actors).

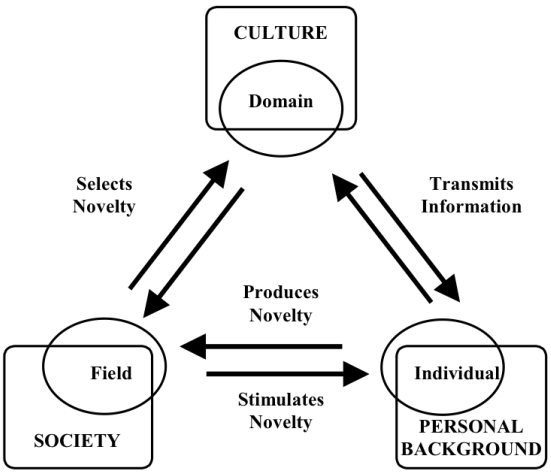


Figure 5 Systemic view of creativity (CSIKSZENTMIHALYI, 1988)

Csikszentmihalyi considers that the individual, its societal (interactive) field and its cultural (symbolic) domain are the basis (boxes plus ellipses) of a creative system. In this system, creativity is fostered by the interactions (arrows) among the individual, the domain and the field. Saunders and Gero (2002) note that in such perspectives, creativity is as much the result of the appreciation of a work as it is the product of the creator.

Following the line of H-creativity, Ritchie (2007) proposes a framework for the assessment of computational creativity by specifying a set of criteria to be applied on a generated artefact. A key assumption on Ritchie's framework is that all the criteria must be based solely on empirically observable factors, ignoring the process behind the production. Focusing on what is observed, Ritchie aims at mimicking how humans judge art works, usually without awareness of the artist's mental or emotional processes that lead to the production. Hence, the proposed framework puts a computer program on the same level of a human creator, since the assessment relies on a human to judge the output of a creative program. Ritchie establishes the set of criteria around the properties of quality and novelty. These properties are specified in terms of class membership and an inspiring set, which is applied during the assessment for comparison purposes. A total of eighteen criteria are defined (e.g. average typicality, average quality, good result, good typical and atypical results) composing an evaluation table.

On the opposite side, Colton (2008) defines guidelines for the assessment of computational creativity following a P-creativity perspective. Colton assumes that the process of generating an artefact does matter, and that humans consider it when judging a creative product. Basing his argumentation on artwork appreciation, especially on the impact that knowing how an art piece was produced has on the enjoyment of a work, Colton defines his framework around three properties: skill, appreciation and imagination. These properties constitute what Colton calls the creative tripod, representing the behaviors that a creative system must exhibit in order to be considered creative. Colton proposes the use of the tripod when developing the creative software – as a guide to the developer – and also as an intuitive way to present computer-generated art to non-technical consumers.

## 2.3 Concept Blending

Since the beginning of this research, our main motivation was how could agents make better use of their knowledge? How can agents go beyond traditional logical reasoning towards other kinds of human-like reasoning? These questions lead us to study how our brains represent and reason over symbols and concepts. Studying philosophical and cognitive studies on how our brain uses and represent concepts (MURPHY, 2002), we came along a hybrid theory that originated from cognitive linguistics that follows developments from neurosciences and philosophy of the mind called concept blending.

This section describes the CB theory and is the basis for our model of creativity (goal G1, G1.1; stages S.A2, S.A3 and S.A5). Although blending is not seen as a theory for creativity, it can be used to explain the production of creative artifacts, since essentially, it explains how our mind produces new conceptualizations.

Concept blending, also known as Conceptual Integration (CI), is considered to be a general cognitive operation related to analogy, categorization, framing and mental modeling (FAUCONNIER; TURNER, 1998). The theory is based on empirical observation of meaning construction in different domains, including mathematics, social sciences, human-computer interaction, literature, music and mainly linguistics. It originated in the field of cognitive linguistics with the goal of explaining how we understand creative phenomena such as metaphors and counterfactuals.

According to Fauconnier and Turner (FAUCONNIER; TURNER, 2002; FAUCONNIER, 2008) conceptual integration follows the developments from embodied cognition and describes a mental capacity that leads to new meaning, global insight, and conceptual compressions useful for memory and manipulation of otherwise diffuse ranges of meaning. Although creative and insightful constructions can be explained in terms of CB products, CB itself does not hold to fully explain creativity since some important points are not defined, such as the selection of inputs for the process.

Despite its importance, especially to provide computational models for the theory, detailed aspects of the operation are not defined and perhaps are not part of the original research agenda of the authors. In (FAUCONNIER; TURNER, 2002) conceptual integration is characterized as a non-algorithmic and non-deterministic operation. These characteristics lead to a mind model where operations are executed in parallel and the innumerous possible lines of thought are seen as a source of variety and creativity. Reasoning operations such as analogy and categorization are also examples of aspects of the theory that are important but not defined.

In a nutshell, conceptual integration takes as input two mental spaces, constructs a partial match between them and then projects selectively to a new space, the blend, which leads to new emergent structure. This structure arises through composition, completion and elaboration. Integration takes place in networks of mental spaces that represent our neurological structure. As blends are created they expand the network and might modify previously established mental spaces. Figure 6 illustrates this structure: the big circles represent the spaces, small black circles inside the spaces are the concepts, dotted lines represent projections while the regular lines represent the cross-space matching between the inputs. Both kinds of lines are seen as connections among spaces.

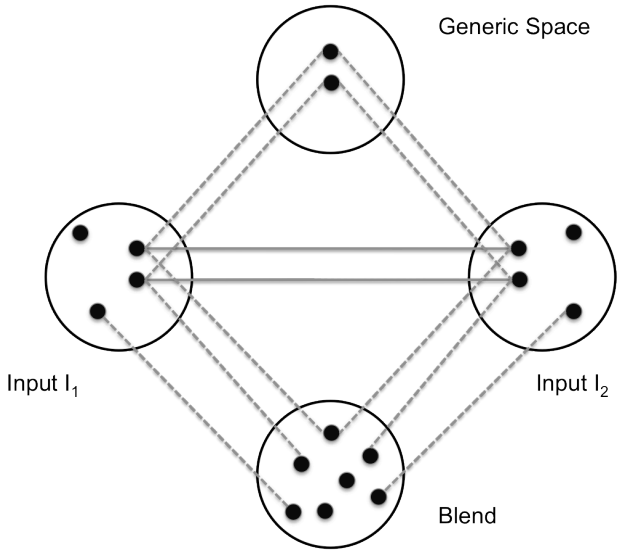


Figure 6 Blending basic diagram (FAUCONNIER; TURNER, 2002)

Input spaces for the integration process are set up according to the perceived stimulus and its context (specific, generic, counterfactual, among others). The perception evokes concepts and related information that will shape the mental spaces at hand. After a network of spaces is established, a partial cross-space mapping connects counterparts in the input spaces. Establishing counterpart connections can be performed with several kinds of relations: analogical, metaphoric, role, frame-based and vital relations mappings.

Along with connecting counterparts, a generic space is built-up from the inputs. This space captures the common structure between the inputs mapping its paired counterparts. The resulting generic space and the cross-space mapping are used during the selective projection operation. Selective projection takes elements from the inputs, mappings and generalizations and projects some of them to a new space, the blend.

Although important, selective projection is not defined in the theory. Nonetheless, the blending theory describes restrictions and principles that can be applied to selective projection, but are not explicitly linked to this operation (mainly composition and completion). We consider that this gap exists because the theory is still in its early days and evidence from neurosciences is necessary before formalizing that kind of process. A formalization of selective projection is one aspect for a complete account of the creative process.

In our interpretation of the blending process, selective projection is performed through composition and completion of the inputs and related spaces. Elements from the inputs can be composed providing relations that do not exist in either input. Composition considers mostly counterpart relations, which can be included separately in the blend (each part is projected onto the same blend, but as separate elements) or as the same element in the blend (a fusion of the elements).

Completion brings background knowledge to the integration process. It uses previously established frames to provide pattern completion. The definition of frame used in blending is broader than the classical one known to AI (MINSKY, 1974). In this case, frames specify general organizational aspects, for any kind of purpose and with any granularity. It is possible to have a “soccer game” frame as well as a “meaning of a PhD” frame. How frames are established is an issue not discussed in the theory.

When the blend space is ready, a process called elaboration can simulate its execution according to the principles defined for the blend (usually obtained from completion). Elaboration attempts to capture our capacity to imagine the impact of an element inside a hypothetical scenario (how it would turn out if it existed in the concrete reality). From the principles (time scale, physics dynamics, space, and so on) innumerous possibilities for elaboration arise. This operation can also execute repeatedly and as long as desired.

A question that comes to mind is when will the process stop? According to the theory, blending is an unconscious process that runs practically all the time. The general goal is to support human-scale interaction and understanding of the world. Human-scale is a property difficult to define and in our opinion can be seen like intelligence is seen by the strong AI community. Bringing human-scale to blending gives us purpose under the form of vital relations, which impose restrictions to the process.

Fauconnier and Turner (2002), characterize the operations that we have presented so far as the constitutive principles of the blending process. Since they contextualize the blending theory under a greater scheme of cognition, the constitutive principles are seen as a first level of constraints to the process. We do not share this view, we consider that the constitutive principles are the basic operations of the process, allowing a network of concepts to be modified, expanded and understood accordingly. Still, we will maintain the original terminology through the text. To summarize, the constitutive principles of blending are: input definition, cross-space connections, generic space definition, selective projection, composition, completion and elaboration.

### 2.3.1 Governing Principles

Along with the constitutive principles, governing principles are also defined as a stronger source of constraints to the process. Constitutive principles are related to the structure and dynamics of the blends while governing principles characterize strategies for optimizing the emerging blend. Governing principles have a higher abstraction level and guide the blending process towards the generic goal of achieving human-scale. Thus, the intuition is to constrain and prioritize operations that will bring about understanding, sensing and acting to blends in a human-like fashion.

Our repetition of what is human-scale is on purpose, since we would like to clarify that it is closer to a general research goal for concept blending – and cognitive sciences – than a conceptualization of a clear goal to be achieved by blending. Nonetheless, achieving human-scale is the purpose behind governing principles, to restrain the operation towards human-scale acting. To conclude, our presentation of these principles assumes that achieving human-scale is implicitly defined in every governing principle, instead of a clearly qualified goal to be achieved. The exceptions to our simplifying assumption are the vital relations.

Although Fauconnier and Turner do not specify a formal connection between vital relations and human-scale, we consider that these kinds of relations characterize some aspects of what they mean by human-scale. Fauconnier and Turner (2002) characterize vital relations as “Vital relations are what we live by, but they are much less static and unitary than we imagine” (FAUCONNIER; TURNER, 2002). Specifically under blending, vital relations are conceptual relations that tend to show up many times, representing relations like cause-effect or change. In our interpretation of the blending theory, vital relations can be applied as filters for the links established during cross and inner space connecting, generic space definition and selective projection.

Fauconnier and Turner (2002) present the following list of vital relations: change, identity, time, space, cause-effect, part-whole, representation, role, analogy, disanalogy, property, similarity, category, intentionality and uniqueness. Having a general idea of vital relations and their role in blending is fundamental to understand the governing principles. Most of them establish operations and unique utilizations of vital relations. The main categories of governing principles are compression, topology, pattern completion, integration, maximization and intensification of vital relations, web, unpacking and relevance.

Although compression can be seen as a general method, in the sense that anything can be compressed into something else (e.g. a carrot can be compressed to a vegetable and a collection of states can be compressed into a country), in blending compression is mostly applied to vital relations. Therefore, the governing principles of compression present some guidelines on how vital relations can be compressed into others, guiding compression towards human scale.

The simplest case of compressing vital relations is when only one relation is compressed into itself. Fauconnier (2008) describes that it can be achieved by scaling and by syncopation. When a vital relation has a scale of some sort, like time that may be represented by a time interval, the scale – and everything it represents – can be compressed into a single point. For instance, a time scale representing how long a student took to complete his graduation course can be compressed into a single moment, his graduation ceremony. A chain of causes and effects can also be compressed to few or only one cause and one effect.

In addition, the range of effects, its kinds, its causalities and the respective kinds may be similarly compressed (e.g. a diffuse or fuzzy causation can be compressed into a precise one). Likewise, multiple roles can be compressed into a single composite role (e.g. mother, father and son become family). Also, patterns with little or fuzzy intentionality and long arrays of intentions can be compressed similarly to the cause-effect compression. Similarly, the vital relation of change also falls into this kind of compression, several changes into an object or concept are intuitively compressed into the final object.

Another way to compress a single relation into itself is through syncopation. Syncopation refers to the process of dropping scalar events, like a moment in a time scale, but keeping a few key events. Like most of the concepts from the blending theory, details of what characterizes a "key event" are left for the reader to consider. An example of syncopation is keeping only being born, first kiss, getting married, having children, having great-children, passing away from a time scale representing an individual's lifetime.

Increasing complexity, there are patterns for compressing one or more vital relations into another. Fauconnier and Turner (2002) proposes hierarchies of compression in order to compress multiple relations into a single one. For instance, consider the vital relation of representation connecting a representation to the thing itself. Inside a blend, representation can be compressed to uniqueness. A concrete example is our direct understanding that a photo represents a person. Contextualizing this example inside blending, between the photo and the person, we have the relations of representation and part-whole yielding a uniqueness relation in the blend. As presented by Fauconnier and Turner (2002), this is what happens – in blending terms – when a police officer points at and ID photo and asks: "Do you know this man?"

The relation among representation, part-whole and uniqueness is an example of the possible interplay among vital relations. Fauconnier and Turner (2002) establishes two hierarchies describing connections among vital relations. Far from being an exhaustive and final list of vital relations hierarchies, they represent what the authors have already researched. Figure 7 illustrates a hierarchical view of the relations among analogy/disanalogy and other relations, while Figure 8 illustrates the hierarchy for the cause-effect vital relation.

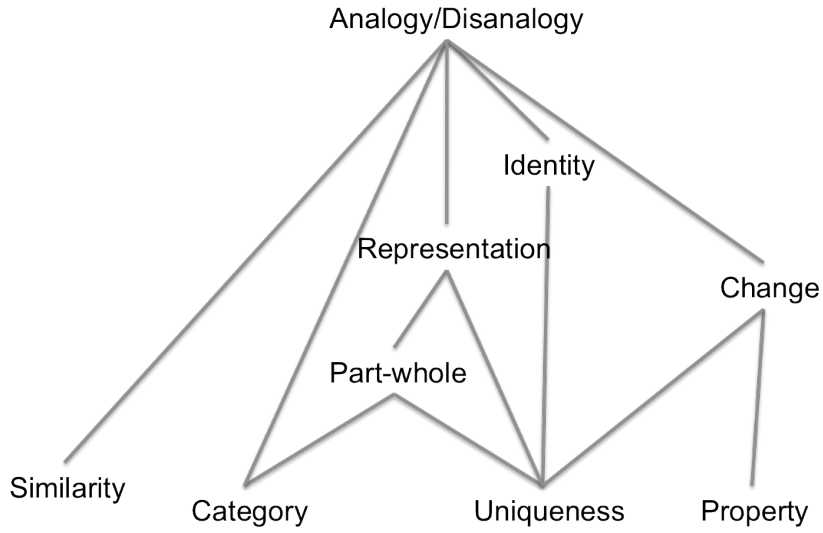


Figure 7 Compression Hierarchy for Analogy/Disanalogy (FAUCONNIER; TURNER, 2002)

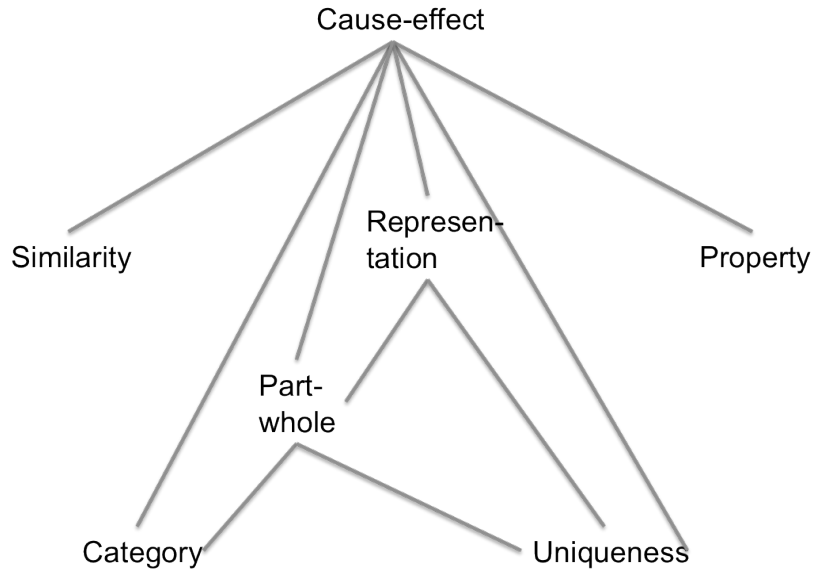


Figure 8 Compression Hierarchy for Cause/Effect (FAUCONNIER; TURNER, 2002)

Another compression principle is achieving inner-space scalability. To understand scalability it is necessary to have in mind the general goal of achieving human scale. In this context, inner-space scalability pushes the network construction towards having all the necessary connections of vital relations available in a timely fashion. Going to the network perspective, this means that even originally outer-space relations, such as representation, analogy, disanalogy and identity must be scalable to a single blend, providing readily available meaning and understanding to different situations –

achieving human-scale. Fauconnier and Turner (2002) exemplify this kind of situation illustrating that the outer-space relation of representation between a person and his label, for example, his name, is available in a single blend, where the name becomes a part and a property of the person.

Next, there is the principle of creating a new relation through compression. This principle relates to the idea that inherent to a new blend there is a possibility to create new connections – thus new meaning – that were not part of the inputs. What this principle states is that creating new relations inside blends constitute an important part of the blending theory, specially regarding emergent meaning.

Highlights compression is another one of the compression principles and regards the property of highlighting in terms of vital relations a whole network of concepts. Once more the importance of compression in connecting different networks and providing all the necessary information at once is prioritized. Finally, borrowed compression is a principle that applies an opposing force to creating new compressions. This principle guides the network construction towards borrowing compressions from networks that already have a tightly integrated scenario projecting a coherent compression to a blend.

The remaining governing principles provide optimization pressures beyond compression. The topology principle states the following: “set up the blend and the inputs so that useful topology in the inputs and their outer-space relations is reflected by inner-space relations in the blend”. Considering the topology principle, Fauconnier and Turner (2002) describe five possibilities to align current topologies from the inputs to the blend while optimizing compression.

First, simply not providing a counterpart of the relation on the blend. Although some information might be lost, the remaining relations might be emphasized and bring a better understanding of its importance. Second, it is possible to project the relation while scaling it in the blend. Third, syncopation may be applied during the projection. Fourth, the relation may be compressed into another relation. Fifth, a mutual inner-relation from the input spaces may be projected to the blend taking the relation from one space, but the compression from the other.

In opposition to topology, is the web principle – although in few situations the former aligns with the latter. While topology pushes blend construction towards keeping the overall topologies from the inputs, web pushes the maintenance of connections among spaces, sometimes limiting topological connections. Correspondingly, the web principle is defined as: “manipulating the blend as a unit must maintain the web of appropriate connections to the input spaces easily and without additional surveillance or computation”. In result of this principle, during blend development we should not disconnect valuable web connections to and from the inputs.

Another governing principle is pattern completion, which states that existing integrated patterns should be used as inputs to complete elements in the blend. Additionally, use an already developed frame whose relations reflect compressed versions of the outer-space relations between the inputs. Related to pattern completion is the integration principle that simply states: “achieve an integrated blend”. Integration in the blended space allows it to be manipulated as a single unit, allowing the blend to be directly utilized without constantly referencing other spaces. Consequently, integration is seen as sub-goal of creating human-scale blends.

Allied to this view are the principles for promoting vital relations through their intensification and maximization. Maximizing vital relations directs projections and creation of new relations towards an integration network with many supplementary vital relations in the blend and in the outer-space connections. Hence, the principle of maximizing vital relations is to maximize them in the whole network and specifically inside and outside the blended spaces (inner and outer space vital-relations). Although maximization can be seen as intensification (and vice-versa), the last reflects the idea to intensify the vital relations already available instead of any one possible.

Since blending has the compression principle, it is expected to have another for decompression. Such principle is called unpacking and regards the possibility that the blend all by itself should prompt for the reconstruction of the entire network. Following this principle, the blend works like a mnemonic or triggering device, presenting compressions that allow us to unpack them into full networks.

Finally, the relevance principle asserts that an element in the blend should have relevance – in a broad sense – including relevance for linking to other spaces and for running the blend. Relevance also pressures networks to have relations in the blend that are compressions of important outer-space relations between the inputs. Fauconnier and Turner (2002) relate relevance to unpacking: "network relevance can be satisfied for an element in the blend if it can be successfully taken as a prompt for unpacking".

In the earlier works on blending theory, the governing principles were called optimization principles (FAUCONNIER; TURNER, 1998). But, since they serve more as guidelines than optimization per se, the name governing principles fits better. In addition, these principles should be seen as competing pressures guiding the blends towards human-scale.

### 2.3.2 Network Typology

By means of constitutive and governing principles the general operation of blending gets described. But still, even considering that all the optimization pressures are applied, and that contextualizing (valuing appropriateness and important relations) is easily available, blending is capable of generating many possibilities of networks. It is argued that this capacity to generate novel conceptualizations – despite its complexity – is where lies the greatest contribution of the blending theory. In the middle of a world of possibilities, Fauconnier and Turner (2002) identified four types of network that have specific interpretations for human cognitive functions.

The first identified type is the simplex network, which is based on frames and roles. One of the input spaces is a frame and the other has possible values to be mapped through roles. In this case, blending is straightforward integrating the frame and the values in the simplest way. Simplex networks do not have competing frames or incompatible counterpart elements, neither organizing frames for the inputs.

Still considering the importance of frames, Fauconnier and Turner (2002) define mirror networks, where all spaces (inputs, generic and blend) share the same organizing frame. An organizing frame specifies the nature of the activity, events, participants, scales, relevance and any other important aspect for the understanding of the scenario. In addition, the organizing frame provides a topology for the space, organizing relations among elements of the space. Since the inputs and generic space share the same organizing frame, cross-space mapping becomes straightforward and clashes may occur only on more specific levels. Considering compressions, mirror networks perform them

over time, space, identity, role, cause-effect, change, intentionality and representation, both for inner and outer space connections.

Now, dealing with different organizing frames are the single-scope networks. In this kind of network, only one of the organizing frames is projected to the resulting blend. The input that supplies the organizing frame is called source and the other, which focuses on the understanding, is called the target. Having these source and target blending can be used to generate and explain "source-target" metaphors. Figure 9 illustrates a single-scope network with the "boxing CEOs" example (FAUCONNIER; TURNER, 2002).

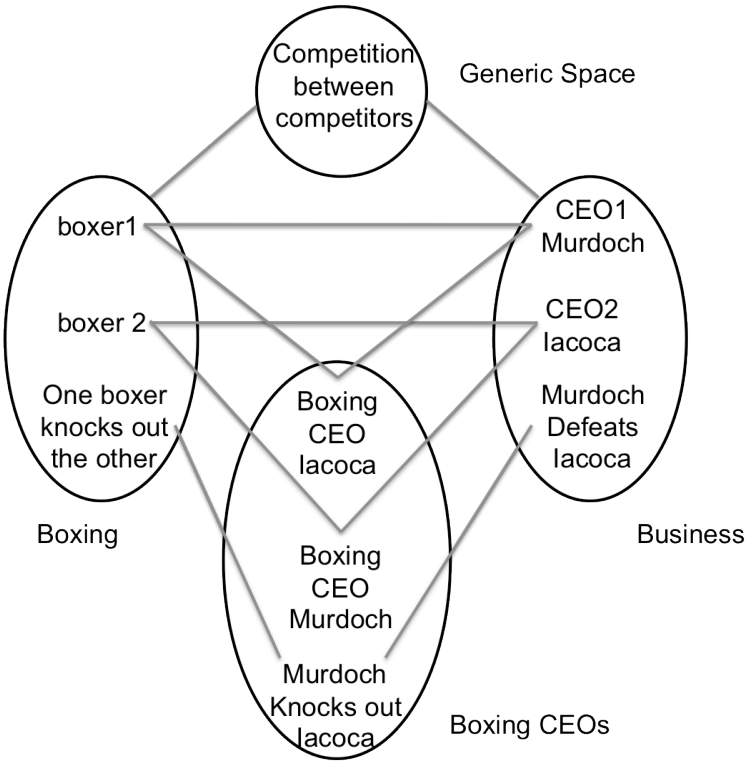


Figure 9 Single-scope network (FAUCONNIER; TURNER, 2002)

Moving to a more cognitive explanation, Fauconnier and Turner (2002) argue that single-scope networks give the feeling that "one thing" is providing insight into "another thing", independently of the distance between the things (spaces). Inferences, compressions and emotions from the source, which already is well known to us, will be applied to the target, thus applying the same general feeling of knowing to the target.

Besides different organizing frames for the inputs, blends generated by double-scope networks have an organizing frame with parts from each of the sources and with an emergent structure of its own. Both organizing frames contribute to the result and the differences between the inputs provide clashes and contradictions offering challenges to the imagination, resulting in potentially creative blends (FAUCONNIER; TURNER, 2002).

An example provided by Fauconnier and Turner (2002) is the computer desktop interface – actually, Imaz and Benyon (2007) provide a rich study on human-computer interfaces and concept blending. In the desktop example, the inputs have different organizing frames: the first is the frame of office work, with files, folders and trashcans; the second, computer commands to create, delete and organize files. The resulting blend

allows finding files, moving things to the trash, printing, and so on. Thus, the blend's frame has structure from both input's frames with a meaning of its own.

Fauconnier and Turner (2002) uses double-scope blending to explain several parts and results of human cognition, like culture, form and meaning, language and grammatical constructions. An unpublished paper by Brandt (BRANDT, 2002), who wrote a PhD thesis on literary analysis using concept blending (BRANDT, 2000) describes a more specified typology of integration networks considering their purpose and cognitive aspects. Figure 10 illustrates Brant's typology.

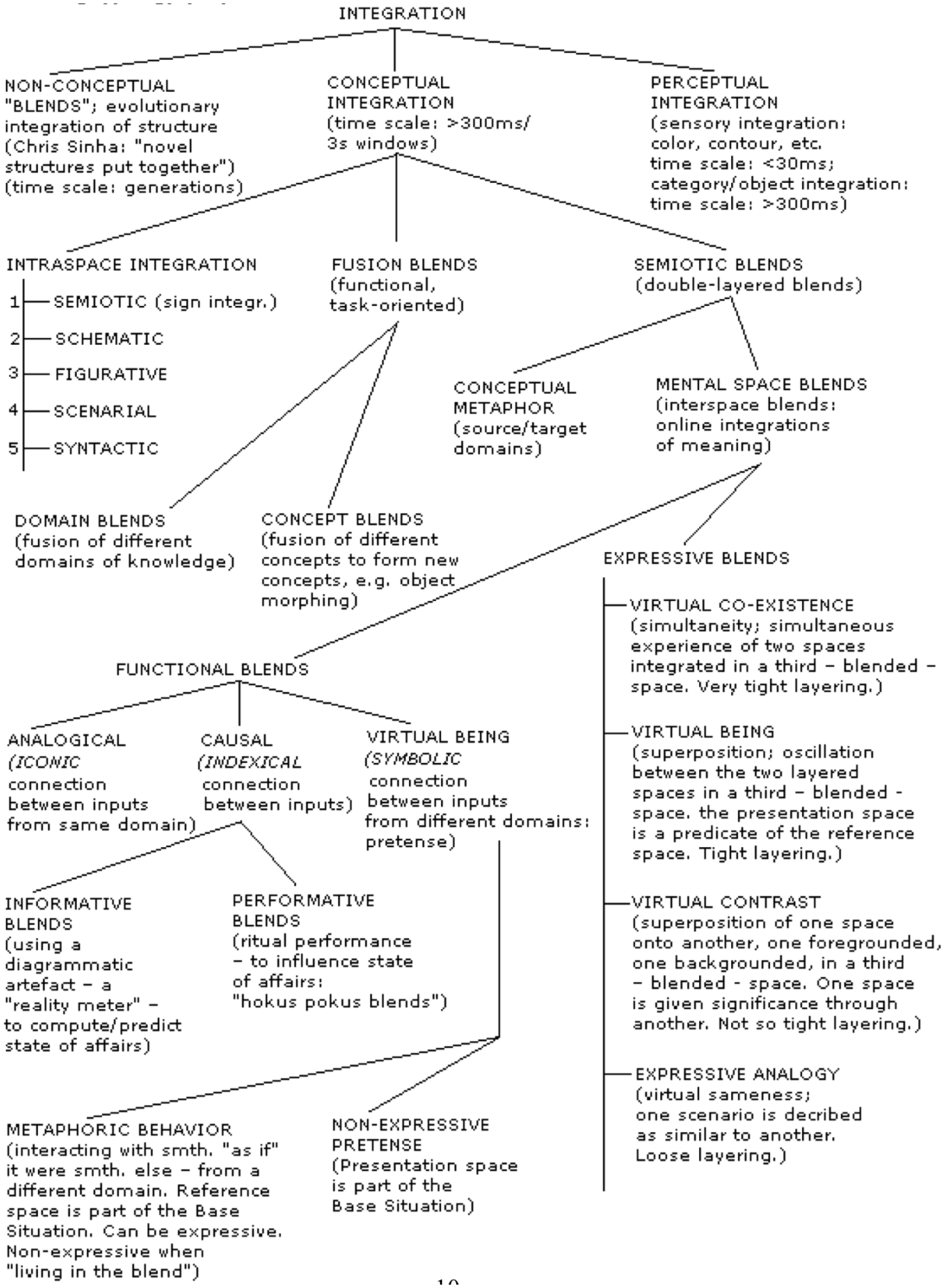


Figure 10 Blending Typology (BRANDT, 2002)

Brandt's typology begins by distinguishing conceptual integration from other kinds of integration, like nature's evolutionary structures and low-level sensorial-only integration. Since the focus of her work is on cognitive semiotics, she goes further on conceptual integration leaving other kinds of integration for future work. Conceptual integration is divided on three groups according to how spaces are considered during integration. Brandt proposes clear distinctions among integrations that may occur inside

one space (intraspace), integrations between two spaces (interspace) and integrations among domains of spaces (fusion).

Intraspace integration has five specific types: semiotic, schematic, figurative, scenarial, and syntactic. Intraspace semiotic integration regards the integration of signs, like an angry facial expression, gestures and greetings. Schematic integrations regard only structural integrations of the space, such as table leg, head of state and more is up, which integrates a quantity schema with a directional one. On the other side, figurative integrations consider aspects like shape and motion leading to purely figurative interpretations (e.g. an hour glass waist and head of lettuce). Scenarial integration considers compositions of space elements to work as single scenario, like the availability of language and utterances in a communication scenario. Ultimately, syntactic integration connects grammatical structures to their meaning in a single space.

Fusion integrations can have more than two spaces as inputs, but here, the spaces are considered to be fusions of categories or entire realms of knowledge. Brandt considers these spaces as domains and not as mental spaces. She characterizes fusion blending as task-oriented and functional, being more applied in problem-solving activities. Dividing fusion blends by their quantitative aspects, Brandt defines domain and concept blends.

Domain blends bring together two or more domains of knowledge blending structure from each input. This kind of blending goes beyond the local task of the individual, considering a broader temporal scope, such as the history of ideas from a whole field. Examples of domain blends are the fields of AI, neuroscience, psychology and literary studies. Reducing the scope of fusion blends, Brandt defines the concept blends category, essentially applying the same process of domain blending to a smaller scale. Recalling Fauconnier's and Turner's terminology, fusion blends would be represented by single and double scope networks.

Ultimately, conceptual integration is sub-divided in order to categorize interspace blends, which are also called semiotic blends. These blends have only two inputs that are not fused (a single-scope network). Thus, semiotic blends organize the inputs as vertical layers; the top one gives meaning to the bottom one, as conceptual metaphor. In this case, the source input works as a predicate of the target (e.g. love is a journey). Mental space blends generalizes the idea considering double-layer integration of any kind – not only source/target. That category is divided into expressive and functional blends.

Expressive blends are related to communication and expressive acts. The first sub-category is virtual co-existence where time is compressed to a single moment, allowing a simultaneous experience of two spaces in the resulting blend (e.g. imagine yourself discussing with Alan Turing the future of computer sciences). Instead of simultaneity, the next sub-category, called virtual being, considers the blend as a superposition between the inputs – in this context regarded as reference and the presentation. During elaboration, the blend oscillates the inputs, giving more attention to the reference at some times and to presentation on the others. For instance, counterfactual, pretense and generalizations – using Garfield to represent the whole category of cats).

Remaining on superposition, virtual contrast blends puts one space in the foreground (presentation) and the other in the background (reference). The difference to virtual being blends is that here the distance between the layers is bigger and, hence, the blend does not oscillate much during elaboration (e.g. negation and irony). Finally, expressive analogy considers analogies produced to explain relations to the individual or someone

else. Such communicative analogies occur, for example, when comparing one situation to another in order to facilitate the listener comprehension.

The second sub-category of mental space blends are functional blends. In opposition to expressive blends, functional blends describe functionalities achieved through blending. On the typology, Brandt (2002) divides functional blends by the way that the functionality is developed. For example, she proposes the functional analogical blends category, where blends are characterized by an iconic connection between the inputs, providing a functionality (e.g. solving something) using analogies. Brandt exemplifies analogical blends referring to a group of people trying to make a coffee machine to work. One person might suggest turning it off and then on again, like on a computer. Another suggests jiggling the handle, like on a water closet. In this example problem and solution are on the same domain, the machinery one, which is another characteristic of analogical blends.

Another way to provide functionality is applying causal links to the inputs. Causal informative blends adopt some kind of measure system to compute a specific state of affairs, which will be represented by the blend. An alternative causal blend is the performative kind, where cause-effect relations serve the purpose to influence a state of affairs, like in cultural rites.

Finally, symbolic connections between inputs from different domains are the last way to develop a functional blend (BRANDT, 2002). Such symbolic connections lead to metaphoric behavior (as if scenarios) and to non-expressive pretense. In metaphoric behavior, the individual (also called cognizer) interacts with something as if it were something else from a different domain (e.g. someone talking to his computer, or threatening it because the printer does not work). The non-expressive variation of that behavior happens when the individual is not aware of his own pretense, it is like he is living in the blend – this point is also discussed by Fauconnier and Turner (2002).

## 2.4 Divago

Under computational creativity, Divago (from the portuguese expression “eu divago”) is the work that most relates to ours. Following the early works of Fauconnier and Turner (1998, 2000), Pereira (2007) developed one of the most complete computational creativity model based on concept blending, the Divago system. Thus, the relation to our model is straightforward, providing a comparison basis and also insights to the design of our model (goal G1 and stages S.A.2)

Pereira (2007) proposes a creative general problem solver – an analogy to the classical problem solver by Ernst and Newell – resulting from the establishment of a set of principles for creative systems. Furthermore, the Divago system, which implements most of the creative problem solver, is presented and analyzed using Ritchie’s framework (RITCHIE, 2007).

According to a top-down approach, we first present Pereira’s principles and then a summary of Divago. After a discussion of creativity and creative systems, Pereira describes a set of guiding principles that represent what he considers as fundamental for computational creativity. First, he describes the knowledge principle, which is the basis for the creative act. Accordingly, Pereira argues that both quality and quantity of knowledge must be available and treated with equal importance. In respect to quantity, heterogeneity of knowledge must also be present, representing not only the problem domain, but also different perspectives and different domains. Related to knowledge is

re-representation, stating that a body of knowledge should be understood according to different viewpoints.

Associated to knowledge reasoning, the bisociation principle represents the ability to find unprecedented associations, usually through cross-domain exploration of structures and concepts apparently distant and unrelated. Still on reasoning, being able to reason about reasoning is also considered as a principle for creative systems – meta-reasoning. Pereira also brings the notion of evaluation, as presented by Csikszentmihalyi (1988) and Boden (1998), in terms of self and society as a principle. Another one is interaction with the environment, which contextualizes creativity inside historical, cultural and societal aspects.

Returning to internal aspects of creativity, the purpose principle asserts that there is always a purpose in any creation, even in the cases where it is very subtle. Such view proposes a goal-oriented perspective for creative systems. In addition, the divergence/convergence principle assumes the existence and importance of different modes of thinking to creativity. Divergent thinking is related to free associations, allows inconsistencies and relaxation of constraints, while convergent thinking is rational and methodic. The last principle presented by Pereira (2007) is ordinary processes. This principle reflects Pereira's theoretical position in terms of how creativity occurs, in cognitive terms. His stand is that, not necessarily, the process behind creativity is cognitively different from the process of rational reasoning – they might share the same grounding of any other cognitive phenomenon.

All those principles are put together in the creative general problem solver that is illustrated in Figure 11. The boxes represent the principles and the arrows interactions and dependencies among them.

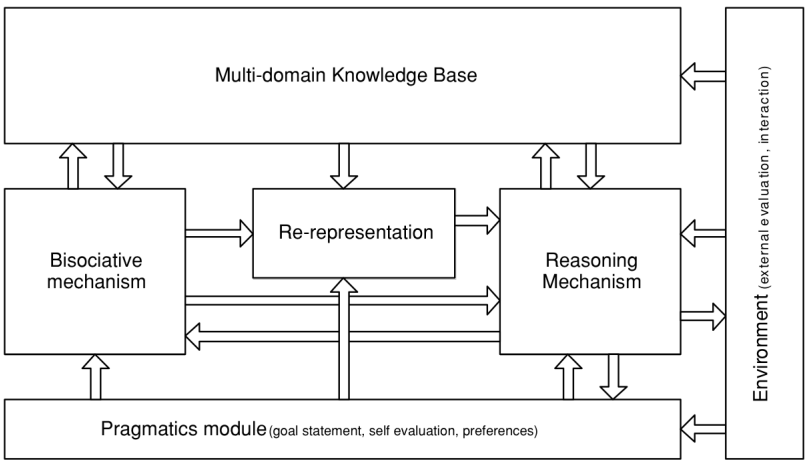


Figure 11 The creative general problem solver (PEREIRA, 2007)

Pereira discusses the implementability of his creative problem solver concluding that, to some extent, any of the components can be computationally implemented, perhaps even reduced to a search problem. Although, meta-level reasoning is considered to be one of the most challenging components, since it requires assessment and self-organization. In terms of applicability, it is argued that the model can be relevant where the generation of concepts is the goal, such as in design, architecture and arts. Another point raised – that actually reflects our position – is that it can help domain-specific implementations when no solution is found or to find novel ways to improve the system.

The creative general problem solver is implemented using logic programming in the Divago system. An overview of the system is illustrated by Figure 12, where it is notable the similarity with the generic model.

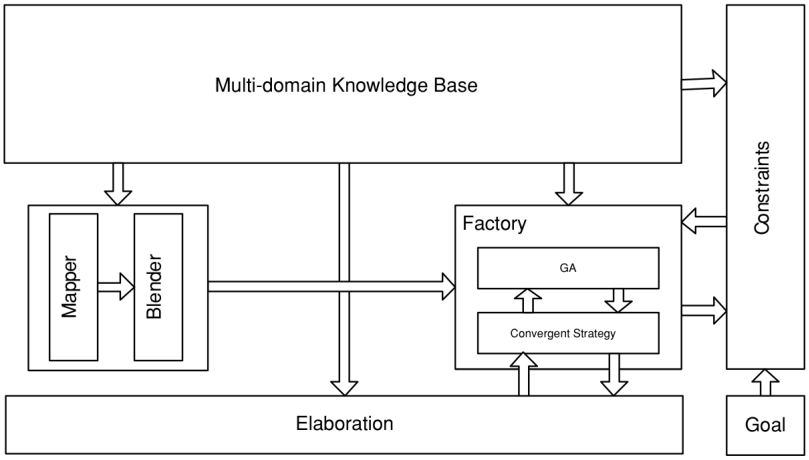


Figure 12 Divago's architecture (PEREIRA, 2007)

As described by Pereira, the process of concept invention begins by feeding the mapper with two concepts. In Divago, the choice of what concepts to use is either given by the user or by a random process. Every concept is kept in the knowledge base and each one is defined through different kinds of representations, namely concept maps, rules, frames, integrity constraints and instances. Although based on the symbolic paradigm (Prolog language), Pereira argues that the same mechanisms could be applied to other paradigms, like neural networks. Pereira's formalization considers concept maps as semantic networks describing concepts or domains. More specifically, each concept map is a graph representing concepts, and the arcs represent the relations among concepts.

Divago itself does not distinguish concepts from domains, meaning that every domain is by itself a concept and every concept can be seen as a domain. Hence, the distinction between concept and domain depends on granularity and is subjective – dependent on the user's input.

Summarizing, concept maps are viewed as the factual part of a concept's representation. Considering reasoning, rules, frames and integrity constraints constitutes the inferential part of the representation. In this context, rules are defined to explain a concept's inherent causality or specific heuristics. Rules are defined by their domain, name, positive and negative conditions and conclusions to be added and removed, if the conditions hold. Considering the whole process, rules can be applied to the inputs, before the process or to the blend, after its construction.

Frames share the same syntax of rules, but they have a different semantics in the system. Usually they describe meta-level concepts integrated to a specific situation, structure or relation (such as cause-effect), tying a set of elements into a single one – intuitively broader and more abstract. Syntactically, frames represent conditions that the associated concept must satisfy. When it does, it is considered that the concept integrates the frame. Semantically, they can be specified as goals and, thus, help structuring the blend towards meaningful results. Inside Divago's process, in a similar way than the utilization of rules, frames are also applied during elaboration.

Accordingly to the frame's functionality, it is classified as organizing, pattern identifying or transforming. A frame is considered as organizing if it determines the whole structure of a concept's concept map. It is considered as pattern identifying if it matches a pattern within a larger concept map. Finally, transforming frames specify specific transformations that may occur during blending.

Our study of Divago indicates that the system itself does not note those distinctions among kinds of frames, they are mostly used by the developer, in order to properly control the process and to analyze the results. In the experiments described in (PEREIRA, 2007), the author specified generic frames representing relations such as analogy, day compression and role projection.

Another kind of representation part of Divago's knowledge base is integrity constraints. This serve to specify logical impossibilities, such as defining that something cannot be dead and alive at the same time. Each constraint consists of the domain/concept where it applies plus logical expressions to be satisfied and unsatisfied (positive and negative sets).

Finally, Divago allows the definition of instances, which allows the user to assign concrete values to parts of the concept's specification (elements of the concept map). Considering all those representations allowed in Divago, a concept is defined by its theory and instances. A concept theory regards concept maps, frames, rules and integrity constraints related to the concept. Intuitively, instances represent the set of concept's instances. Besides different concept theories and instances, the knowledge base also maintains a generic domain, in which generic frames and special kinds of map's relations are specified.

Having the concepts, the mapper provides a structural alignment between them. The general idea of this component is to implement the cross-space matching of concept blending. Recalling that, inside Divago, each concept is associated to a concept map, given two concepts, the mapper finds a set of mappings between their concept maps, each pair having one element from each map. In this implementation, a spread activation (QUILLIAN, 1968) algorithm (using the flood-fill technique) looks for the largest isomorphic pair of sub-graphs inside the concept maps. Two graphs are considered to be isomorphic if they have the same relational structure (arcs), without considering the elements (nodes). Another aspect of Pereira's algorithm is that it begins with a random pair of elements, so, the perfect mapping (largest isomorphous sub-graphs) is not guaranteed.

Given a pair of concepts and a mapping between them (either generated by the mapper or by any other source), the blender module generates the set of all possible blends (represented by a set of projections). Each projection is a non-deterministic operation that maps an element $x$ from a concept map $CM$, and the respective counterpart $y$, when available by the mapping, to a new element in the blend, which is either $x$, 0, $x|y$ or $y$. The compound $x|y$ can be read as both $x$ and $y$ at the same time (PEREIRA, 2007). Thus, a blending projection defines for each element of CM, what its correspondent will be in the blend. When element x has a counterpart (y), it can be projected as a copy of itself (x), as a compound (x|y), as a counterpart (y) or not be projected at all (0). Figure 13 illustrates these projections, with the horse element and its counterpart bird, mapped by the relation M. In this case, the projections are horse, horse|bird, bird or nothing.

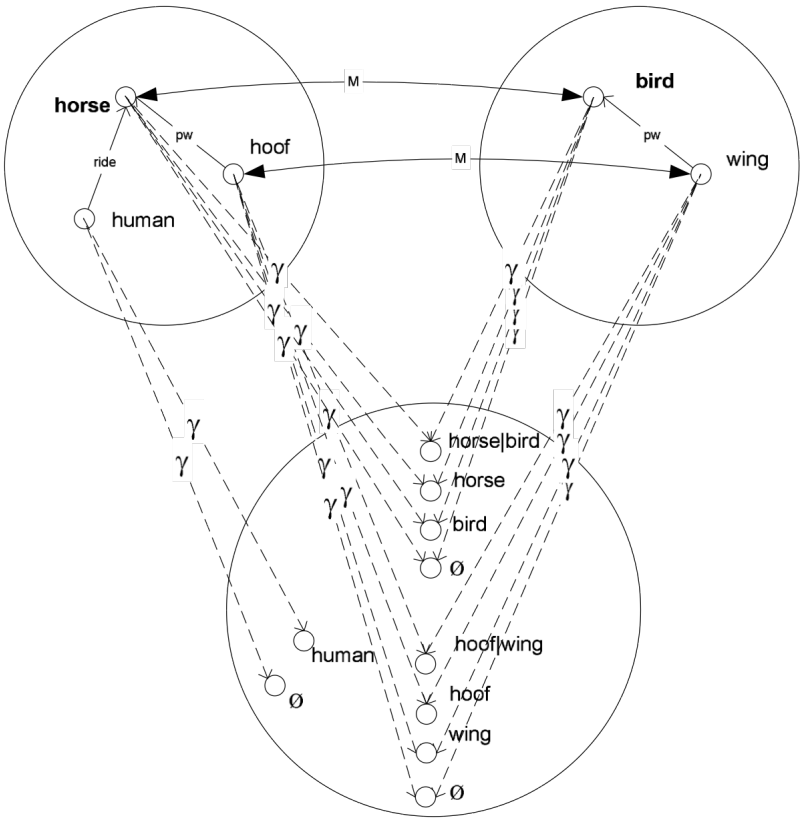


Figure 13 Blending projection applied to two concept maps (PEREIRA, 2007)

Pereira (2007) specifies an algorithm that transfers the knowledge from the inputs to a temporary space, called blendoid, which have all the possible projections associated with the respective knowledge elements. According to the blending theory, this stage would be the composition of the blend.

Continuing the data flow of Divago, the blender will provide a blendoid (representing the search space of all possible blends) to the factory that will search this space trying to find the best blend. A new concept, represented by the selected blend, is the output of the factory module. Relating to the blending theory, the factory provides an implementation of the selective projection, choosing which elements from the input will be projected to the blend, and how. Going to a more detailed level, Divago's factory implements selective projection through a genetic algorithm that performs search over strings representing the possible projections (the blendoid). Pereira (2007) presents an overview of the complexity of the search, which is, for an input of two concept maps and a mapping of size $s$, $4^{2s}$ x $2^{l-2s}$ where $l$ represents the elements from the inputs.

Each sequence of projections is represented by the individuals and the fitness function – phenotype – is based on blending's governing principles (provided by the constraints module) and on integrity verification by the elaboration module. The algorithm uses roulette-wheel selection, prioritizing individuals with high fitness. In consequence of the genetic algorithm, the best solution is not guaranteed.

When the factory receives the blendoids, they have already been pre-processed by the constraints module, which evaluates each possibility according to the governing principles. Each principle supplies an evaluation value that will be part of a weighted

sum, resulting in the fitness value of each individual (set of projections). The user gives the weight of each principle. Pereira makes clear that his concretization of the governing principles regard solely his model, it is not generic and was not based on measurements from cognitive experiments. Under Divago, each governing principle is formalized as a function applying some sort of comparison. In the case of integration, pattern completion, unpacking and relevance the formalization uses frame comparisons. Topology considers the amount of relations from the input maps that were projected without modifications. Maximization of vital relations uses the amount of vital relations present in the blend (compression is not considered and the modeling of VR is like any other relation, with the difference that a set of labels is defined for them). Web depends on values for topology and unpacking.

Another input to the factory component is made by the elaboration module, responsible for providing blending elaboration and completion. In theory, elaboration and completion occurs after the blend is ready, but Divago considers those stages during blend selection. Thus, elaboration and completion have equal importance relating to the other considerations of the selection. The elaboration module runs the respective rules and frame conclusions and also attempts to complete frames using pattern completion. First, rule-based elaboration takes place, executing the rules from the generic domain and then specific rules applicable to the blend's domain. If a rule's conditions hold, the engine executes the respective consequences, already present in the rule's definition. Frame-based elaboration verifies which frames are integrated by the blend and then executes their consequences, just like the rules.

Considering the divergent and convergent strategies described by Pereira (2007), Divago actually implements these strategies at the same time, through the factory and elaboration modules.

## 2.5 GRIOT and Algebraic Semiotics

Zhu and Harrel (2008) describe GRIOT, an interactive narrative system that uses concept blending and an intentionality scale to control the system. An algorithm called Alloy, which implements the algebraic semiotics formalization (GOGUEN, 1999; GOGUEN; HARREL, 2004), provides the creative and imaginative part of the system. The algebraic semiotics approach was one of the first formalizations of the blending theory and represents another important related work (research stage S.A2). Although a partial formalization, the approach of using semiotics is unique and has its computational counterpart, providing us with another system to compare with.

Applied to narrative generation, imagination (provided by Alloy) is used to integrate memory from dreams with the current situation. Thus, providing what the authors call daydreaming narratives. Intentionality plays a secondary role on the described system. It is characterized by a numeric scale defining how interactive the system will be. Such intentionality scale also defines the proportion of daydreams relative to the main narrative, the proportion of automatically selected actions against user-selected actions and the proportion subjective output (generated through blending) relative to previously defined structures of descriptive exposition.

Goguen's algebraic semiotics approach formalizes blending by adopting algebraic semantics to describe the structure of complex signs and the blends from these structures. The basic notion of algebraic semantics is a theory, consisting of type and operations and the respective subtype declarations and axioms (ZHU; HARREL, 2008).

Related to the basic notion of theory, is the conceptualization of a semiotic system (also known as semiotic theory or sign system), which adds level ordering to the types and priority ordering on the elements of each level (establishing a hierarchy among the signs). The priority is defined by a sign's constructor, which represents the rules for combining signs resulting in a new sign of a different kind. In Goguen's (1999) formalization, constructors are further specified by a set of parameters (e.g. a "cat" sign on a computer screen can have parameters for defining its size and location on the screen, but do not change the cat's identity). Given these basic structure of algebraic semantics, it is possible to define semiotic theories such as a book theory: book can be the top sort (type), chapter the secondary sort, head and content tertiary sorts, and title and page number fourth level sorts. In this example, one book's constructor may build chapters from their heading and content, while another builds page heads from a title and page number. Among the constituents of head, title has priority over page number, and among those for chapter, head has priority over content (GOGUEN, 1999; ZHU; HARREL, 2008).

In addition, Goguen (1999) uses sign systems to represent blending's conceptual maps. Next, Goguen specifies semiotic morphisms, which provide a way to describe the dynamics (mappings, translations, interpretations and representations) of signs in one system to signs into another system. Ideally, a semiotic morphism preserves as much of the structure of the source system as possible. But, since the ideal is not always practical, Goguen defines the notion of partial morphisms.

Hence, morphisms are defined as functions or predicates that provide mappings from sorts to sorts, sub-sorts to sub-sorts, data sorts to data sorts, constructors to constructors, and so on. These morphisms are specified by rules, defining also their properties (identity, association, isomorphism and inversion) and how to compose them. Given the level of preserving the source's definition, another formalization is of the notion of morphism's quality. Considering all the constructs of a semiotic theory, concept blending is formalized as semiotic morphisms between inputs, among inputs and generic space and, among generic space, input and blend.

Zhu's implementation of concept blending based on algebraic semiotics produces blends based solely on the structure of the spaces, without considering the meaning of the signs. The algorithm was implemented in LISP, using depth first strategy over two binary trees representing the possible relations among inputs and generic space. Since the algorithm considers only structure, data sorts and constants are not identified (GOGUEN; HARREL, 2004). Zhu and Harrel (2008) state that only governing principles related to structure were implemented on the algorithm. In conclusion, Zhu and Harrel argue that the main contribution of his model of blending is as an "experimental formal tool for precisely representing and testing structural aspects of concept blending", rather than a cognitive model.

## 2.6 Agent Adaption

In this section we present approaches based on planning to provide agent adaptation. Such works are directly related to goals G2 and G2.1, and also to research stages S.A4 and S.C1. The approaches described here follow a traditional planning perspective to adaptation. Thus, they provide an important source for comparison, since our work suplies adaptations through concept blending, instead of planning.

Considering adaptation to unforeseen situations through further applicability of agent's knowledge, Meneguzzi and Luck, (2008, 2009) describes an approach based on the manipulation of plans. In (MENEGUZZI; LUCK, 2008), an extension to AgentSpeak is defined in order to allow dynamic plan generation and declarative goal representation. Declarative goals are constituted by a conjunction of beliefs desired to be true simultaneously. They also establish triggers to activate the planning mechanism. Essentially, the planning module translates relevant plans to the Stanford Research Institute Problem Solver (STRIPS) (FIKES; NILSSON, 1990) language, allowing the construction of new plans according to the translated operators. If the STRIPS planning find a way to achieve the goal, the resulting plan is added to agent's library. Furthermore, Meneguzzi and Luck (2008) also specify an algorithm to generate the plan's context so that it can be executed with less chance of failing – only the necessary conditions will be part of the context.

Going in a similar direction, but with different goal, Meneguzzi and Luck (2009) specify template plans to deal with norms and also primitives for meta-reasoning over plans. With these two constructs, the idea is to be able to modify the agent's behavior according to the norms it has chosen to follow. The primitives were implemented as internal actions, under Jason's framework (BORDINI; WOOLDRIDGE; HÜBNER, 2007). Therefore, those primitives are used inside the plans definition, allowing the specification of plans that interfere with the practical reasoning. For instance, a plan to prohibit certain actions to be performed.

Subagdja, Sonenberg and Rahwan (2009) describe an architecture for an intentional learning agent. This approach positions learning as any other task to be performed by the agent, allowing learning to be controlled by the deliberation process. Hence, computational resources can be controlled in the usual fashion. According to Subagdja, Sonenberg and Rahwan (2009), intentional learning refers to the cognitive processes that explicitly have learning as a goal instead of learning as an incidental outcome. A requirement of this kind of learning is the awareness of one's own knowledge. In addition, it requires strategies or know-how about how to accomplish a learning goal. Subagdja argues that agent learning is triggered when the agent's reasoner needs to improve its performance in some task. Thus, in this case, the learning framework was not developed to deal with novel situations or problems.

Subagdja implements intentional learning for BDI agents using meta-level plans allowing an introspective operation over the agent's internal state (beliefs, desires and intentions). Such approach is not new and was present on the original PRS implementation (GEORGEFF; LANSKY, 1987; RAO; GEORGEFF, 1991). A meta-level plan contains actions to monitor intentions, control deliberations and manage commitment (modifying current intentions) (SUBAGDJA; SONENBERG; RAHWAN, 2009). Consequently, the representation of learning through meta-level plans begins by defining the conditions that will trigger the learning process. For instance, such conditions can be based on changes in performance, amount of failures and time constraints. These conditions also represent the general learning goal of the plan. In addition to the learning goal, the developer also defines the learning context and applicability, specifying preconditions and/or utility. Finally, the meta-level plan's body specifies the learning activities, such as belief updates, plan modification and intention monitoring.

Considering reasoning, Subagdja's approach implements manipulative abduction. Classical abduction infers plausible causes to explain certain effects – in opposition to

deduction where effects are inferred from a set of clauses. In manipulative abduction, the lack of knowledge is compensated by the execution of actions. It is as if the agent is thinking through doing and not only about doing. Subagdja illustrates the difference between abduction and manipulative abduction with Figure 14. The top most part of the figure illustrates classical abduction, where the agent first constructs a series of actions and then, if reasonable, he executes them. Next, on the bottom of Figure 14, the agent uses the actions themselves to construct the plan. It does not have the complete plan at the beginning, the plan is built while the actions are executed.

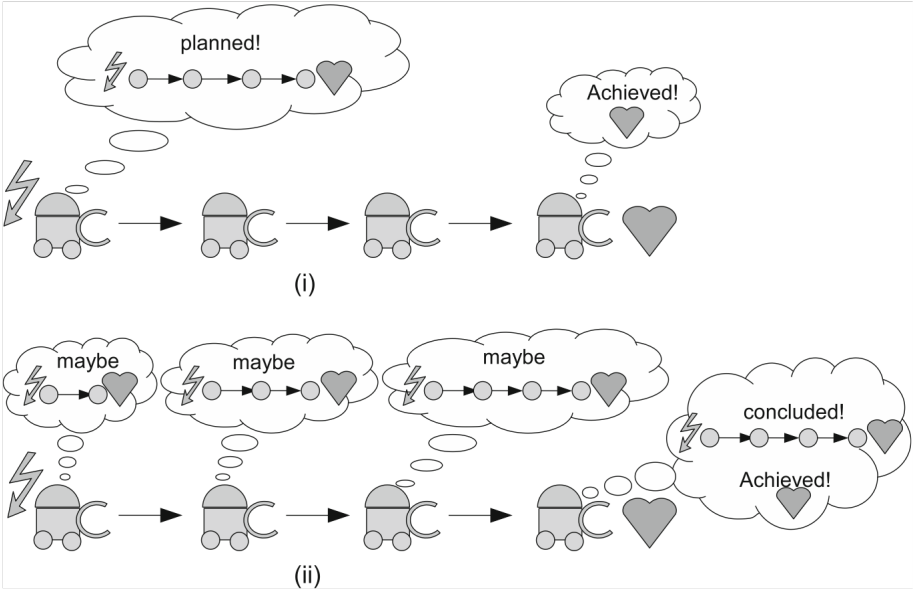


Figure 14 Classical abduction (i) compared to manipulative abduction (ii) (SUBAGDJA; SONENBERG; RAHWAN, 2009)

Moving to a lower-level of abstraction, Subagdja, Sonenberg and Rahwan (2009) specifY two sets of primitives to be used to describe a learning plan. Those sets are defined in general terms, no concretization in an agent language is provided. Although, the authors argue that most PRS-based implementations have the necessary constructs to implement the primitives. The first set of primitives relates to monitoring and managing intentions and overall execution (e.g. achieve, wait, perform, monitor, history, current plan, applicable plans, current goals). Second, a set of primitives to manipulate plans is defined (e.g. create plan, update plan, remove plan, append plan acts, set plan conditions and include utility).

Given this framework, Subagdja, Sonenberg and Rahwan (2009) describe experiments developed using the JAM agent architecture (HUBER, 1999). The experiments illustrate how the learning framework can be used to implement domain-specific learning strategies. It is clear that the developer is left with the task to define the learning plans based on the primitives and the general intuition of manipulative abduction. As future work, Subagdja, Sonenberg and Rahwan point in the direction of the specification of learning design patterns.

Brenner and Nebel (2009), propose the continual planning approach, which implements manipulative abduction in a different way. Differently from (SUBAGDJA; SONENBERG; RAHWAN, 2009), Brenner's focus is on planning itself, rather than learning. This approach to continual planning specifies a balance between planning and action in the form of a planning algorithm that manipulates constructs defined in MAPL

(BRENNER, 2003). Thus, agents are capable of deliberately postpone parts of their planning process in order to prioritize information gathering, relevant for the later refinement of the plan.

The research presented by Leite and Soares (2006) specify an agent architecture that allows the modeling of the agent evolution. In this context, evolution regards the changes on the agent's knowledge representation and reasoning in response to environments where change occurs in several levels (e.g. not only the facts, but also the rules that govern an environment may change). Although not explicitly characterized as a work on agent adaptation, we observe that the framework can be applied to this end.

## 2.7 Summary

Concluding our related work presentation, the works that are directly related to ours are the ones that propose computational models of blending. Under computational creativity, such models can be positioned under general creative models. Inside Figure 15, the contextualization on computational creativity is depicted by the ellipsis on the left side of the Figure. On the left side of Figure 15 we also summarize the main streams and works on computational creativity. Another stream of related works consider agency and, more specifically, approaches to agent adaptation. Our presentation of that field focused on rational – or traditional – strategies to allow the agent alternatives of action to unpredicted situations. Under adaptation, our work proposes a creative approach, differing from the state-of-the-art on the possible applications of the model and, mainly, on the kind of reasoning applied to generate the alternative options. On Figure 15, the ellipsis on its rightmost side depicts our context under agency.

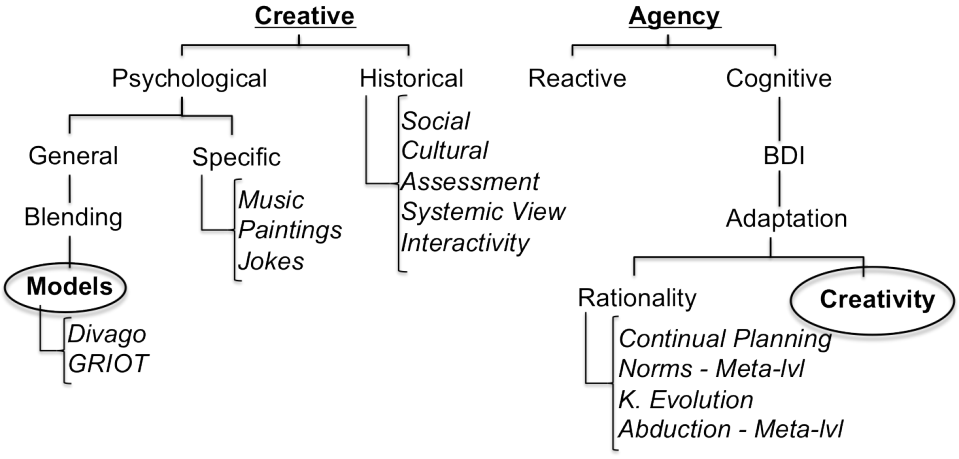


Figure 15 Relationship between our work and the state-of-the-art

Going further on the relations between this work and computational models of concept blending, we compare the models considering which elements of the blending theory are specified, which applications are described and which knowledge representations paradigms were adopted. Table 2 presents a summarized comparison of the blending models according to those criteria.

Table 2 Summary of computational models for concept blending

| Model | Elements of Blending | Applications | KR&R |
|---|---|---|---|
| Divago | Constitutive principles governing principles (partial) | Concept generation | Logic programming (PROLOG) and |

| | | | |
|---|---|---|---|
| | Vital relations (partial) | | conceptual maps |
| Griot | Constitutive principles (partial)<br>Governing principles (partial) | Narrative generation | Sign system,<br>LISP |

# 3 CONCEPT BLENDING MODEL

In this section we present a computational specification of the blending process using operational semantics. This model focuses on the constitutive principles and on the general operation of blending. Therefore, vital relations and governing principles are not present in our model. We impose these limitations to the model since those elements require a broader view of cognition that cannot be achieved considering only autonomy. For instance, the governing principles of pattern completion and relevance are directly related to the evaluation of the decisions on the environment and on the performer as well.

Thus, without considering learning, we could model those principles only superficially. However, considering blending as a part of a broader cognitive structure and modeling the remaining elements from blending, constitute the main future work of this research. With regard to our scope limitations, first we specify a computational model of blending (Section 3.1). Then, we describe how adaptation can be specified with regard to the blending model (Section 3.2). In a similar fashion, a recommendation system based on blending is defined on Section 3.3.

## 3.1 Specification in operational semantics

Concept blending is defined as a set of constitutive and governing principles applied to two conceptual spaces (input spaces) resulting in a new conceptual space, called blend. Theoretically, blending produces new knowledge, gives new meaning and constitutes the fabric of our imaginations.

Our computational model of blending begins with the definition of a mind *M*, which represents an agent's available knowledge. Under blending, knowledge is represented by concepts and their relations (set of concepts *C*). Besides that, there might also be available organizational knowledge that can be applied to group concepts into conceptual spaces with specific meaning, called organizational frames. In order to avoid misinterpretations regarding Minsky's frame representation (MINSKY, 1974), we will call those blend-related frames as organizational schemes, denoted by *O*. Thus, a mind is defined by:

$$M = \langle C, O \rangle$$

And an organizing scheme is defined by its terminology that specifies the concept's hierarchy (*TE*), the respective classes (*CA*) and properties (*PA*) assertions:

$$O = \langle TE, CA, PA \rangle$$

In concept blending, a concept can be anything – incorporating all kinds of perceptions and elaborations from our imagination. Our approach to model this very

broad conceptualization is to consider a concept as any computational resource annotated (described) with terminological information. Therefore, a concept is symbolically defined as an individual from a description logics (DL) representation (BAADER; HORROCKS; SATTLER, 2007). Specifically, we adopted the OWL[1] language (MOTIK, B. et al.. 2008) to provide the descriptive syntax for the concepts.

Essentially, an individual is defined by assertions representing either object or data properties. Object properties allow the association of individuals, while data properties associate direct values (e.g. string, integer, bytecode) to an individual. Consequently, a concept c is constituted by sets of data and object properties (*DP* and *ObP*, respectively). Besides, an individual may also have class assertions specifying its membership to a set of classes (*CA*).

$$c = \langle DP, ObP, CA \rangle$$

Given this configuration, a conceptual space *CS*, can be defined as any sub-set of *M*, $CS \subseteq M$. Granularity, in this case, depends on the current situation and, consequently on the purpose of the blend. The blending theory does not elaborate on how conceptual spaces are produced.

According to the blending theory, the resulting blend *B*, may contain concepts that were not part of the input spaces nor of the whole set of concepts, *M*. Thus, although *B* is a conceptual space, it does not adhere to the rule that a conceptual space is a sub-set of *M*. In fact, concepts from *B* might be added to *M*. Consequently, we loosely define a blend as a set of concepts (*C*) and an organizing scheme (*o*).

$$B = \langle C, o \rangle$$

The blending process is defined by a set of operations that can be performed on two input spaces – conceptual spaces named $I_1$ and $I_2$ ($I_1, I_2 \subseteq M$) resulting in a blend B. Actually, it can be viewed as an operator, that, when applied to two conceptual spaces, will generate a new one according to the blending theory.

The operations performed (denoted by the set *OP*) are the constitutive principles of the theory: modification, completion and composition – which may be applied during the selective projection – and elaboration, which is executed after the blend is ready. Selective projection itself is considered as an operation under our model. Not officially part of the constitutive principles is the establishment of counterpart relations between the input spaces. Counterpart relations are established given the set of defined comparison functions, *CF*. Finally, BC refers to configuration parameters (blend typology – $\kappa$, element selection function – $\alpha$, a stopping condition – $\phi$, and an operation selection function $S_{op}$). Hence, the blending process is initially specified as follows:

$$BP = \langle I_1, I_2, OP, CF, BC \rangle$$

$$BC = \langle \kappa, \alpha, \phi, S_{op} \rangle$$

$$\kappa \in \{\text{simplex, mirror, single} - \text{scope, double} - \text{scope}\}$$

From the theory, it is unclear when and how to apply each constitutive principle given a particular configuration in terms of input concepts and their relations to the

[1] Web Ontology Language: www.w3.org/TR/owl-features/

other input. Hence, we focus more on the definition of the operations rather than their order of execution. Nonetheless, some ordering must be provided, thus, we define a selection function $S_{op}$, which given the inputs, their counterpart relations and a set of possible operations, chooses a specific action to be performed. Pereira (2007), uses a different approach, generating all the possible blends and then searching this space for a blend that better satisfies a given criteria. In a certain way, our selection function plays the role of Pereira's criteria. But, at this point, we have no evidence to support a full account of when each operation must be performed given a certain situation. The set OP of available operations is specified by the set of modification functions $\mathrm{M}$, a completion function *co* and a composition function *cm*. Elaboration is left out since it is executed after the blend is ready.

$$OP = \langle \mathrm{M}, co, cm \rangle$$

The modification operation is applied to a single concept performing changes on it. Ideally, for each type of concept representation, we would have available a modifier function $\mu$ that changes the concept in some way while keeping its membership to its original type. The set $\mathrm{M}$ represents all the available modification functions, $\mu$. Thus, given the selection to execute a modification function to a concept *c*, and the availability of the respective type specific function, rule Mod1 is applied. If there is no type specific modification function available, rule Mod2 is applied. This rule attempts to apply a function directly with on raw representation, given that there is one available.

Modification rule Mod1:

$$\frac{S_{op}(OP) = \mu \quad \mathrm{M}\left(type(c)\right) \neq \{\}}{c \rightarrow c'}$$

where

$$c' = \mu(c)$$

Modification rule Mod2:

$$\frac{S_{op}(OP) = \mu \quad \mathrm{M}\left(type(c)\right) = \{\}}{c \rightarrow c'}$$

where

$$c' = \mu\left(rawType(c)\right)$$

Another possible operation is completion (*co*) where the intuition is to bring information from sources related to the space to complete a given concept. Whether a concept is incomplete or not is always in respect to other representations of the same type. Thus, we ground our completion operation on the possible organizing schemes of the considered space (rule Comp1). In the case that the organizing scheme does not contain such type, we look into the concept's declaration verifying if there is a type assertion and, that this assertion holds for rest of the concept's declaration.

Completion Rule Comp1

$$\frac{S_{op}(OP) = co \quad isMember(c,O) \neq \{\}}{c \rightarrow c'}$$

where

$c' = c \cup assertionsFrom(T)$

$T = membership(c,O)$

Completion Rule Comp2

$$\frac{S_{op}(OP) = co \quad declaredType(c) \neq \{\}}{c \rightarrow c'}$$

where

$c' = c \cup assertionsFrom(D)$

$D = declaredType(c)$

Composition ($cm$) is a constitutive principle that aims at joining two concepts into a single one. Although Fauconnier and Turner, 2003 shows only examples of composition between two, they never state that it cannot be executed with several concepts. Here, we consider only composition between two concepts. A composition may hold certain properties from one concept while bringing the remaining from the other. According to the blending theory, composition may occur with any two concepts from the input spaces, related or not.

Essentially, we define the composition in terms of the concepts' asserted properties. We also consider the relation between the concepts – if assertive – into the composition. This principle makes use of the element selection ($\alpha$) function that here takes the form of a random function. Our idea is that given more knowledge about the current domain, the $\alpha$ function might be customized to represent some of heuristic or tendency.

Composition rule

$$\frac{S_{op}(OP) = cm \quad assertionsFrom\left(c_1,c_2,relations\left(c_1,c_2\right)\right) \neq \{\}}{\left\langle c_1,c_2\right\rangle \rightarrow c'}$$

where

$c' = newConcept\left(\alpha(A)\right)$

$A = assertionsFrom\left(c_1,c_2,relations\left(c_1,c_2\right)\right)$

One thing that comes to mind when we consider applying this kind of operations into a symbolic representation is: will the result be useful, or, will it make sense? In our opinion, if the resulting concept from a composition actually is useful at a given situation is not for the blending process to decide. We consider that such evaluations should be made inside a broader cognitive context, possibly integrating learning, embodiment and other cognitive properties into a single structure.

Returning to the blending process, the establishment of counterpart relations between the inputs can be seen as the first step of the process. Thus, the initial step is triggered by the rule *defCPR* to define the inputs' counterpart relations Briefly, the intuition is to make explicit the relations between the concepts from $I_1$ and $I_2$. As most of the elements in the blending theory, any kind of relation can be considered.

Fauconnier and Turner (2002) point out to the role of vital relations as potential concept comparators. Given such broad scope of potential relations between concepts, we define concept comparison as a function $fc(c_1,c_2)$ that returns a relation between the concepts. We consider that a blending engine has several comparison functions available, each one defined in terms of the types of concepts it is able to handle and also a terminological description of the relation.

Given our descriptive logics usage, we consider that *fc*($c_1$, $c_2$) applies when to concepts $c_1$ and $c_2$ (represented as individuals in DL) are members of the class specified in the function definition (denoted as $t_1$ and $t_2$, respectively). An individual's membership to a certain class can be directly asserted in the A-Box or it can be inferred based on the individual's properties and the representation's T-Box. Class membership is a descriptive logic inference and the reader is referred to Baader, 2007 for a full account of this inference mechanism.

Definition of counterpart relations rule – *defCPR*

$$\frac{appComp \neq \{\} \quad I_1 \cup I_2 \neq \{\}}{\langle I_1,I_2,OP,CF,BC,defCPR\rangle \rightarrow \langle I_1,I_2,OP,CPR,BC,defGen\rangle}$$

*where*

$$appComp = \forall c_1 \in I_1, \forall c_2 \in I_2 \rightarrow CF(t_1 = type(c_1), t_2 = type(c_2))$$

$$CPR = \forall c_1 \in I_1, \forall c_2 \in I_2, \forall fc \in appComp \rightarrow fc(c_1,c_2)$$

$$type(c) = membership(c)$$

Following the establishment of the counterpart relations (denoted by *CPR*), he rule to configure the generic space (*defGen*) is applied. Here, the generic space contains common aspects between the inputs. Thus, it is constituted by the intersections between the inputs' concepts and organizing schemes.

Generic space definition rule – *defGen*

$$\frac{O_1 \neq \{\} \wedge O_2 \neq \{\}}{\langle I_1,I_2,OP,CPR,BC,defGen\rangle \rightarrow \langle I_1,I_2,OP,CPR,GEN,BC,Blend\rangle}$$

where

$$I_1 \rightarrow \langle C_1,O_1\rangle$$

$$I_2 \rightarrow \langle C_2,O_2\rangle$$

$$GEN = \left(O_1 \cap O_2\right) \cup \left(assertions(C_1) \cap assertions(C2)\right)$$

Now that the counterpart relations and the generic space are specified, the process continues by applying the constitutive principles and selectively projecting the elements to the blend. These operations are performed according the desired typology of the blend, specified by the blending configuration ($\kappa$). In general, the typology determines the conceptual space where the constitutive principles are applied. Therefore, before presenting the blending rules we define the function for applying the constitutive principles, represented by the set of operations *OP*. The function is applied to a conceptual space $cs \in M$. Another important parameter received by the function is the current blending configuration *BC*. Considering the stopping condition $\phi$, the constitutive principles are applied to *cs*, introducing changes (defined by $S_{op}$) on the space until $\phi$ is satisfied.

Application of constitutive principles – *applyCP*

$$applyCP(cs,BC) = \begin{cases} \text{while } \phi \\ \left\langle \alpha\left(c_1 \in cs\right), \alpha\left(c_2 \in cs \wedge c_2 \neq c_1\right), S_{op}(OP) \right\rangle & \text{if } S_{op}(OP) = cm \\ \left\langle \alpha\left(c_1 \in cs\right), S_{op}(OP) \right\rangle & \text{otherwise} \end{cases}$$

Following the blending typology defined by Fauconnier, 2002 we specify four possibilities for blending. A simplex blend considers a simple frame and role integration. In this case, the first input space is regarded as the frame and, thus supplies $O_1$ as the blend's organizing scheme while $I_2$ supplies the values to fill $O_1$'s roles (property assertions denoted by $p$).

Simplex blend rule

$$\frac{\kappa = simplex \quad O_1 \neq \{\}}{\left\langle I_1, I_2, OP, CPR, GEN, BC, Blend \right\rangle \rightarrow B}$$

$where$

$I_1 \rightarrow \left\langle C_1, O_1 \right\rangle$

$I_2 \rightarrow \left\langle C_2, O_2 \right\rangle$

$BC \rightarrow \kappa$

$B = terminology(O_1) \cup roles$

$roles = \forall p \in O_1 \rightarrow \alpha(simplex \cup applyCP(simplex))$

$simplex = c \in C_2 \cap p$

Next, Fauconnier and Turner (2002) specify the mirror blend where all the spaces (inputs, generic and blend) have the same organizing scheme. Thus, the blend will maintain the organizing space but will selectively project the result of applying the constitutive principles against all spaces.

Mirror blend rule

$$\frac{\kappa = mirror \quad \left(O_1 = O_2 = O_{gen}\right) \wedge O_1 \neq \{\}}{\left\langle I_1, I_2, OP, CPR, GEN, BC, Blend \right\rangle \rightarrow B}$$

where :

$I_1 \rightarrow \left\langle C_1, O_1 \right\rangle$

$I_2 \rightarrow \left\langle C_2, O_2 \right\rangle$

$GEN \rightarrow \left\langle C_{gen}, O_{gen} \right\rangle$

$BC \rightarrow \kappa$

$B = terminology(O_1) \cup assertions$

$assertions = \forall p \in O_1 \rightarrow \alpha(mirror \cup applyCP(mirror))$

$mirror = c \in \left(C_1 \cup C_2 \cup C_{gen}\right) \cap p$

Considering different organizing frames for the inputs, single-scope blends choose only one of the organizing schemes to be projected to blend (the source). The other input (called target) is used to fill the source's properties.

Single-scope blend rule

$$\frac{\kappa = single - scope \quad O_1 \neq \{\} \vee O_2 \neq \{\}}{\langle I_1, I_2, OP, CPR, GEN, BC, Blend \rangle \rightarrow B}$$

where :

$I_1 \rightarrow \langle C_1, O_1 \rangle$

$I_2 \rightarrow \langle C_2, O_2 \rangle$

$BC \rightarrow \kappa$

$B = terminology(O_{source}) \cup assertions$

$source = \alpha(I_1, I_2)$

$target = (I_1 \cup I_2) \not\subset source$

$assertions = \forall p \in source \rightarrow \alpha(C_{target} \cup applyCP(C_{target}))$

Finally, double-scope blends use both organizing schemes to form a new one. Thus, this kind of is, potentially, the one that might generate the most surprising blends.

Double-scope blending rule

$$\frac{\kappa = double - scope}{\langle I_1, I_2, OP, CPR, GEN, BC, Blend \rangle \rightarrow B}$$

where :

$I_1 \rightarrow \langle C_1, O_1 \rangle$

$I_2 \rightarrow \langle C_2, O_2 \rangle$

$GEN \rightarrow \langle C_{gen}, O_{gen} \rangle$

$BC \rightarrow \kappa$

$B = organization \cup concepts$

$organization = \alpha(O_1 \cup O_2 \cup O_{gen} \cup appCP(O_1 \cup O_2 \cup O_{gen}))$

$concepts = \alpha(C_1 \cup C_2 \cup C_{gen} \cup appCP(C_1 \cup C_2 \cup C_{gen}))$

Considering the blending semantics defined in section, its utilization requires the specification of terminological and assertive knowledge from the domain. Besides, specific comparison and modification functions might be defined for a proper operation of the blend. We do not state that such functions are mandatory because we can always reduce a representation to simple data types (e.g. char, real and bytecode). Clearly, this comes with a price. When performing generic operations directly in original representation, we ignore the concept's structure and any other semantics and rules that might be attached to it. Moreover we risk corrupting the concept's representation.

Despite its importance to generate working blends, concept-specific functions and operations also have their price. The issue with this design is that we add more tasks to the developer, difficulting even more the development of AI systems. Fortunately, the result of hard work pays well. Having specific functions and the respective terminology and assertions regarding a representation actually allows our system to work with

heterogeneous knowledge representation. Hence, we go a little further on the blending model making it closer to the theory itself. According to our theoretical model, it is possible to integrate symbols, images and other representation paradigms in a single blend. Another feature of our model is the specification of blending typology allowing the utilization of different kinds of blends. Actually, none of the studied computational models of blending specify Fauconnier's and Turner's typology.

## 3.2 Blend-based Adaptation

Here we describe an integration of the blending model (Section 3.1) with an AgentSpeak agent architecture, provided by the Jason framework. The purpose of this study is to describe how blending can be applied as a mechanism for agent adaptation (Goal 1.1). By integrating blending to a BDI agent architecture we also want to study how inputs for the blending process can be composed (Goal 2.1).

Such integration was achieved by specializing Jason's agent model (Appendix 1), in a way that every intention failure or lack of applicable plans triggers the adaptation mechanism. Using Jason 1.3.3 we also implemented an event listener to provide callbacks to the agent on the event of intention failure (Appendix 2). Another part of the integration is the definition of a descriptive representation of the agent's knowledge. In this case, we did not develop an OWL specification of all the BDI components. Rather, we defined a shallow terminology for our domain of BDI agent adaptation.

Our intuition is that specific agent configurations, both static (beliefs and plans) and dynamic (desires and intentions), are modeled as individuals of that terminology. Therefore, descriptive logics work as a bridge between a specific representation – in this case the agent components – and the blending engine. We consider the agent terminology as a shallow one since we focused more on the vocabulary and basic terms to differentiate concepts than on a big hierarchy and a set of rules and axioms to restrict it. Another reason to follow this line is that the rules associated to the concepts – in the case of BDI – are modeled and used by the Jason reasoning engine.

Observing the terminology, it is clear that we did not consider the full Jason's syntax to represent beliefs and plans in OWL. For instance, according to the Jason implementation of AgentSpeak-L, a belief is constituted by a literal that can be negated and can also represent variables, constants and terms. We did not go into such detail in the OWL representation since in the blending process itself, this information is not used. A more specific terminology facilitates the associations between concepts, and can be further used when the reasoning is also specified on the same abstraction level. In this study the reasoning is separated, it is not manipulated during blending.

Although it is possible to model reasoning in more abstract terms, we realized that such modeling would be too ambitious to cover on this thesis, since it would be necessary an in-depth study on meta-level reasoning (future work). Thus, most of classes' definitions are used directly to specify the properties, making little use of class-specific axioms. An overview of the adaptation terminology is depicted by Figure 16, where each ellipsis represent a class. When an adaptation situation is configured, the OWL terminology is used to create the inputs for the blending engine. Hence, each conceptual space is represented by one individual and, the concepts are represented by the property assertions of each individual (e.g. hasBelief, hasPlan, hasFailedPlan, hasTrigger).

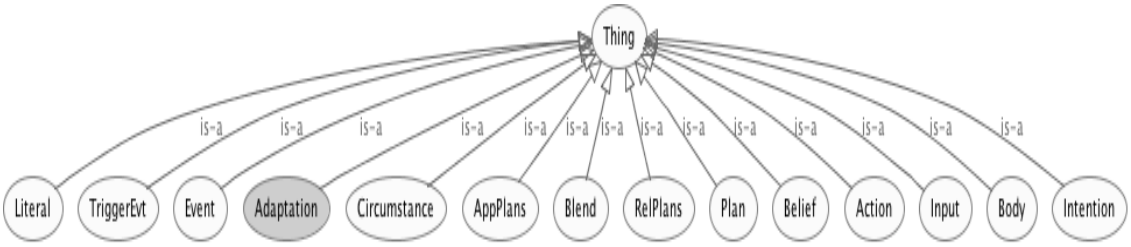


Figure 16 Adaptation Terminology

Although not specified by the blending theory, it is clear that the definition of the input spaces consider, at least, the current situation and the purpose of the blend. Regarding our blending model, we need to specify two inputs in terms of their concepts and organizing schemes. Given that a terminology about the situation and purpose is available, any concept definition can be used as an organizing scheme for an input. Similarly, all the available individuals representing the concepts may be used to compose the space given the chosen organization. So far, we have not envisioned a way to automatically compose the input spaces without some kind of representation (declarative or imperative) regarding a situation or purpose.

Therefore, this study has adaptation as the purpose and failed intention or lack of options as situations. Sometimes, these situations should simply fail and the agent should directly move on with its reasoning. Nonetheless, given those situations, our agent will always attempt an adaptation. Since those situations are detected directly on the agent's reasoning cycle, we do not explicitly define them in a terminology.

However, a class defines adaptation specifying the kinds of concepts that should be available in order to achieve an adaptation. This definition does not specify an output since it would not be coherent to the blending theory, where the desired output is not established. The initial direction is given by the purpose but the output is not explicit. Hence, the following class defines adaptation:

Class: Adaptation
  EquivalentTo:
    and (hasFailedPlan exactly 1 Plan)
    and (hasTriggerEvt exactly 1 string)

Throughout this section we will use a simple agent program to exemplify the blending functions and rules. We developed a single reactive agent that takes his decisions based solely on its current knowledge, with no other kind of decision reasoning. This agent program has knowledge related to ingredients available in his kitchen (beliefs) and also knowledge on how to cook a risotto and sushi rolls (plans) (the full code is presented on Appendix 3). Our example agent uses the modified agent class in order to init the adaptation mechanism. Since we separated the agent model from the agent program the configuration of which class to use is defined on system's configuration file, which can be easily modified.

When the practical reasoning has no available options for a given world configuration or when an intention has failed, we automatically create an individual of the adaptation class. In the case of no options, the property hasFailedPlan is not added to the individual. Consequently, this individual and the definition of the class adaptation constitute a conceptual space and is the input $I_1$ for the blending process, $BP = \langle I_1, I_2, OP, CF, BC \rangle$. Returning to our agent example, the failure of the intention

to cook a risotto – with the plan to add the ingredients being the plan that failed – generates the following adaptation individual:

```
Individual: adaptation1
   Types:
      Adaptation
   Facts:
      hasFailedPlan  p2,
      hasTriggerEvt  "+!cozinharRisoto(X, Y)"^^xsd:string
Individual: p2
   Types:
      Plan
   Facts:
      hasTriggerEvt  "+!adicionarIngredientesMexer(P)"^^xsd:string,
      hasContext  "cozinhando(P)"^^xsd:string,
      hasPlanBody       "?ing(funghi);
                         ?ing(brie);
                         ?ing(vbr);
                         ?ing(manteiga);
                         .adicionaIng(P, vbr);
                         .mexer(10);
                         .adicionaIng(P, fungui);
                         .mexer(2);
                         .adicionaIng(P, brie);
                         .mexer(2);
                         .adicionaIng(P, manteiga);
                         .print("fim do plano addIngMexer")."^^xsd:string
```

Making an analogy with the blending theory, the whole individual adaptation1 represents a conceptual space to be used as the first input. The explicit type declaration stating that the individual belongs to the Adaptation class also represents an assertion of the space's organizing scheme. Finally, the definition of the individual also contains two property assertions, which refer to the concepts that constitute the conceptual space.

As for the second input space, it is possible to also define it as an individual member of an already defined class (an organizing scheme), in the same way that we defined the adaptation one. However, for this scenario we chose not to use an organizing scheme for the second input. We followed this line because, in this context, the relevant organizational information is already modeled in the AgentSpeak semantics. Consequently, the second input, $I_2$ is an individual representing the agent's plan library plus its current beliefs. In fact, the second input could be composed either only by the agent's plans or only by its beliefs. It could also contain more information from annotated beliefs and knowledge from external sources (e.g. the semantic web). Thus, according to our agent example, the second one would be represented by the following individual (we suppressed the plan individuals and listed only part of the beliefs):

```
Individual: cs
   Facts:
      hasPlan  pcs1,
      hasPlan  pcs2,
      …
      hasLiteral  "ing(agriao)."^^xsd:string,
```

hasLiteral "ing(arroz_jap_curto)."^^xsd:string,
hasLiteral "ing(nori)."^^xsd:string,
hasLiteral "panelaFogao(pan_ferro)."^^xsd:string
…

Figure 17 illustrates both inputs in an analogy to the way that Fauconnier graphically represents the blends. In this case, the ellipses represent conceptual spaces and, thus both $I_1$ and $I_2$. Each underlined element characterizes a concept specified by a property assertion.

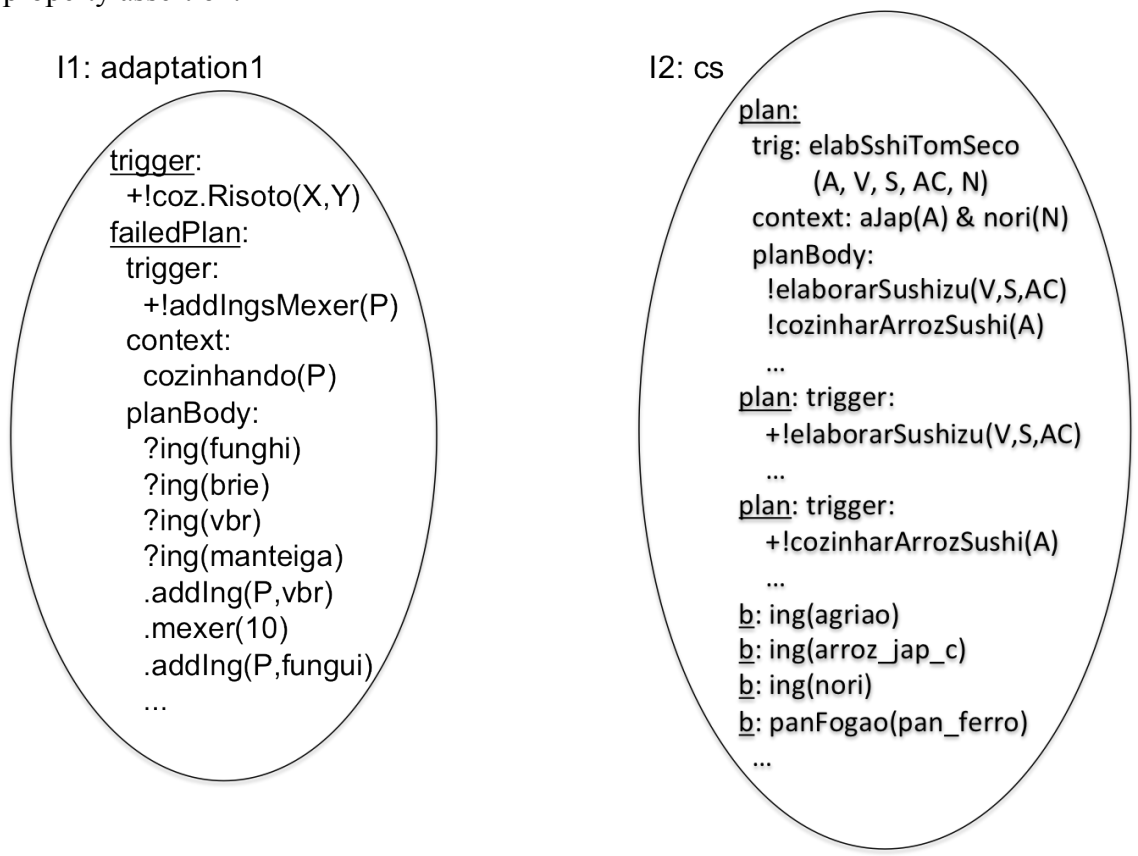


Figure 17 Adaptation inputs

Recalling the components of the blend process $BP = \langle I_1, I_2, OP, CF, BC \rangle$, OP represents the set of constitutive principles to be applied on the input spaces. The constitutive principle of modification actually requires domain-specific information in order to modify the concept without corrupting it. Thus, we developed modification functions to deal with beliefs and plans.

Concept modification might modify the concept in a way that brings great insight into a situation or it might make no sense at all. In our opinion it is not on blending that evaluation plays an important role. Rather, evaluating if a modification is useful or not, besides dependency on the situation, is executed by other cognitive processes, like learning or perceiving.

Hence, we modeled one annotation-based modification function that is used only if a belief is annotated with terminological information – semantically enhanced belief – (using Jason's DL extension defined by Klapiscak and Bordini (2008)). In this case, the function attempts to substitute an individual with another one from the same type (modification by substitution). We consider this function to be annotation-based

because it can be applied to any kind of annotated representations – such as resources from the semantic web.

Modification function for semantically enhanced beliefs (Appendix 4)

$$\mu(b[o]) = \begin{cases} fb \cup \alpha\left(c \in membersOf\left(fb, o\right)\right) \text{ if } b \in CA \\ fb \cup \alpha\left(c \in membersOf\left(ra\right)\right) \text{ if } b \in PA \wedge \text{ if } b \text{ has range def.} \\ fb \cup \alpha\left(c \in membersOf\left(oldTerm\right)\right) \text{ if } b \in PA \end{cases}$$

where

$fb = functor(b)$

$CA = classAssertions$

$PA = propertyAssertions$

$ra = rangeAssertions(b)$

$oldTerm = type(\alpha(terms(b)))$

For instance, the belief "ingredient(short_sushi_rice)[o(kitchen)]" indicates that this sushi rice is member of the class ingredient from the restaurant ontology. Thus, this belief would fall into the first possibility of the modification function. Consequently, the aforementioned belief is modified by maintaining its functor (ingredient) and by substituting short sushi rice for another member of the class ingredient. One possible outcome would be "ingredient(apple)" or "ingredient(pasta)". If the hierarchy is more specialized, the modifications will be closer to the original concpet. For example, if instead of ingredient we considered a hierarchy for food with rice being one subclass, then, the belief "rice(short_sushi)" could be modified to "rice(long_risotto)". A further specified ontology also allows the utilization of upper-classes in the cases where there are no other members of the class being considered.

When the belief represent a property (e.g. hasIngredient(norimakisushi, nori) or hasExpirationDate(nori, 11-2012)), the second option of the modification rule is to use the property's range definition, if available. Considering that the property hasIngredient is defined with a range assertion of classes Ingredient or Food, the specified modification function will change the attributive part of the assertion. Again, we use the members of the classes from the range definition to substitute the attribute. For example, hasIngredient(norimakisushi, nori) may be modified to hasIngredient(norimakisushi, lasagna_pasta). Finally, if the property does not have a range assertion, the function attempts to infer the membership of the attribute and, if inferred, the attribute is modified with another member of the inferred class.

Relying on randomness we also developed one function to deal with beliefs and another to deal with plans. The main issue with this kind of function is that the modifications might result in non-executable plans and in non-grounded beliefs. This issue is related to the symbol grounding problem (HARNAD, 1990; WILLIAMS, 2008; NILSSON, 2007; CREGAN, 2007) where, essentially, symbolic representations are detached from a real observation of the world. Usually, grounding approaches study how to model the link between perception and symbolic representations. Without this kind of information, which is readily available for us humans and, thus, is assumed as given in the blending theory, the modification of a symbol does not yield its imaginative counterpart (based on the perception). In awareness of this limitation, we defined the random modification functions as follows:

Modification function for beliefs (random modification, Appendix 5)

$$\mu(b) = fb \cup rm\left(\alpha\left(terms(b)\right)\right)$$

where

$fb = functor(b)$

$rm = randomModification(term)$

Considering the belief ingredient(dried tomato), the application of this function might result the belief ingredient(tomato) but it might result on the belief ingredient(tofehan_sqc), which possibly can not be used during the execution of plans.

Modification function for plans

$$\mu(p) = rm(te) : rm(ct) \leftarrow rm\left(\alpha\left(actions\left(body\right)\right)\right)$$

where

$rm = randomModification()$

$p = te : ct \leftarrow body$

The next component of the blend process is the set of comparison functions (*CF*). This component is responsible for establishing the counterpart relations between the inputs. Recalling the theory, these relations between concepts from the inputs may be constructed from any kind of connection. According to our interpretation, the counterpart relations are the result of applying certain types of associations over the inputs. For example, it is possible to apply similarity measures, deduction, subsumption and part-of relations to two given concepts from the inputs. Thus, we consider that the associations are realized by applying certain kinds of reasoning on the concepts.

Taking adaptation into account, the role of the comparison functions is to reveal associations between the problem situation and the available knowledge, potentially contributing to a solution. Therefore, the adaptation comparison functions (CF) represent an adaptation strategy to generalize a plan's context, easing its applicability and also the straightforward strategy to look for alternatives given the intention trigger.

Relying on the set of programming interfaces and implementations provided by Jason, the comparison functions were developed with more focus on testing our engine than on providing an in-depth concept analysis. We developed two sets of comparison functions, one focused only on beliefs and the other on plans.

Our first belief-based comparison (Appendix 6) verifies if two beliefs share the same functor. In this function it is assumed the availability of a functor function (represented by *func*) able to retrieve the functor of a belief. Figure 18 illustrates this function by establishing associations between the belief *ing(agriao)* and other beliefs that share the same functor from the other space. Those associations are depicted by the lines bellow the (a) marking. For illustrative purposes we did not depict the functor relations from the belief *ing(arroz_jap)*.

$$fc(b1,b2) = \begin{cases} func(b1) & \text{if } func(b1) = func(b2) \\ \{\} & \text{otherwise} \end{cases}$$

Another belief comparison verifies if one belief contains the other, as a simple metric of lexical similarity (Appendix 7). In this case, we assume that the intersection is

computed regarding each term or list of terms that compose the beliefs. Figure 18 shows an example of the belief comparison considering the literal's terms. The association depicted by a line marked with (b) shows the relation between two beliefs that share a common term ("arroz_jap").

$$fc(b1,b2) = b1 \cap b2$$

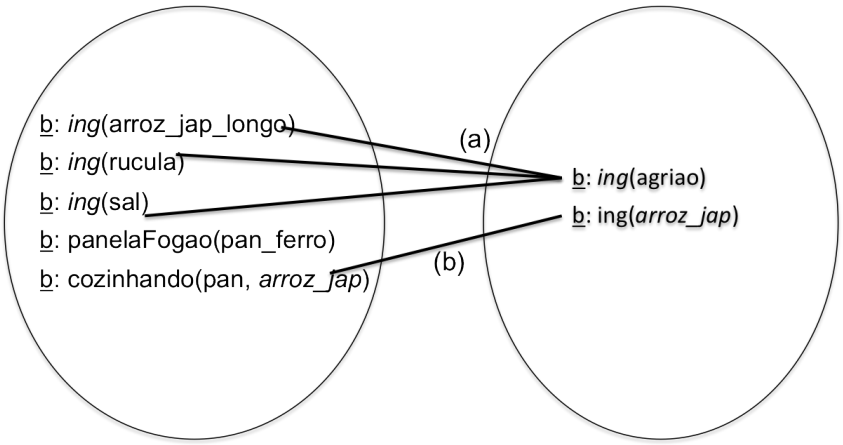


Figure 18 Belief comparison functions

Comparing literals (beliefs and goals) to plans, we check if a plan is triggered by the literal, considering all the possible events (addition or deletion of beliefs, achievement and test goals). This comparison is based on the unification operator $\theta$ (unification) as defined by Bordini, Wooldridge and Hübner (2007). By simulating events on a given literal, this function verifies if an already defined trigger is able to handle the simulated event. In the opposite way, the function analyses if the simulated event would have a candidate plan (Appendix 8). Figure 19 illustrates three associations (denoted by the lines) realized by the trigger comparison. The topmost line illustrates the relation between ing(nori) and the trigger +ing(X), meaning that the addition of that belief would be handled by the +ing event. Similarly, simulating a test goal addition with the belief, would generate an event to be handled by ?ing(X) – illustrated by the middle line. The last relation illustrate the applicability of the function on events generated inside plans, in this case, the event is already constructed and only the unification is performed.

$$tc = \{+!,-!,+?,-?\}$$

$$fc(l,p) = \begin{cases} tc \cup l \text{ if } te \ \theta \ tc \cup l \\ \{\} \quad \text{otherwise} \end{cases}$$

where

$$p = te : ct \leftarrow h$$

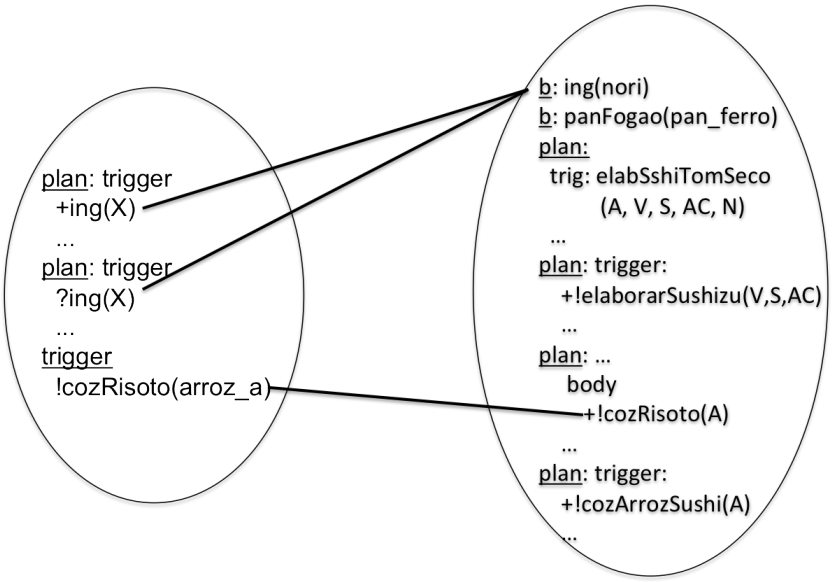


Figure 19 Trigger comparison function

Finally we compare a belief against the plan's context. This comparison can go in several ways, one way is to check if the belief alone satisfies the context (1) Appendix 8 – PlanHelper class. In Figure 20 the association through context satisfiability is depicted by the line marked with (a). The relation is established since the belief cozinhando(arroz_jap_c) is sufficient to satisfy the context cozinhando(P). Another comparison based on a plan's context is to verify if a belief constitutes it, ignoring the remaining of the context's logical expression (2). Hence, this last function analyses only if there is a lexical relation between the given literal and the context Appendix 9. An example of this function is presented on Figure 20 by the relations (depicted by lines) marked with (b). In this case there is a partial relation between the context and the beliefs.

(1)

$$fc(b,p) = \begin{cases} b & \text{if } b\ \theta\ ct \\ \{\} & \text{otherwise} \end{cases}$$

where

$p = te : ct \leftarrow h$

(2)

$$fc(b,p) = \begin{cases} b & \text{if } \left(b \cap ct \neq \{\}\right) \\ \{\} & \text{otherwise} \end{cases}$$

where

$p = te : ct \leftarrow h$

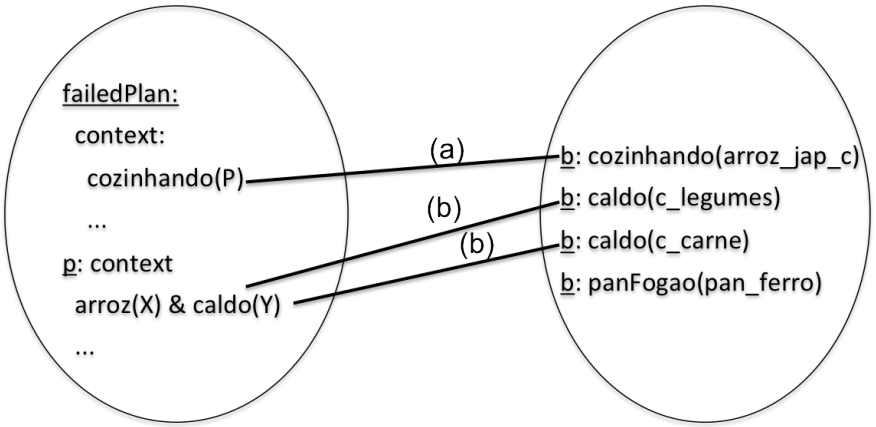


Figure 20 Context comparison function

Following our description of blending components for the adaptation study, the last one is the blending configuration given by the set $BC = \langle \kappa, \alpha, \phi, S_{op} \rangle$. α specifies a decision function applied to choose one concept given a collection of them. Actually, this function represent blending's selective projection, which chooses certain concepts from the spaces and their relations and projects them (after the application of the constitutive principles) to the resulting blend. However, the theory does not elaborate on the criteria for the element selection. In our opinion, this aspect is related to other cognitive functions such as learning from experience, emotions, current paradigm and sensorial feedback. Since, in this work we are focusing on the blending model itself, we specify α as a random selection function:

```
static Object alpha( Collection<?> c ){
        Random r = new Random();
        int i = r.nextInt(c.size());
        return c.toArray()[i];
}
```

One stream of future work on the model actually is to use blending and learning in a single model. Thus, we will be able to study different ways to implement selective projection given the agent's experience. Closely related to α is the function to select a constitutive principle to be applied during the selective projection ($S_{op}$). Hence, in our model the selective projection is modeled by α (selection of concepts) plus $S_{op}$ (selection of constitutive principles). We consider that this function also represents an input from other cognitive functions, and so, it is left undefined by the blending theory. In conjunction with α, $S_{op}$ can be applied to denote domain-specific heuristics or restrictions. For instance, given a domain where modification cannot occur, $S_{op}$ can be modeled to never allow this operation over concepts. On the subject of agent adaptation, $S_{op}$ does not represent any kind of heuristic, rather, like α, it is implemented as a random decision function with the same probability for each operation (Appendix 10):

$$S_{op} = \begin{cases} float\ \mathrm{r} = \mathrm{random}(0,1) \\ \mu & \text{if r} \leq 0.33 \\ cm & \text{if r} > 0.33 \text{ and r} \leq 0.66 \\ co & \text{if r} > 0.66 \end{cases}$$

Another part of the blend configuration is given by $\phi$, which specifies a stopping condition for the process. In the blending theory, there is no condition to suspend the process. Instead, it is assumed that blending occurs all the time in a subconscious level directly integrated with the other cognitive functions. Since our blending specification is not yet part of a broader model for cognition we added a stopping condition to the process. Therefore, this condition can be specified in terms of a domain-specific evaluation function to be applied to current blend or an iteration threshold. Considering our adaptation scenario, it is possible to define an evaluation in terms of option availability, but it does not ensure the quality of the option – which is closely related to the agent's domain. For that reason, we chose to specify the stopping condition as an iteration limit (Appendix 11).

Finally, the last element of the blend configuration is the kind of blend, according to Fauconnier's typology. Although we modeled the blend process following the typology, the specified types do not restrict the process itself. In the theory, a typology is defined to exemplify the most common blend and how they come about in our daily life. Nonetheless, for the purposes of our model the typology allowed us to specify a process that, as the authors themselves state, is non-algorithmic and non-deterministic.

Although it is theoretically possible that blending occurs without the definition of its type, in our adaptation study we restricted ourselves to blending as a process governed by a given type. Considering both the amount of possibilities and the utilization of blending components – which provides a richer example – we applied double-scope blending to the adaptation study (κ=double-scope). Hence, examining the double-scope rule, it is noticeable that it does not have any requirement for its application and the initial configuration of the process follows the same approach as the other kinds of blend.

Thus, the elements described so far constitute the initial configuration for the blend process, $BP = \langle I_1, I_2, OP, CF, BC \rangle$. Looking into the configuration of the blending rules the transition to the blend parts from a slightly different configuration: $\langle I_1, I_2, CPR, GEN, OP, BC \rangle \rightarrow Blend$. Thus, from the initial configuration we first establish the generic space, given the two input spaces. As defined by the rule *defGen*, (presented on Section 3.1), in essence, the generic space contains the elements that are common to both inputs.

Recalling our example agent and its inputs – illustrated in Figure 17 – the only common aspect between the inputs is the fact that both of them contain at least one plan. Although the referred plan and the property assertion are different, the common aspect is that both inputs point to a plan. Since the inputs have different organizing schemes, this aspect is not projected to the generic space. In the blending theory the generic space may contain the blend's purpose, but in a very subjective and abstract way. The purpose sometimes is beyond the organizing scheme. This aspect is not present in our current model and constitutes another part of our future research. Figure 21 illustrates the generic space constructed given the adaptation inputs. Like the input spaces, the generic is depicted by an ellipsis that represents a conceptual space. The dashed lines illustrate the common concepts that originated the generic space.

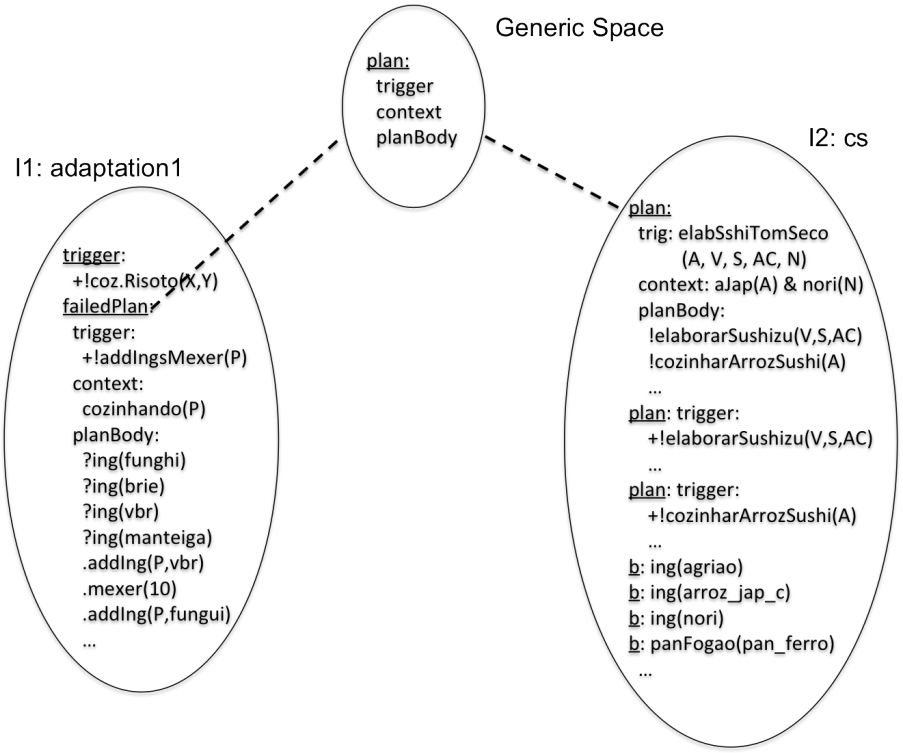


Figure 21 Inputs and generic space for the adaptation example

After the establishment of the generic space, the rule defCPR is applied to retrieve the counterpart relations between the concepts. Given the set CF of comparison functions, the rule attempts to apply, for each pair of concepts, the respective function. If no compatible function is available, it is verified if a more generic function – using the primitive types – may be applied. Hence, in

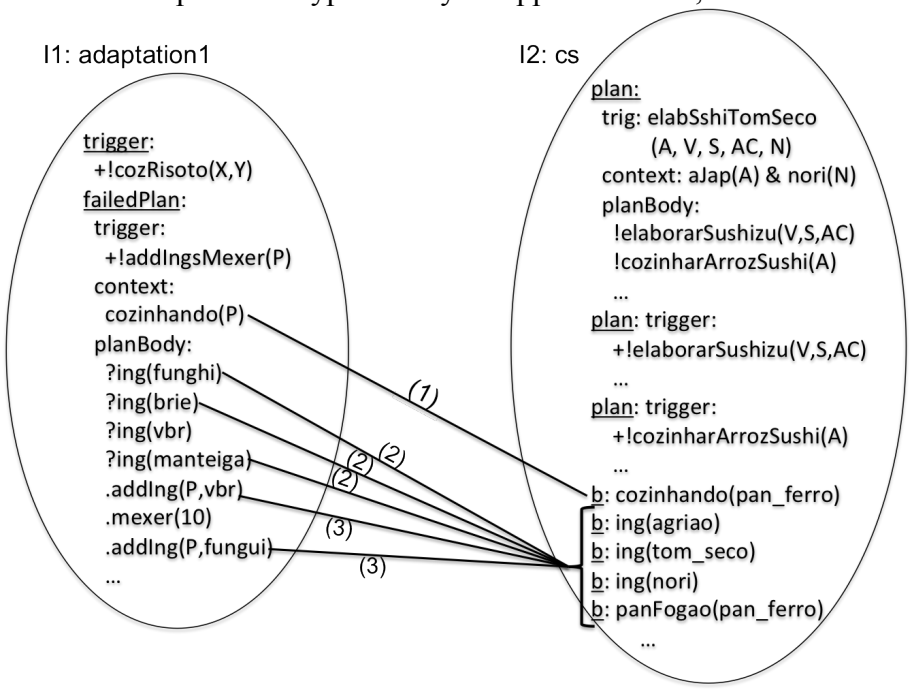


Figure 22 we illustrate the counterpart relations established from the comparison functions defined in this Section. A numbered straight line depicts each counterpart relation.

The first relation (1) shows a contextual relation between a plan's context and a literal that satisfies it. Next, the relations marked with a (2) denote both a literal relation – considering the test goal as a literal instead of a goal – and also a trigger/literal relation since simulating the addition of a test goal with the literals will trigger the test goal. During the establishment of the counterpart relations the descriptive representation is considered only to tell which are the concepts of the space. After that, our implementation of the BDI comparison functions uses the original representation since most of the methods use the Jason API to compute the similarity. This technical detail depends on each implementation, but we believe that more specialized knowledge representations, like multimedia, will also compute the associations using the original reasoner instead of implementing it on top of OWL. Another reason is that, for the blending, OWL has the role of describing the knowledge and thus, does not represent all the dimensions of the knowledge.

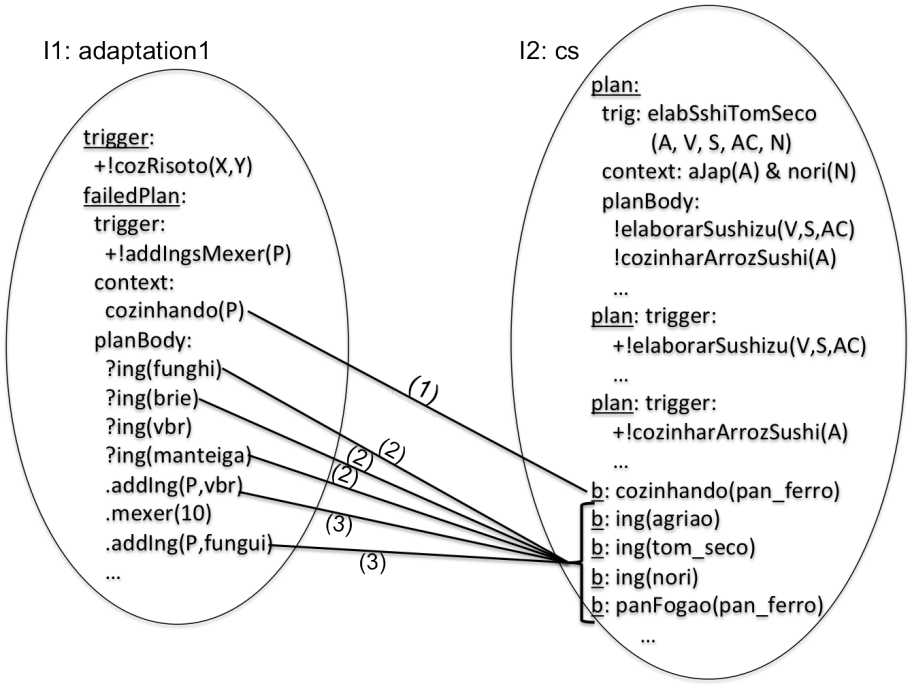


Figure 22 Counterpart relations from the adaptation study

Assuming the definition of the generic space and of the counterpart relations, the configuration to continue the blending is set. Figure 23 illustrates the initial configuration for the double-scope blending rule. Again, the conceptual spaces are depicted by the ellipses ($I_1$, $I_2$ and the generic space), the dashed lines represent the concepts that lead to the generic space (*GEN*) and the black lines illustrate the counterpart relations (*CPR*).

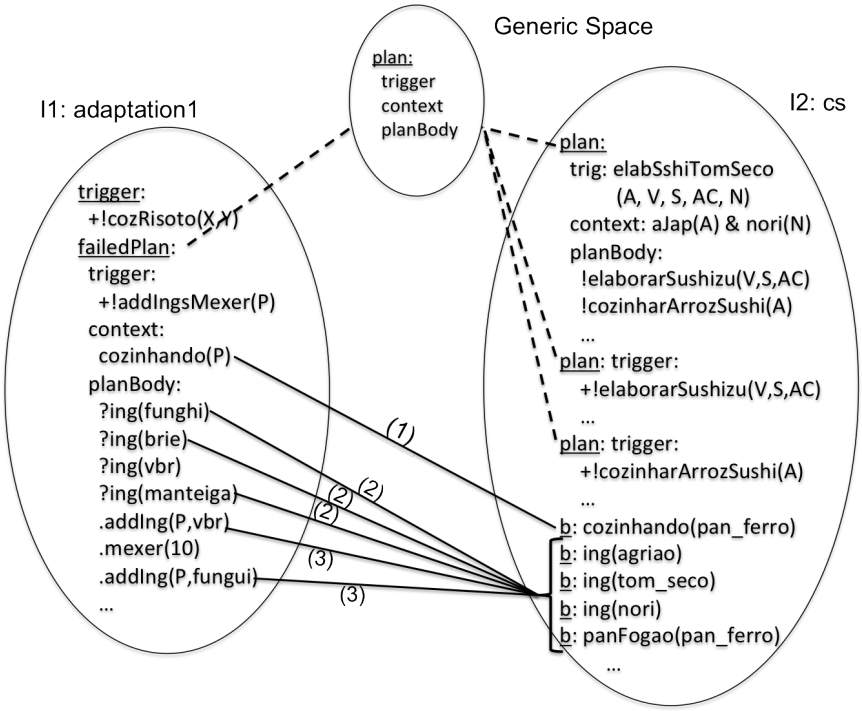


Figure 23 Initial configuration of the adaptation blend

Now that the necessary elements are ready, the selective projection takes place. In the case of double-scope blending, both the organizing schemes and the concepts are subject to change. Although modifying the organizing scheme is present in the double-scope rule, in this adaptation study it does not have an impact over the practical reasoning since only the agent performs it. In order for the modifications on the organizing schemes to impact the practical reasoning, it is necessary to model it with meta-level rules. Given a definition of the reasoning in meta-level rules, any modification on a meta-rule also modifies the outcome of the reasoning. Such specification of practical reasoning with meta-level rules and a possible generalization to other kinds of reasoning characterize a future work of this research.

The selective projection of concepts, denoted by $concepts = \alpha\left(C_1 \cup C_2 \cup C_{gen} \cup appCP\left(C_1 \cup C_2 \cup C_{gen}\right)\right)$, chooses concepts ($\alpha$) from the inputs, generic space, counterpart relations and from the application of the constitutive principles on the concepts. Given our current definition of $\alpha$ as a random selection function, any concept might be projected. Thus, it is as if we randomly chosen an answer from our imagination and then tested on the real world if the answer is correct. This is a subject where further researches from cognitive and neurological sciences needs to be incorporated in the blending theory in order to specify the selective projection. Figure 24 illustrates a possible outcome of applying double-scope blending to the inputs. In that figure the gray dashed lines depict the connection to generic space while the gray direct lines depict the associations. The black lines represent the concepts that were chosen by the selective projection and that constitute the blend. Inside the Figure 24, the blend is depicted by the ellipsis on the bottom.

In this specific case, the intention's trigger was projected allowing a possible plan to unify its trigger to the one of the intention (depicted by line marked with 1). A plan from the second input space (cozinharArrozSushi) is also projected to the blend and its

parameter is unified with the event that generated the intention (line 2). Besides that, three beliefs are projected to the blend (lines that follow the mark 3).

Given the resulting blend, the modified agent class (Appendix 1) adds a plan to handle failure of the intention or a plan to handle the event, in the case of lack of options. Then, the plan is added as defined in the blend – considering that a plan is defined. If there are beliefs in the blend, they are added to the library. After the modifications, the agent cycle continues normally, which corresponds to the elaboration phase of blending, where the blend is executed.

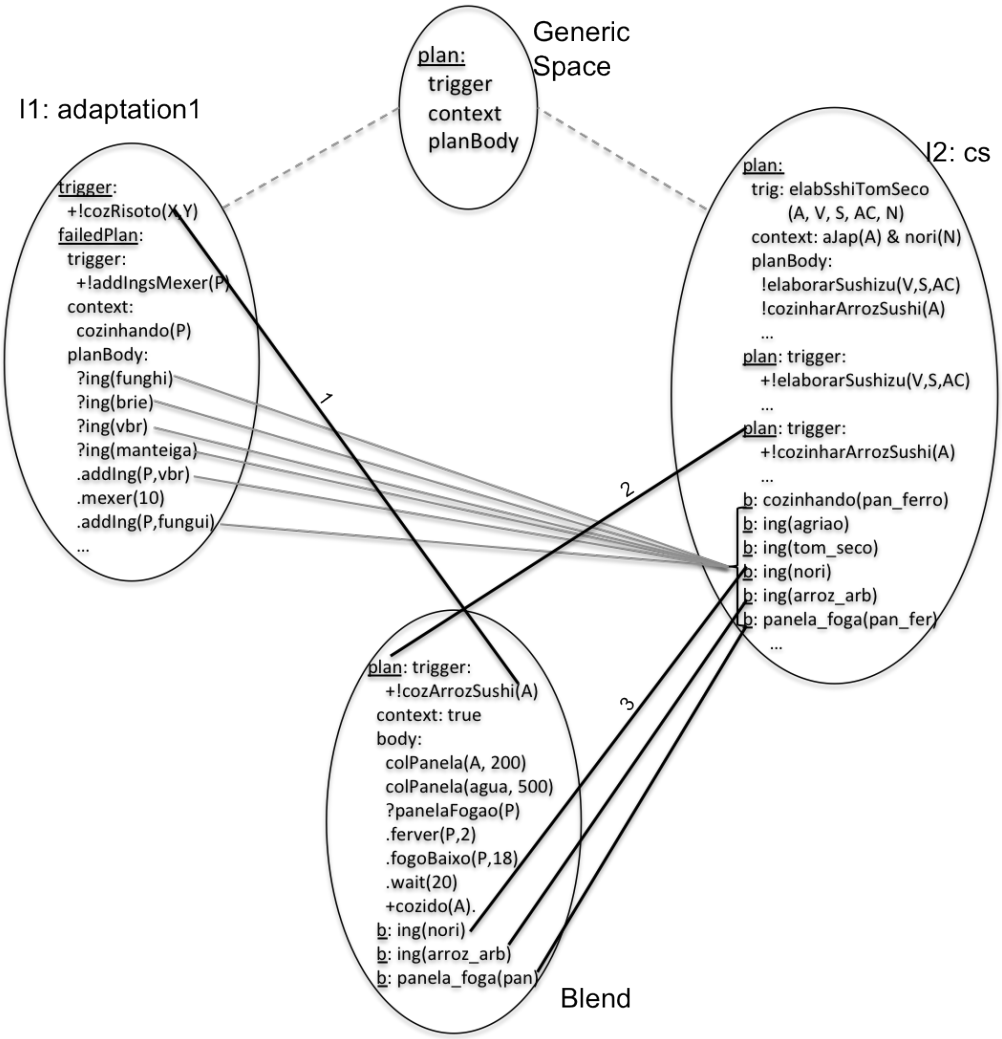


Figure 24 Double-scope blend for adaptation

In our adaptation study, the counterpart relations are established between beliefs, plans and beliefs and plans. Prioritizing the projection of concepts with relations avoid the direct copy of concepts unrelated to the situation. In a similar way, $S_{op}$ can be customized, for instance, to use more completion and suppress modification. Due to those modifications on the selection of concepts, the process tend to generate blends that make more sense – what Pereira (2007) characterize as a convergent strategy. Whether or not making sense is better for adaptation depends on the agent's domain and its context. As we previously stated, this kind of evaluation is not a part of the blending theory and should be modeled separately. Here, we describe how blending can be used as a way to generate alternatives for adaptation, without evaluating them. Figure 25

illustrates an adaptation blend where counterpart relations have more chance to be projected than non-related concepts. The notation is equal to the one adopted on Figure 24, where the selective projection is depicted by black lines. Here, the plan that failed was projected and, due to the relations between the beliefs and the test goals, they had more chance to be subject to the constitutive principles. Thus, one of them was modified and the other two were composed with different individuals of the same type that were available on the second space.

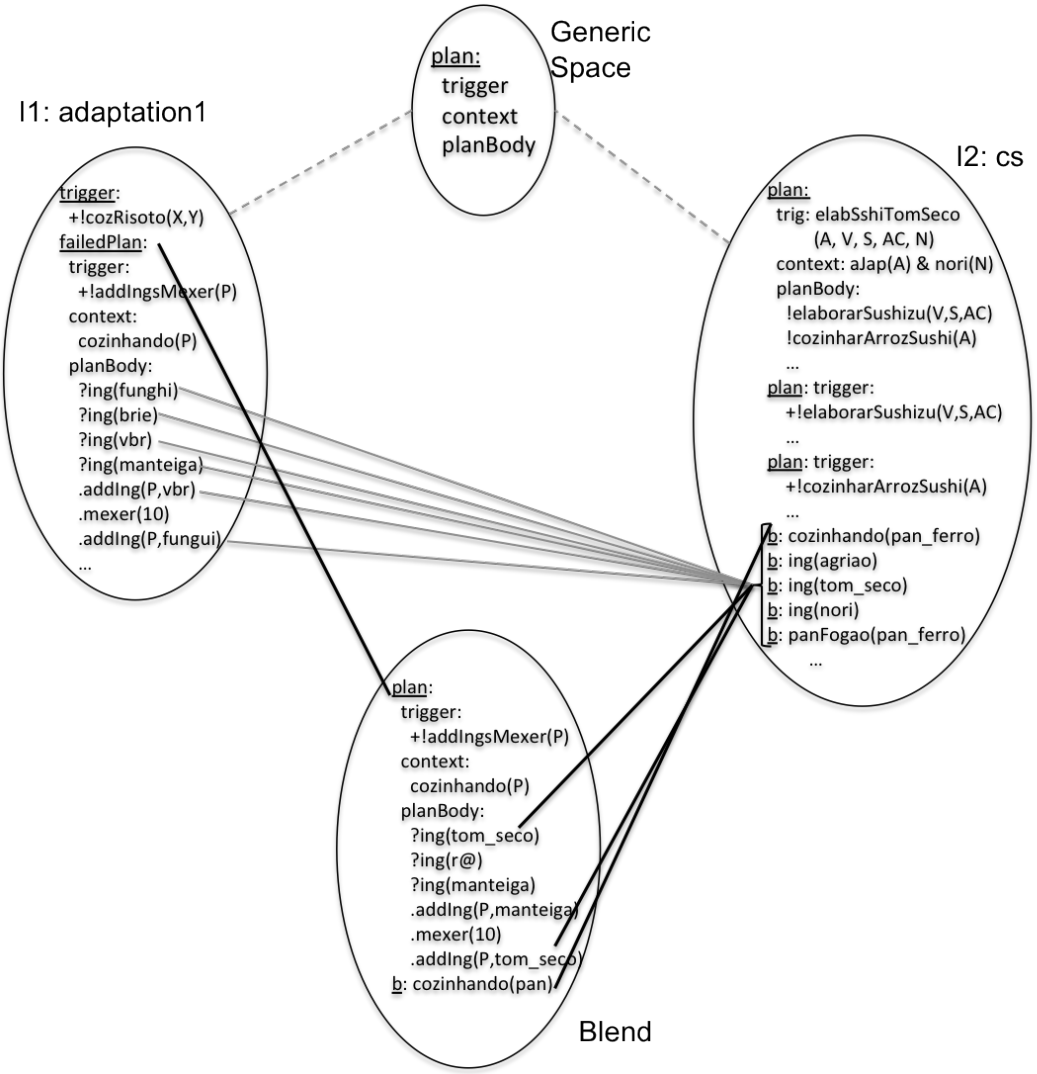


Figure 25 Adaptation blend with modified $\alpha$ and $S_{op}$

Our approach to model adaptation is in terms of the blending constructs defined in Section 3.1 (terminology and comparison and modification functions). The definition of the comparative functions is a fundamental aspect to produce blending. In this study, we defined simple functions that reflect a simple kind of adaptation (similarity and alternative plans given a common context and/or triggering event). Despite its simplicity, the model indicates several possibilities for enhancements and experimentation on adaptation strategies. Our intuition is that with the blending model, we are able to integrate different adaptation mechanisms in the same structure. There is also the possibility to experiment with different kinds of blend and selection functions ($\alpha$ and $S_{op}$).

## 3.3 Blend-based Recommendation

Our second study considers an educational recommender system designed as a BDI agent that uses blending to provide content-based recommendation (RESNICK; VARIAN, 1997; WEI; HUANG; FU, 2007). As presented in Sections 1.4.3 and 1.4.4, the purpose of this study is to test the blending model as a reasoning process. Thus, recommendation provides only an application context and, improving the state-of-the-art on recommender systems is not part of this research. Each user is modeled by a single recommender agent which, given a certain situation, will automatically recommend educational content.

We consider two main recommendation situations, one characterized by learning and the other by authoring. Here, we simplify the learning process and consider only recommendations of learning objects (CHURCHILL, 2007), given the current learning path being followed by the student. Considering authoring, the recommendation is directed for didactic materials developers, tutors and course teachers. Hence, authoring recommendations are constituted by parts of objects representing suggestions and examples about the specification of the metadata.

Since it is the agent that controls when an user will receive a recommendation, the blending mechanism was implemented as a Jason internal action (Appendix 12). Thus, blending may be executed inside any plan. The only requirement to use that internal action is a mapping between the beliefs and an OWL representation – since our implementation of blending adopts the OWL syntax.

Another aspect of the recommendation study is that we consider the existence of multiple agents, but they are unaware of each other. Hence, there is no interaction between the agents. In the context of recommender systems, agent interaction is more relevant when we consider other kinds of recommendation, such as collaborative filtering (RESNICK P. et al. 1994; SHARDANAND; MAES, 1995). Here, each agent manages a single user model that follows the terminology established by FOAF[2], IMS (Instructional Management Systems) LIP[3] and IMS-LD[4]. FOAF already provides an OWL representation compatible with our implementation. LIP and LD were modeled with the OWL language in the context of the OBAA research project.

Consequently, FOAF, LIP and LD establish a vocabulary of terms and properties that can be used to model learning activities and preferences of the user. Thus, the definition of the user model does not impose any restriction in terms of minimal information to construct an instance of it. This requirement depends on specific application contexts and, in the case of the learner recommendation, the minimal information is a description of the current learning activities. Moreover, in the authoring scenario the requirement is the existence of at least one metadata about the under development object. Those restrictions are modeled as the context of the respective recommendation plans.

Given the user-related ontologies, the agent environment supplies perceptions representing the user's activities according to the concepts defined by them.

---

[2] Friend of a Friend - http://www.foaf-project.org/

[3] Learner Information Package - http://www.imsglobal.org/profiles/

[4] Learning Design - http://www.imsglobal.org/learningdesign/

Consequently, the environment provides the interface between the Intelligent Tutoring System (ITS) and the recommender agent. Following traditional agent models, the actions are performed in the environment and, in this case, they reflect changes in the ITS, possibly being noted directly by the user through his interface.

After processing each perception, the respective beliefs are updated accordingly. Thus, the set of user-related beliefs actually represents the user model. Considering our blending model, those beliefs constitute the first input ($I_1$) of the process. As for the second input, the available learning objects constitute it. Figure 26 illustrates an example of $I_1$ and $I_2$ in the context of educational recommendation for students. Again, the conceptual spaces are depicted by the ellipses and each underlined element represents a concept. According to our implementation, each conceptual space is an OWL individual and the concepts are modeled as property assertions.

Therefore, the example of $I_1$ in Figure 26 considers an individual instead of the AgentSpeak representation. In practice, this conversion is performed directly by the internal action. Likewise, $I_2$ is also constructed by the action. Although it is recommended to use declarative knowledge during agent modeling, we left the information about learning objects hidden from the agent. We followed this approach since our recommendation model is still in its early stages and, for now, the agent would not use that information.

For instance, considering trust measures on repositories or previous evaluations of content developers imply on more elaborated decision procedures that justify the awareness of the resources by the agent. In fact, those decision aspects and the integration of more recommendation approaches constitute a subject for future research. Hence, in this study, the recommendation internal action gathers the available learning objects from the repository and assembles them into an OWL individual representing the second input – depicted by $I_2$ in Figure 26.

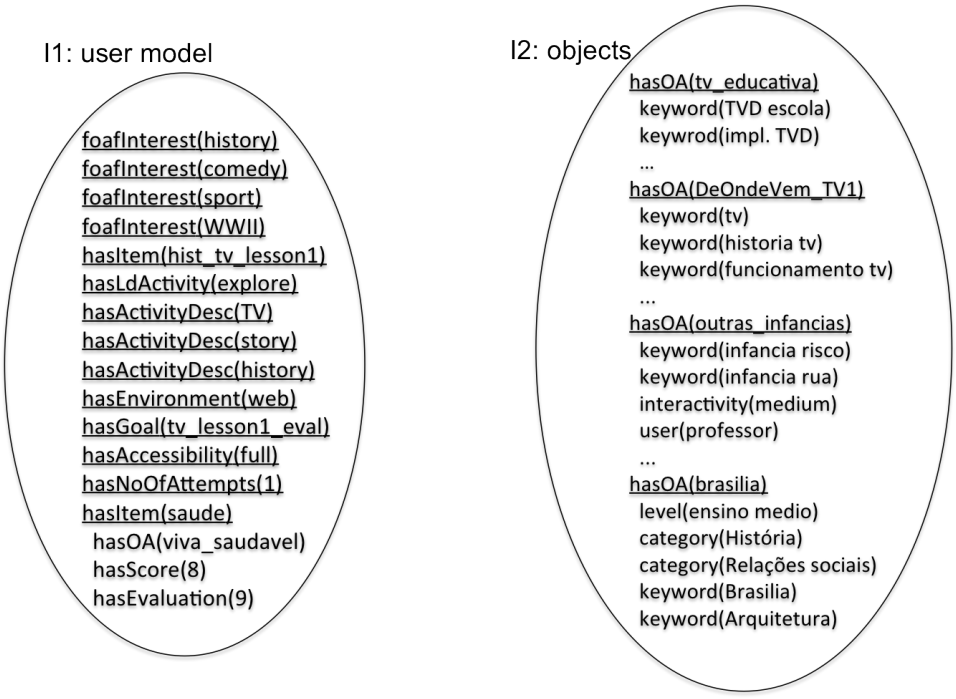


Figure 26 Inputs for the learning recommendation blending

Although constructed by the internal action, the second input is also based on a terminology for learning objects. Specifically we adopted the OBAA ontology[5] that represents a Brazilian standard for educational resources developed in the Informatics Institute and the Center for Interdisciplinary Research on Education from UFRGS (BEZ; VICARI; SILVA; RIBEIRO; GLUZ; PASSERINO; SANTOS; PRIMO; ROSSI; BORDIGNON; BEHAR; FILHO; ROESLER, 2010). Assuming the availability of objects, the second input space constitutes possible recommendations in terms of learning resources. In practice, it is necessary to define a filter to reduce the amount of candidate objects. Such filter can be applied outside of blending or it could be implemented in the $\alpha$ function plus a more restrictive comparative function. Our test considered only a prototype repository of learning objects that work on the Web, Digital TV and mobile devices. Thus, the amount of objects was not an issue.

Considering an authoring recommendation, the definition of the first input follows the same approach as described for the student recommendations. Since the user is a tutor or a course organizer developing an object, its model reflects this situation by containing only more generic information about the user (FOAF) and about the object being developed (i.e. a partial description of its metadata). In Figure 27 we illustrate an example of the inputs for the authoring recommendation. Similar to the learning context, in the authoring recommendation the available learning objects also constitute the second input. We follow the same graphical notation as the previous Figure.

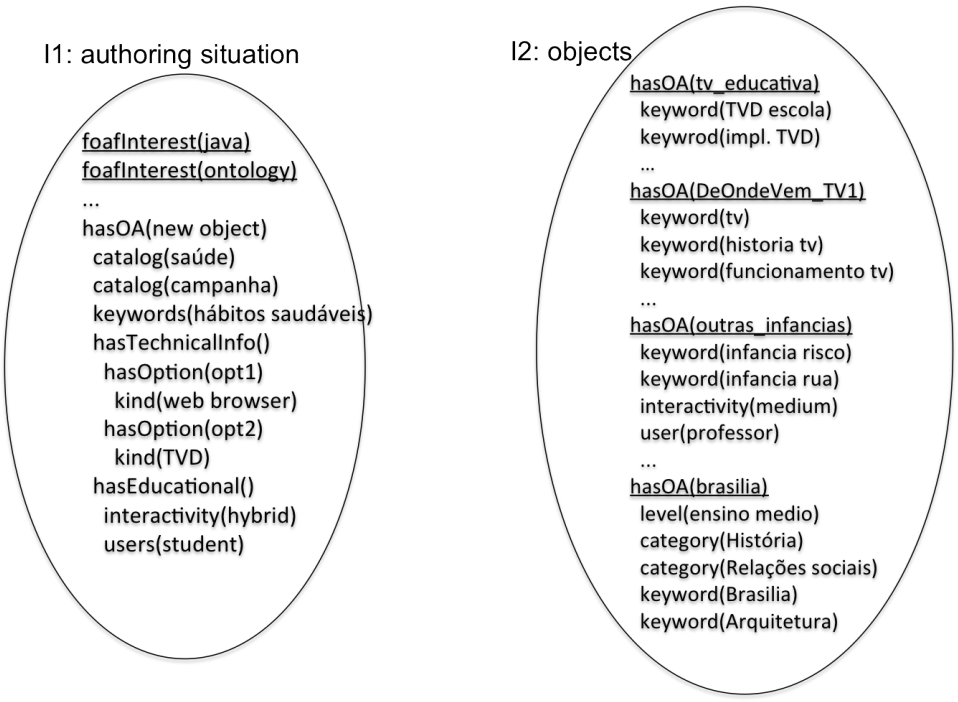


Figure 27 Inputs for the authoring recommendation blending

Another element of the blending process is the set of comparison functions. In the case of recommendation, these functions represent the similarity measures being considered. Here, we consider three kinds of similarity between an user model and a learning object model. Since both models are represented in OWL, it is possible to analyze their concepts in regard to their positions in the hierarchy. Thus, we check if

[5] http://www.portalobaa.org/obaac/padrao-obaa/concretizacao-de-metadados-em-owl

one class is a super or sub-class in relation to the other. The implementation of this function is presented in the Appendix 13. In Figure 28 we show examples of hierarchy relations according to an ontology that describes educational categories. Following this ontology, the individuals "ser humano e saúde", "tecnologia e sociedade", "terra e universo" and "vida e ambiente" are members of the class "ciências naturais" which is a branch of the ontology. Thus, all these individuals have at least one direct membership in common (depicted by the lines in the Figure 28).

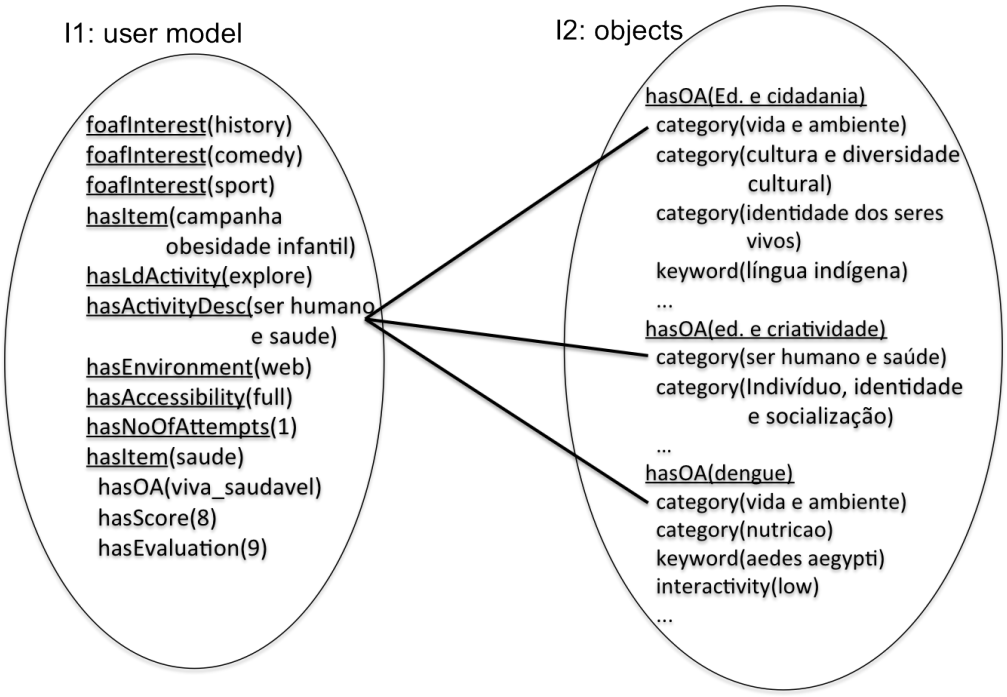


Figure 28 Hierarchy-based comparison function example

Disregarding the terminology, the literal content of related properties (e.g. activity description and category, from the user and learning object models respectively) is compared using a word similarity approach that uses Wikipedia as a corpus (PONZETTO; STRUBE, 2007). In the Figure 29 we show an example of applying this function on the properties foafinterest and hasActivityDescription from the user model and the properties keyword and category from the learning objects. Thus, when the similarity measure is above the defined threshold the words are regarded as similar (illustrated by the lines in the Figure 29). The implementation of the function (Appendix 14) is based on the Wikipedia Similarity API[6]. Formally, we specify that function as follows:

Word comparison function according to Wikipedia

$$\mu(w_1, w_2) = boolean\left(\text{extAPI} : wikiMeasure\left(w_1, w_2\right)\right)$$

where

$$boolean(real) = \begin{cases} true & \text{if } real > 0.8 \\ false & \text{otherwise} \end{cases}$$

[6] http://www.h-its.org/english/research/nlp/download/wikipediasimilarity.php

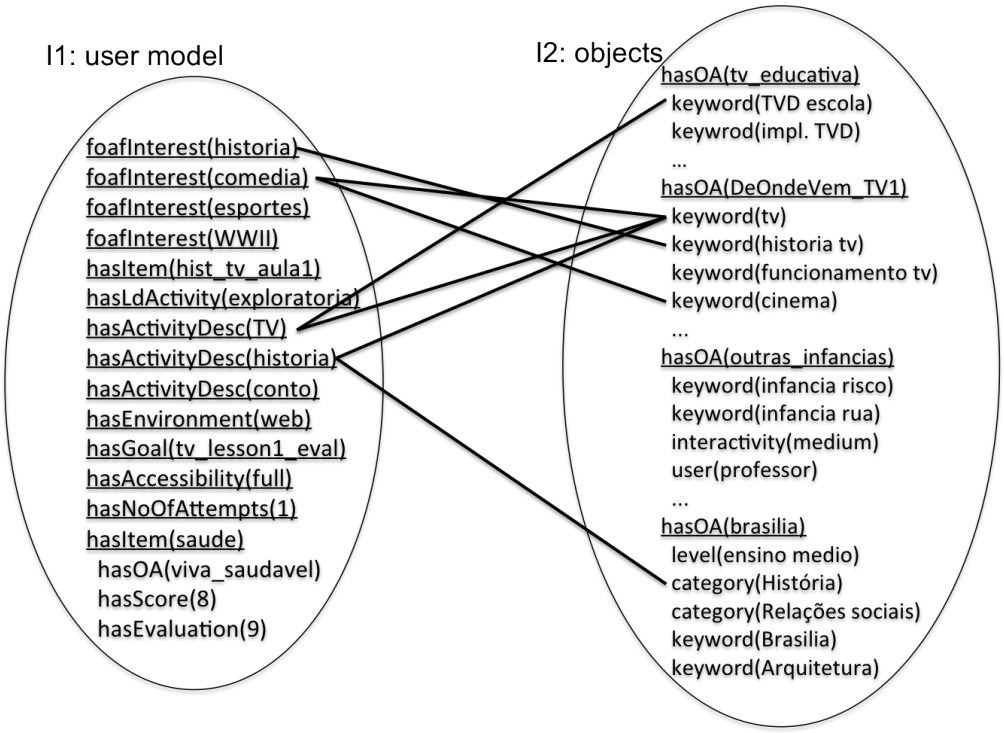


Figure 29 Word similarity function based on Wikipedia

Finally, also considering the literal content of the properties, we verify if one word is analogous to the other with regard to the ConceptNet database (LIU; SINGH, 2004). ConceptNet, essentially, is a semantic network representing common sense knowledge built and evaluated by volunteers through the Internet. Havasí, C. et al.. (2009) describe an analogy-based inference mechanism to virtually reduce the size of the network – making it more usable by humans.

The inference mechanism is distributed as an API but its results can also be accessed directly using ConceptNet's REST (Representational State Transfer) API. Therefore, the results of the inference establish an analogy space inside ConceptNet (SPEER; HAVASÍ; LIEBERMAN, 2008). Grounded on the analogy space, ConceptNet is able to compute similarity among concepts. Thus, we refer to that analogy space to compute analogies for the recommendation conceptual spaces. Our resulting analogy comparison function is very similar to the Wikipedia word comparison. Actually, the difference is on the way that similarity is computed and on the interpretation of the measure.

To the best of our knowledge, the integration of ConceptNet and the analogy space was developed and made available for the English corpus. Hence, the Portuguese version of ConceptNet does not use the analogy space. Since the learning objects from our repositories are in Portuguese, we were not able to test the comparison function directly with the objects.

Function to compute analogy between words according to ConceptNet

$$\mu(w_1, w_2) = boolean\left(\text{extAPI} : analogyMeasure\left(w_1, w_2\right)\right)$$

where

$$boolean(real) = \begin{cases} true & \text{if } real > 0.5 \\ false & \text{otherwise} \end{cases}$$

With the definition of $I_1$, $I_2$ and $FC$, the remaining blending elements to be specified are $BC$ and M – part of the OP component. The latter is a set of modification functions that represent the constitutive principle of modification. Considering the constitutive principles, we do not define any specific function for the learning recommendation. Although possible to model, we would need to consider different kinds of multimedia and its technical specificities in order to not corrupt the object. Besides, we are not confident that in the case of recommending educational resources, it is interesting to modify the objects since they represent complete and modular learning units. In our perspective, this specific recommendation benefits more from the integration of the comparative functions than on the constitutive principles.

However, in the context of authoring recommendation, the application of the constitutive principles yields on blends that can be used on practice. For instance, during authoring tasks, the agent may recommend suggestions on how to specify the metadata based on already developed objects ($I_2$). In this scenario of developing a new object, certain metadata from $I_2$ can be projected to the blend as a modification that reflects given properties of the under-development object. Assuming the availability of domain ontologies, that can be used to specify the object's properties, it is possible to use modification functions like the ones defined for the adaptation study.

Hence, we specify a modification function that, given an object property assertion – stating that a subject is related to a target object – it modifies the target object by choosing another individual from the same class or from the range definition. Thus, the property assertion remains the same except for the target. On the function, the definition of the axioms variable contains an intersection among domain, range and the suspension points that represent the remaining OWL property axioms, such as symmetrical and functional.

Modification function for authoring recommendation

$$\mu(assertion) = \begin{cases} modTarget = \alpha\left(c \in membersOf\left(range\right)\right) \text{ if } range \neq \{\} \\ modTarget = \alpha\left(c \in membersOf\left(class\left(targetInd\right)\right)\right) \text{ otherwise} \end{cases}$$

where

$assertion = property(subjectInd, targetInd)$

$property = name \cup axioms$

$axioms = domain \cup range \cup ...$

Concluding our specification of the recommendation blending, we define the configuration $BC = \langle \kappa, \alpha, \phi, S_{op} \rangle$. Together, $\alpha$ and $S_{op}$ represent the selective projection – $\alpha$ chooses which concepts will be project and $S_{op}$ which constitutive principles will be applied. Considering our recommendation context, we define three alpha functions that are chosen by the agent and applied on different situations. First, we specify a selection function following the same approach adopted in the adaptation study. Thus, this function improves the chance to project concepts that have counterparts in the other space.

Our second $\alpha$ function aims at choosing concepts that have more chance to surprise the user. Clearly, there are several approaches to achieve this kind of recommendation (WEI; HUANG; FU, 2007; SHARDANAND; MAES, 1995). Here, we look for objects

that belong to different domains in relation to the ones that the user is more used to. In addition, it also considers objects developed for different learning strategies. Inside a learning context, this kind of recommendation is useful when the current strategy and its adopted objects are not resulting in satisfactory learning outcomes (specified by the grades).

Finally, we developed an $\alpha$ function specific to prioritize objects that relate to at least one that the learner has used and evaluated well. Ideally, this function should be used with mirror blending, defined by the $\kappa$ component. According to the blending theory, mirror networks are capable of modeling metaphoric reasoning. By maintaining the roles (represented by the properties) of one input, and projecting the values from the other input would result in an "as if…" interpretation. Since, in this study, the inputs do not share the same organizing frame but have common roles (their objects), this alpha simulates that common aspect and improves the chances of projecting concepts of this role. Another aspect considered by this function is the specific properties from the objects, such as the learning strategy and utilization context.

Since mirror blending requires the same organizing schemes for both inputs, we applied this last $\alpha$ in conjunction with single-scope blending. In this type of blending one input serves as a source (providing the roles/properties) and the other as the target (supplying the values/concepts). Opposing mirror blending, in single-scope there might occur contradictions (called clashes in the theory) since the organizing schemes may differ. Currently, the organizing schemes do not have such impact in our model. We emphasize that the main reason for that is not considering meta-level reasoning rules inside the schemes.

Thus, in Figure 30 we show an example of applying the metaphor inspired $\alpha$ and single-scope blending (defined by $\kappa$) to gather recommendations for the student. Recalling that, in this specific recommendation, the constitutive principles are not applied, the role of blending is to choose which concepts (objects) to recommend, based on the inputs, generic space and counterpart relations. This role can be noted in Figure 30 by the direct projection of objects "dengue" and "educação e criatividade" since they relate to the category property of the already evaluated object ("viva saudável"). In the Figure 30 the gray lines illustrate the relations among concepts – lines with the number 1 refer to hierarchy relations and with 2 refer to word similarity – and the dotted black lines represent the projection of the concepts.

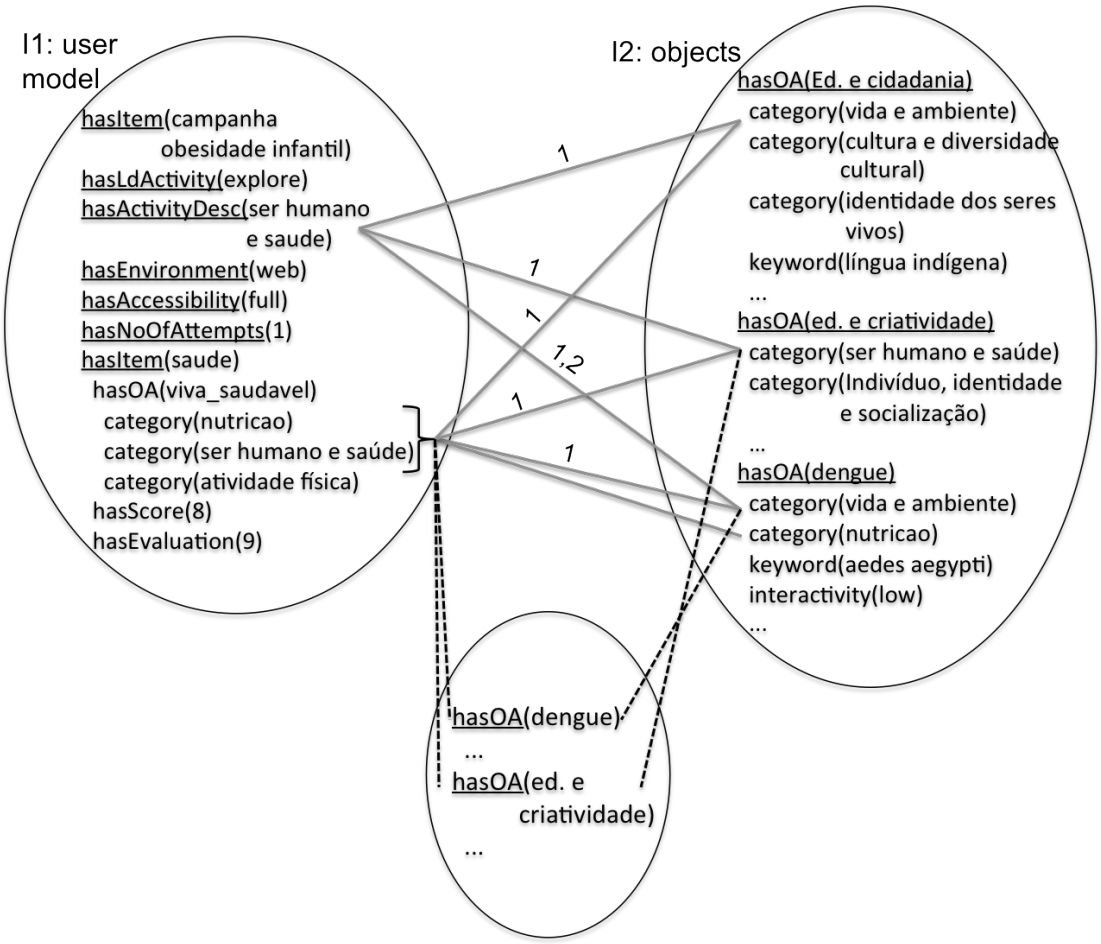


Figure 30 Metaphor-based recommendation blend

In the remaining situations we adopt double-scope blending allowing the projection of any concept from the considered spaces to the blend. Therefore, this kind of blend allows the projection of concepts from $I_1$ (the user model) to the blend (the recommendation). Although such projection is correct in regard to the theory, for this specific scenario concepts projected from $I_1$ are disregarded since they do not affect the recommendation.

Considering authoring, we use only the $\alpha$ function that projects more concepts with counterpart relations and double-scope as the kind of blending. Also present in this BC configuration for authoring recommendation is the $S_{op}$ function that chooses which constitutive principles to apply. In this case, we use the same $S_{op}$ function defined for the adaptation study, where each constitutive principle has the same probability to be chosen. Thus, in the Figure 31 we illustrate the a possible authoring blend, where the topmost dotted gray lines denote the definition of the generic space, the gray lines refer to the relations among concepts and the dashed lines and the black dashed lines pointing to the blend space refer to the selective projection.

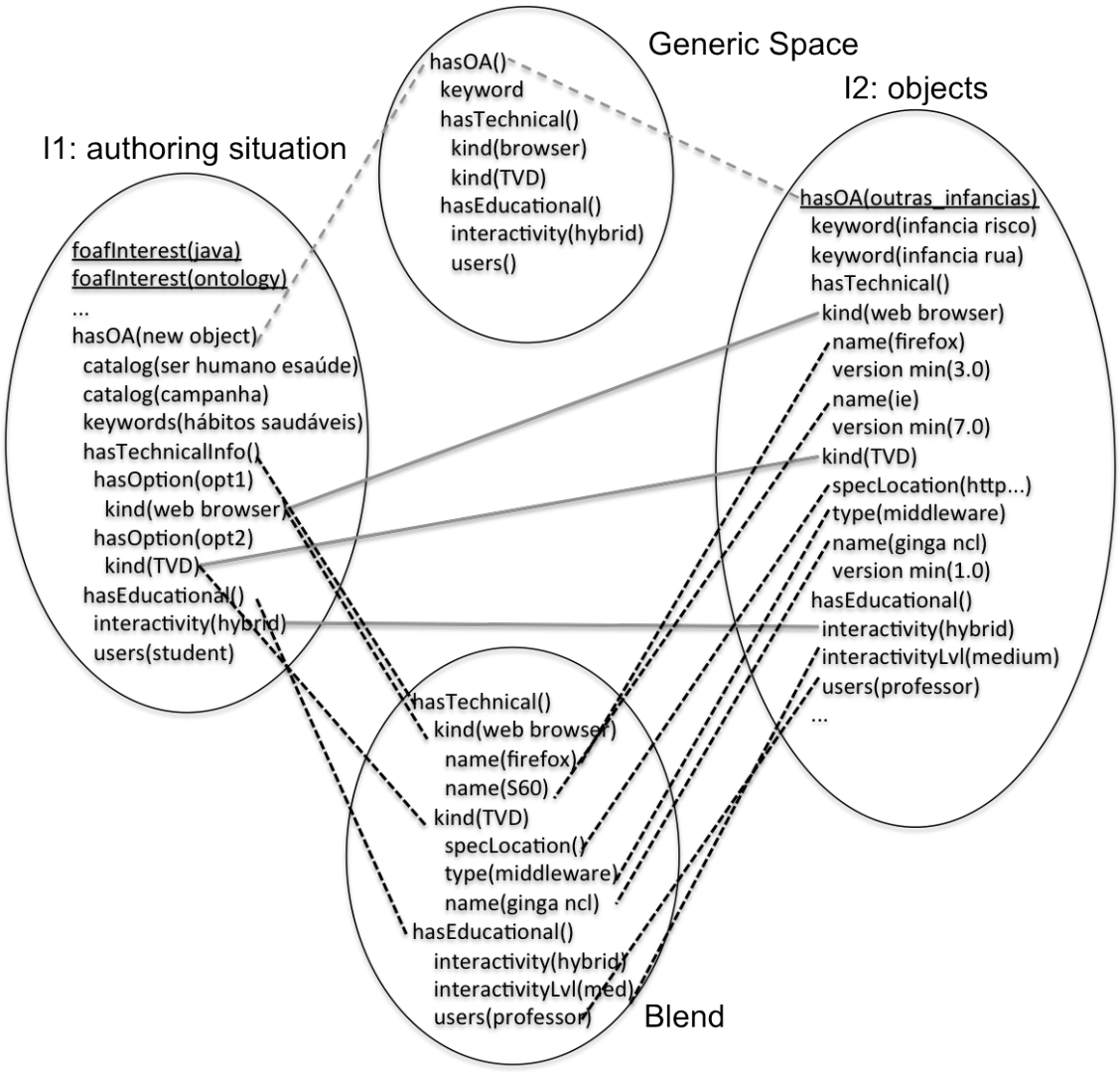


Figure 31 Authoring recommendation blend

In this study we showed how the restrictions and specificities of a given domain can be modeled as blending elements. Our model of student recommendation imposes restrictions to the blending mechanism, disabling the application of constitutive principles. Without these principles, blending is used to consider different representations and reasonings into a single model. Specifically in this recommendation study, blending is used to choose leaning objects given the applied similarity measures between the user model and the objects.

Following the blending theory, the resulting blends may serve as inputs for other blends, establishing a network of blends. The definition of rules to coordinate the chaining of blends and of the network utilization is defined by the governing principles, which were not modeled and are subject of future research. Given the model presented on Section 3.1, it is possible to specify blending in terms of a domain terminology and assertions and comparison and modification functions. Our applications on adaptation and recommendation indicates that the adoption of an agent structure complements well the blending model since it provides a way to automate the specification of inputs and the triggers of the blending mechanism.

Comparing to the model proposed by Pereira (2007), the main difference is on the integration to a BDI structure, allowing blending to be automatically applied (in terms of input construction and selection functions). Divago has the advantage to also model blending's governing principles. Although, we observe that governing principles is not his main focus of research, since the author does not indicate evidence from blending or other cognitive theories to argument his decision on the specification.

Still considering our related work, our approach to abstractly model the comparison and modification functions is similar with the morphisms specified by Griot's algebraic semiotics. In terms of expressiveness, Griot further defines the semantics of a modification in the context of symbolic representations. Directly comparing to our model, Griot is more expressive for manipulating symbols but is less expressive on the representation of non-symbolic representations.

# 4 CONCLUSION

Given our main research question: how can creativity support intentionality? We contextualize our contributions under two main perspectives (intelligent agency and computational creativity). Positioning our contributions on computational creativity we relate it to our first intermediary question: How can creativity be computationally modeled in order to produce theoretical and practical knowledge? Initially, the path taken to answer this question was to specify a blending model focused on practical knowledge (plans and actions). However, we noted that such path would also pass by theoretical knowledge and that, according to a higher abstraction level, all kinds of knowledge could be seen as a conceptualization. With this abstraction in mind, it is possible to specify the process of blending in a more generic way. Therefore, we specified blending as a set of rules and transitions among states representing the process of constructing a blend. Given the set of rules and constructs necessary for blending, we also defined four kinds of blend, as specified by theory's typology developed by Fauconnier and Turner (2002).

Regarding computational models of blending, one contribution of our model is an explicit definition of the original blending typology. Future work on this subject lead to the specification of other kinds of blends as proposed by Brandt (2002). Another contribution to computational creativity is a blending model capable of manipulating heterogeneous knowledge representations. This is possible due to the abstraction of a concept and the utilization of descriptive logics as a bridge between the blending operation and the knowledge representation.

Our utilization of descriptive logics follows the same approach as the utilization of the OWL language to annotate resources on the semantic web. Specifically, we follow the OWL syntax and semantics to describe the knowledge representations. Thus, our model deals with heterogeneous representation by following a given terminology and assertions about that representation. Further exploring this aspect of our model, it is possible to experiment it with multimedia representations. We can benefit from already developed ontologies and standards for multimedia on the semantic web and apply it to our model. In fact, enabling blending to work with multimedia allows us to further develop our second proof of concept, the educational recommender system. Therefore, the recommendations could also consider the content of the resources.

Pereira (2007), describes an experiment where the concept of a house has a symbolic representation and a geometric one for each of the house's component. Consequently, any symbolic blend with the house also has its graphical counterpart. Our model differs in the consideration that there is only one representation annotated with information that the blending process will use. Another difference is our utilization of OWL providing compatibility with documents from the semantic web. Defining our model abstractly – using operational semantics – allows developers to implement blending in any programming language. We developed a partial implementation of the rules in Java, using also the OWL-API (HORRIDGE; BECHHOFER, 2009) to manipulate the OWL

syntax, Pellet (SIRIN, E. et al.. 2007) to reason over the OWL representations and Jason as the agent framework.

Recalling our motivation on the interplay between autonomy and creativity, we define agent-based creativity as a subject of study of this thesis. On this matter, our main idea was to use the BDI constructs to model the vital relation of intentionality inside the blending model. However, intentionality is a property that results from the application of practical reasoning over a set of beliefs and desires. Therefore, a simplistic approach would be to map intentional relations between input spaces when, for instance, one belief from input one is a desire in the other. Although it could be possible to specify it in our model as comparison function, we concluded that it would not impact on the blending process itself since we are considering a very specific case of intentionality that does not correspond to broad view of intentionality as a vital relation. Thus, an additional study of intentionality – possibly modeled with meta-level components – is left for future work.

Adding meta-level reasoning to our model allows different reasoning rules to be used inside the blending process. One way to use it would be on the definition of the counterpart relations, where we could infer cause and effect chains among concepts. Another interesting feature is the possibility to modify the reasoning rules and verify the behavior of a conceptual space given such modifications (specially useful for double-scope blending).

Following our agent-based creativity perspective, an important point that can be modeled using agent constructs is the definition of the inputs, context, and purpose of the blending. These elements are considered as given by the blending theory (there is no definition on how the inputs are constructed). Hence, according to an agent perspective, its current intentions and world configuration are applied to trigger the blending process. Actually, the composition of the input spaces by the agent characterizes a contribution for computational models of blending since none of the related works specified how the inputs were constructed. The inputs are always given by the developer who defines all the parameters for the blending. In the case of our study on adaptation, the blending is triggered when an intention fails or when there is no available option. In addition, the inputs are automatically constructed given a pre-defined terminology on adaptation. Furthermore, our study on recommendation also work in a similar way, the only difference is on the blending trigger, which is declared in the agent code as an achievement goal.

Shifting the perspective to creative agency, the main result of this work is the possibility to integrate different reasonings, strategies and representations in a single blend. During the development of the agent adaptation study, we realized that the strength of our model lies on the integration, on the blending itself, rather than on an adaptation measure. Considering our blending model, adaptation is defined by a terminology and functions for comparing and modifying belief and plans. In our study, we defined comparison functions representing very simple adaptation strategies. However, the model can also be applied to model other adaptation methods, like abduction and/or hierarchical planning.

Thus, these methods are modeled as additional comparison functions *fc* that will compose the *FC* set, used during blending. Moreover, the developer may specify the random selection function $\alpha$ to consider a domain specific tendency or any heuristic. In fact, that function can also be integrated to a learning mechanism. Similarly, the

function to select which constitutive principle to apply $S_{op}$, may also be customized. Comparing to our related work on agent adaptation, our approach differs on the reasoning mechanism. Currently, we are not able to provide any analysis in terms of which adaptation method is better given a certain situation. On the field of adaptation, we consider that our contribution is on a model to integrate different adaptation mechanisms into a single one. Consequently, future work on blended adaptation may specify case studies to analyze the integration of strategies against the adoption of a single one.

Given our two main perspectives, agent-based creativity and creative agency we realize that creativity and agency may also be viewed as parts of a greater cognitive model. Many of the elements from blending that are too abstract or too subjective to model computationally, could be at least partially modeled given the availability of other cognitive functions like learning and embodiment. With these two additional operations, the blending mechanism may also function as a supplier of actions to be performed and perceived on several levels (e.g. internal and external). As the agent begins to try actions and perceive its consequences the learning mechanism may infer new cause and effect chains predicting effects of actions not yet tested. Therefore, blending could be seen as playing the role of our imagination. Despite challenging, we believe that this work initiated an approach to the integration of different cognitive functions into a single model. Perhaps even towards a restricted concretization of Minksy's society of the mind (MINSKY, 1986).

Finally, returning to our main research question, we position our contribution under a more theoretical perspective. Our model can be used to study aspects of CB theory that have not been thoroughly studied. We see this work as the initial step towards a deeper study on the cognition of creativity using computational models. Furthermore, we see future works on theoretical aspects by the adoption of results from genetic and neurosciences. Considering more technical aspects, given the concept model defined here, the main future work is the specification of meta-reasoning improving the expressivity of the organizing schemes. Therefore, the elaboration phase of the blending process can also account the modifications on the reasoning rules. Still on the blending model, another continuity of the research is the specification of the network model – as envisioned by Fauconnier and Turner (1998) – which provides the foundation for the definition of the governing principles.

Given a specification of the network model and of meta-level reasoning rules, it is possible to experiment with integration of learning. Thus, we add a cognitive operation that can provide evaluations and feedback over the generated conceptualizations. Another aspect that can be modeled similarly is the social impact of the new artifacts. Hence, we move on the direction of integrating H-creativity (BODEN, 2004) to conceptual blending. The specification of Brandt's typology (BRANDT, 2002) also adds more contextual information to the blending operation. Together with the network model and other cognitive operations, a further specified typology allows the creation more fine-tuned conceptualizations.

# REFERENCES


BAADER, F.; HORROCKS, I.; SATTLER, U. Description Logics. In: VAN HARMELEN, F.; LIFSCHITZ, V.; PORTER, B. (Ed.). **Handbook of Knowledge Representation**. [S.l.]: Elsevier, 2007. p. 135-180.

BEZ, M.; VICARI, R. M.; SILVA, J. M. C. da; RIBEIRO, A.; GUZ, J. C.; PASSERINO, L.; SANTOS, E. R.; PRIMO, T.; ROSSI, L.; BORDIGNON, A.; BEHAR, P.; FILHO, R.; ROESLER, V. Proposta Brasileira de Metadados para Objetos de Aprendizagem Baseados em Agentes (OBAA). **Renote**, [S.l.], v.8, n.2, 2010.

BINSTED, K. et al. Computational Humor. **IEEE Intelligent Systems**, Piscataway, NJ, USA, v.21, n.2, p.59–69, 2006.

BLAIN, P. J. **A Computer Model of Creativity Based on Perceptual Activity Theory**. 2007. Tese (Doutorado em Ciência da Computação) – Griffith University.

BODEN, M. A. Creativity and artificial intelligence. **Artificial Intelligence**, [S.l.], v.103, n.1-2, p.347 – 356, 1998. Artificial Intelligence 40 years later.

BODEN, M. A. **The Creative Mind**: myths and mechanisms. 2.ed. New York: Routledge, 2004.

BORDINI, R. H.; WOOLDRIDGE, M.; HüBNER, J. F. **Programming Multi-Agent Systems in AgentSpeak using Jason** (Wiley Series in Agent Technology). [S.l.]: John Wiley & Sons, 2007.

BRANDT, L. **Conceptual Integration Typology (4.0)**. 2002. Disponível em: <http://www.hum.au.dk/semiotics/docs2/pdf/brandt_line/conceptual_integration.pdf>. Acesso em junho de 2010.

BRANDT, L. **Explosive Blends**: from cognitive semantics to literary analisys. 2000. Tese (Doutorado em Ciência da Computação) – Roskilde University.

BRATMAN, M. E.; ISRAEL, D. J.; POLLACK, M. E. Plans and resource-bounded practical reasoning. **Computational Intelligence**, [S.l.], v.4, p.349–355, 1988.

BRATMAN, M. **Intentions, Plans, and Practical Reason**. Harvard: Harvard UniversityPress, 1987.

BRENNER, M. Multiagent planning with partially ordered temporal plans. In: IJCAI'03, 2003, San Francisco, CA, USA. **Anais. . .** Morgan Kaufmann Publishers Inc., 2003. p.1513–1514.

BRENNER, M.; NEBEL, B. Continual planning and acting in dynamic multiagent environments. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.19, n.3, p.297–331, 12 2009.

BROERSEN, J. et al. The BOID architecture: conflicts between beliefs, obligations, intentions and desires. In: AGENTS '01: Fifth International Conference On Autonomous Agents, 2001, New York, NY, USA. **Anais. . .** ACM, 2001. p.9– 16.

CARDOSO, A. et al.. An architecture for hybrid creative reasoning. In: PAL, S. K; DILLON, T. S.; YEUNG, D. S. **Soft computing in case based reasoning**. London: Springer-Verlag, 2001. p.147–177.

CARROLL, J. M.; KELLOGG, W. A. Artifact as theory-nexus: hermeneutics meets theory-based design. **SIGCHI Bull.**, New York, NY, USA, v.20, n.SI, p.7–14, 1989.

CHOLVY, L.; GARION, C. Desires, Norms and Constraints**.** In: AAMAS '04, 2004, Washington, DC, USA. **Anais. . .** IEEE Computer Society, 2004. p.724–731.

CHURCHILL, D. Towards a useful classification of learning objects. **Educational Technology Research and Development**, [S.l.], v.55, n.5, p.479–497, 2007.

CREGAN, A. Symbol Grounding for the Semantic Web**.** In: ESWC, 2007. **Anais. . .** Berlin:Springer-Verlag, v.4519, p.429-442, 2007. (Lecture Notes in Computer Science).

COLMERAUER, A. Prolog in 10 figures. **Commun**. ACM, New York, NY, USA, v.28, n.12, p.1296–1310, 1985.

COLTON, S. Creativity versus the perception of creativity in computational systems. In: AAAI Spring Symp. On Creative Intelligent Systems, 2008. **Anais. . .** [S.l.: s.n.], 2008. p. 14-20.

COLTON, S.; MÁNTARAS, R. L. de; STOCK, O. Computational Creativity: coming of age. **AI Magazine**, [S.l.], v.30, p.11–14, 2009.

CONTE, R.; ANDRIGHETTO, G.; CAMPENNÍ, M. The Immergence of Norms in Agent Worlds. In: ESAW, 2009. **Anais. . .** [S.l.: s.n.], 2009. p.1–14.

CSIKSZENTMIHALYI, M. Society, Culture, and Person: a systems view of creativity. In: STERNBERG, R. J. **The Nature of Creativity**. [S.l.]: Cambridge University Press, 1988. p.325–339.

D'INVERNO, M. et al. The dMARS Architecture: a specification of the distributed multi-agent reasoning system. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.9, n.1-2, p.5–53, 2004.

DASTANI, M. et al.. A Programming Language for Cognitive Agents Goal Directed 3APL. In: PROMAS, 2003. **Anais. . .** [S.l.: s.n.], 2003. p.111–130.

DASTANI, M.; TINNEMEIER, N.; MEYER, J.-J. C. A Programming Language for Normative Multi-Agent Systems. In: DIGNUM, V. (Ed.). **Handbook of Research on Multi-Agent Systems**: semantics and dynamics of organizational models. [S.l.]: Information Science Reference, 2009.

DORIN, A.; KORB, K. A New Definition of Creativity. **Artificial Life: Borrowing from Biology**, [S.l.], p.11–21, 2009.

DUCH, W.; PILICHOWSKI, M. Experiments with computational creativity. **Neural Information Processing - Letters and Reviews**, [S.l.], v.11, n.4-6, p.123–133, 2007.

FAUCONNIER, G. How Compression Gives Rise to Metaphor and Metonymy. In: 9th Conference On Conceptual Structure, Discourse, And Language. **Anais. . .** [S.l.: s.n.], 2008.

FAUCONNIER, G.; TURNER, M. Conceptual Integration Networks. **Cognitive Science**, [S.l.], v.22, n.2, p.133–187, 1998.

FAUCONNIER, G.; TURNER, M. **The Origin of Language as a Product of the Evolution of Modern Cognition**. [S.l.]: Equinox Publishing, 2008.

FAUCONNIER, G.; TURNER, M. **The Way We Think**: conceptual blending and the mind's hidden complexities. [S.l.]: Basic Books, 2002.

FIKES, R. E.; NILSSON, N. J. STRIPS: a new approach to the application of theorem proving to problem solving. In: ALLEN, J.; HENDLER, J.; TATE, A. (Ed.). **Readings in Planning**. San Mateo, CA: Kaufmann, 1990. p.88–97.

FRANOVÁ, M.; KODRATOFF, Y. On Computational Creativity, 'Inventing' Theorem Proofs. In: ISMIS, 2009. **Anais. . .** [S.l.: s.n.], 2009. p.573–581.

FUJITA, M. Intelligence Dynamics: a concept and preliminary experiments for open-ended learning agents. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.19, n.3, p.248–271, 12 2009.

GANGEMI, A. Norms and plans as unification criteria for social collectives. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.17, n.1, p.70–112, 2008.

GEORGEFF, M. P.; LANSKY, A. L. Reactive Reasoning and Planning. In: AAAI, 1987. **Anais. . .** [S.l.: s.n.], 1987. p.677–682.

GOGUEN, J. A.; HARRELL, D. F. Style as a Choice of Blending Principles. In: ARGAMON, S.; DUBNOV, S.; JUPP, J. Style and Meaning in Language, Art, Music, and Design: Papers from the AAAI Fall Symposium. **Anais...** [S.l.: s.n.], 2004.

GOGUEN, J. An Introduction to Algebraic Semiotics, with Application to User In terface Design. **Computation for Metaphors, Analogy, and Agents**, [S.l.], p.242–291, 1999.

GOVERNATORI, G.; ROTOLO, A. BIO logical agents: norms, beliefs, intentions in defeasible logic. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.17, n.1, p.36– 69, 2008.

HARMELEN, F. van; LIFSCHITZ, V.; PORTER, B. (Ed.). **Handbook of Knowledge Representation (Foundations of Artificial Intelligence)**. [S.l.]: Elsevier Science, 2007.

HARNAD, S. The Symbol Grounding Problem. **Physica D**: Nonlinear Phenomena, [S.l.], v. 42, p.335-346, 1990.

HAVASI, C. et al. Digital Intuition: applying common sense using dimensionality reduction. **Intelligent Systems, IEEE**, [S.l.], v.24, n.4, p.24–35, 2009.

HÉLIE, S.; SUN, R. Knowledge integration in creative problem solving. In: Annual Meeting Of The Cognitive Science Society, 30., 2008. **Anais. . .** [S.l.: s.n.], 2008.

HERVÁS, R. et al. Enrichment of Automatically Generated Texts Using Metaphor. In: MICAI, 2007. **Anais. . .** [S.l.: s.n.], 2007. p.944–954.

HOFWEBER, T. Logic and Ontology. In: ZALTA, E. N. (Ed.). **The Stanford Encyclopedia of Philosophy**. Stanford: [s.n.], 2009.

HORRIDGE, M.; BECHHOFER, S. The OWL API: a java api for working with owl 2 ontologies. In: OWLED, 2009. **Anais. . .** CEUR-WS.org:259, 2009.

HUBER, M. J. **JAM**: a bdi-theoretic mobile agent architecture. In: The Third International Conference On Autonomous Agents (AGENTS'99), 1999. **Anais. . .** ACM Press, 1999. p.236–243.

IMAZ, M.; BENYON, D. **Designing with Blends**: conceptual foundations of human-computer interaction and software engineering. [S.l.]: The MIT Press, 2007.

JIANG, H.; VIDAL, J. M.; HUHNS, M. N. EBDI: an architecture for emotional agents. In: AAMAS, 2007. **Anais. . .** [S.l.: s.n.], 2007. p.11.

KASANEN, F.; KARI, L.; ARTO, S. The constructive approach in management accounting research. **Journal of Management Accounting Research**, [S.l.], v.5, p.243–259, 1993.

KLAPISCAK, T.; BORDINI, R. H. **JASDL**: a practical programming approach combining agent and semantic web technologies. In: Declarative Agent Languages And Technologies VI, 2008. **Anais. . .** [S.l.: s.n.]. v.5397.

KOWALSKI, R. **Logic for problem-solving**. The Netherlands: North-Holland Publishing, 1986.

KUMAR, D.; SHAPIRO, S. C. The OK BDI Architecture. **International Journal of Artificial Intelligence Tools**, [S.l.], v.3, n.3, p.349–366, March 1994.

LEITE, J.; SOARES, L. Adding Evolving Abilities to a Multi-Agent System. In: CLIMA VII, 2006. **Anais...** Springer, 2006. p.246–265. (Lecture Notes in Computer Science, v.4371).

LIU, H.; SINGH, P. ConceptNet - A Practical Commonsense Reasoning Tool-Kit. **BT Technology Journal,** Hingham, MA, USA, v.22, n.4, p.211–226, 2004.

LUKKA, K. The constructive research approach. In: Case Study Research In Logistics, 2003. **Anais. . .** [S.l.: s.n.], 2003. v.B1, p.83–101.

MÁNTARAS**,** R. L. Towards artificial creativity : examples of some applications of ai to music performance. In: FERNÁNDEZ-CABALLERO, A. et al. (Eds.). **50 Años de la Inteligencia Artificial**. [S.l.]: UCLM, 2006. p.43–49.

MARCH, S. T.; SMITH, G. F. Design and natural science research on information technology. **Decision Support Systems**, [S.l.], v.15, n.4, p.251 – 266, 1995.

MARTINS, J. M.; MIRANDA, E. R. A Connectionist Architecture for the Evolution of Rhythms. In: EVOWORKSHOPS, 2006. **Anais. . .** Springer, 2006. p.696–706. (Lecture Notes in Computer Science, v.3907).

MARTINS, J. M. et al. Enhancing Sound Design with Conceptual Blending of Sound Descriptors. In: Workshop on Computational Creativity (CC'04) - European Conference on Case-Based Reasoning (ECCBR), 2004, Madrid. **Anais. . .** Madrid: Universidad Complutense de Madrid, 2004. p.243-255.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. **Bulletin of Mathematical Biophysics**, [S.l.], n. 5, p.115-133, 1943.

MENEGUZZI, F. R.; LUCK, M. Leveraging New Plans in AgentSpeak(PL). In: DALT, 2008. **Anais. . .** [S.l.: s.n.], 2008. p.111–127.

MENEGUZZI, F.; LUCK, M. **Norm-based behaviour modification in BDI agents**. In: AAMAS'09, 2009, Richland, SC-USA. **Anais...** International Foundation for Autonomous Agents and Multiagent Systems, 2009. p.177–184.

MINSKY, M. **A Framework for Representing Knowledge**. Relatório Técnico, MIT-AI Laboratory, AIM-306. Cambridge, MA, USA, 1974.

MINSKY, M. **The society of mind**. New York, NY, USA: Simon & Schuster, Inc., 1986.

MORAES, M. C. **Agentes Improvisacionais como Agentes Deliberativos**. 2004. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

MORLEY, D.; MYERS, K. The SPARK Agent Framework. In: AAMAS '04, 2004, Washington, DC, USA. **Anais. . .** IEEE Computer Society, 2004. p.714–721.

MOTIK, B. et al.. Modeling ontologies using OWL, Description Graphs, and Rules. In: OWLED 2008. **Anais. . .** [S.l.: s.n.].

MURPHY, G. L. **The Big Book of Concepts**. [S.l.]: The MIT Press, 2002.

NILSSON, N. J. The Physical Symbol System Hypothesis: Status and Prospects. In: LUNGARELLA, M. et. al. (Org.). **50 Years of Artificial Intelligence**. Berlin:Springer-Verlag, v.4850, p.9-17, 2007.

PEARCE, M.; MÜLLENSIEFEN, D.; WIGGINS, G. A. A Comparison of Statisti- cal and Rule-Based Models of Melodic Segmentation. In: ISMIR, 2008. **Anais...** [S.l.: s.n.], 2008. p.89–94.

PEARL, J. Bayesian networks: a model of self-activated memory for evidential reasoning. In: Conference Of The Cognitive Science Society, California, Irvine, 1985. **Proceedings. . .** California: [s.n.], 1985. p.329–334.

PEREIRA, F. C. **Creativity and AI**: a conceptual blending approach. [S.l.]: Mouton de Gruyter, Berlin, 2007. (Applications of Cognitive Linguistics (ACL)).

PEREIRA, F. C.; CARDOSO, A. Experiments with free concept generation in Divago. **Knowl.-Based Syst.**, [S.l.], v.19, n.7, p.459–470, 2006.

PILICHOWSKI, M.; DUCH, W. Neurocognitive Approach to Creativity in the Domain of Word-Invention. In: ICONIP (2), 2008. **Anais. . .** [S.l.: s.n.], 2008. p.88–96.

PLOTKIN, G. D. **A Structural Approach to Operational Semantics**. [S.l.]: Computer Science Department, Aarhus University, 1981. (DAIMI FN-19).

PLOTKIN, G. D. A structural approach to operational semantics. **Journal of Logic and Algebraic Programming**, [S.l.], v.60-61, p.17–139, 2004.

PONZETTO, S. P.; STRUBE, M. Knowledge Derived from Wikipedia for Computing Semantic Relatedness. **Journal of Artificial Intelligence Research**, [S.l.], v.30, p.181–212, 2007.

QUILLIAN, M. Semantic Memory. In: MINSKY, M. (Ed.). **Semantic Information Processing**. [S.l.]: MIT Press, 1968. p.227–270.

RAO, A. S. AgentSpeak(L): bdi agents speak out in a logical computable language. In: MAAMAW '96, 1996, Secaucus, NJ, USA. **Anais...** Springer-Verlag New York:[S.n.], 1996. p.42–55.

RAO, A. S.; GEORGEFF, M. P. Modeling Rational Agents within a BDI-Architecture. In: KR, 1991. **Anais. . .** [S.l.: s.n.], 1991. p.473–484.

RESNICK, P. et al. GroupLens: an open architecture for collaborative filtering of netnews. In: ACM Conference on Computer Supported Cooperative Work, 1994, Chapel Hill, North Carolina. **Anais. . .** ACM, 1994. p.175–186.

RESNICK, P.; VARIAN, H. R. Recommender systems. **Commun. Communications of the ACM**, [S.l.], v.40, n.3, p.56–58, 1997.

RITCHIE, G. Some Empirical Criteria for Attributing Creativity to a Computer Program. **Minds Mach**., Hingham, MA, USA, v.17, n.1, p.67–99, 2007.

SAUNDERS, M.; THORNHILL, A.; LEWIS, P. **Research Methods for Business Students**. 4.ed. [S.l.]: FT Prentice Hall, 2006.

SAUNDERS, R. Supporting Creativity Using Curious Agents. In: COMPUTATIONAL CREATIVITY SUPPORT - WORKSHOP CCS 2009, 2009. **Proceedings...** [S.l.: s.n.], 2009.

SAUNDERS, R. Towards a Computational Model of Creative Societies Using Curi- ous Design Agents. In: ESAW, 2006. **Anais. . .** [S.l.: s.n.], 2006. p.340–353.

SAUNDERS, R.; GERO, J. S. How to study artificial creativity. In: Creativity & Cognition, 2002. **Anais. . .** [S.l.: s.n.], 2002. p.80–87.

SEN, S.; AIRIAU, S. Emergence of Norms through Social Learning. In: IJCAI, 2007. **Anais. . .** [S.l.: s.n.], 2007. p.1507–1512.

SHARDANAND, U.; MAES, P. Social Information Filtering: algorithms for automating “word of mouth”. In: ACM CHI’95 Conference On Human Factors In Computing Systems, 1995. **Proceedings. . .** [S.l.: s.n.], 1995. v.1, p.210–217.

SHOHAM, Y.; POWERS, R.; GRENAGER, T. If multi-agent learning is the answer, what is the question? **Artificial Intelligence**, [S.l.], v.171, n.7, p.365 – 377, 2007. (Foundations of Multi-Agent Learning).

SIRIN, E. et al. Pellet: a practical owl-dl reasoner. **J. Web Sem**., [S.l.], v.5, n.2, p.51–53, 2007.

SPEER, R.; HAVASI, C.; LIEBERMAN, H. AnalogySpace: reducing the dimensionality of common-sense knowledge. In: AAAI, 2008. **Anais. .** . [S.l.: s.n.], 2008. p.548–553.

STEUNEBRINK, B. R.; DASTANI, M.; MEYER, J.-J. C. A Logic of Emotions for Intelligent Agents. In: AAAI, 2007. **Anais. . .** [S.l.: s.n.], 2007. p.142–147.

STONE, P. Learning and Multiagent Reasoning for Autonomous Agents. In: IJCAI, 2007. **Anais. . .** [S.l.: s.n.], 2007. p.12–30.

SUBAGDJA, B.; SONENBERG, L.; RAHWAN, I. Intentional learning agent architecture. **Autonomous Agents and Multi-Agent Systems**, [S.l.], v.18, n.3, p.417–470, 06 2009.

VAISHNAVI, V.; KUECHLER, W. 2004. **Design Research in Information Systems**. Available at: <http://desrist.org/design-research-in-information-systems> Accessed on december, 2009.

VEALE, T.; HAO, Y. A Fluid Knowledge Representation for Understanding and Generating Creative Metaphors. In: COLING 2008, 2008. **Anais. . .** [S.l.: s.n.], 2008.

WEI, K.; HUANG, J.; FU, S. **A Survey of E-Commerce Recommender Systems**. In: International Conference on Service Systems and Service Management, 2007. **Anais. . .** [S.l.], p.1– 5, 2007.

WIGGINS, G. A. A preliminary framework for description, analysis and comparison of creative systems**. Knowledge-Based Systems**, [S.l.], v.19, n.7, p., 2006.

WILLIAMS, M.-A. Representation = Grounded Information**.** In: PRICAI, 2008. **Anais. . .** [S.l.], v.5331, p.437–484, 2007. (Lecture Notes in Computer Science).

WOOLDRIDGE, M. **Reasoning about Rational Agents**. [S.l.]: The MIT Press, 2000.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: theory and practice. **Knowledge Engineering Review**, [S.l.], v.10, p.115–152, 1995.

ZHU, J.; HARRELL, D. F. Daydreaming with Intention: scalable blending-based imagining and agency in generative interactive narrative. In: AAAI Spring Symp. On Creative Intelligent Systems, 2008. **Anais. . .** [S.l.: s.n.], 2008. p. 156-163.

# APPENDIX 1 <AGENT MODEL WITH INTEGRATED ADAPTATION MECHANISM>

```
package agent;

import java.util.*;

import org.semanticweb.owlapi.model.OWLOntology;

import reason.Blender;
import reason.IterationPhi;
import reason.Modification;
import reason.OWLBlender;
import reason.RBeliefModfication;
import reason.RandomAlpha;
import reason.RandomSop;
import reason.SEBModification;

import edu.uci.ics.jung.graph.Hypergraph;
import edu.uci.ics.jung.graph.SetHypergraph;


import asSemantics.CBEventsGoalListener;
import asSemantics.CBTransitionSystem;

import jason.JasonException;
import jason.RevisionFailedException;
import jason.architecture.AgArch;
import jason.asSemantics.ActionExec;
import jason.asSemantics.Agent;
import jason.asSemantics.Circumstance;
import jason.asSemantics.Event;
import jason.asSemantics.IntendedMeans;
import jason.asSemantics.Intention;
import jason.asSemantics.TransitionSystem;
import jason.asSemantics.Unifier;
import jason.asSemantics.GoalListener.GoalStates;
import jason.asSyntax.ASSyntax;
import jason.asSyntax.Atom;
import jason.asSyntax.Literal;
import jason.asSyntax.LiteralImpl;
import jason.asSyntax.LogicalFormula;
import jason.asSyntax.Plan;
import jason.asSyntax.PlanLibrary;
import jason.asSyntax.Pred;
import jason.asSyntax.StringTermImpl;
import jason.asSyntax.Trigger;
import jason.asSyntax.Trigger.TEOperator;
import jason.asSyntax.Trigger.TEType;
import jason.asSyntax.parser.ParseException;
import jason.bb.BeliefBase;
import jason.runtime.Settings;
import kr.*;

public class CBAgent extends Agent {

    //Mind mind;
    HyperMind mind;
```

```
	private RelatorFactory rf;

	public CBAgent() {
		// TODO Auto-generated constructor stub
		super();
		rf = new RelatorFactory();
		//mind = new Mind();
		mind = new HyperMind();
	}

	public void initAg() {
		super.initAg();
	}

	public void initAg(String asSrc) throws JasonException {
		super.initAg(asSrc);
		//inicializar a imaginacao do agente com base nas crencas e etc, agora que sabemos
		//que o agente foi iniciado corretamente.
		CBTransitionSystem cbt = new CBTransitionSystem(this, ts.getC(), ts.getSettings(),
ts.getUserAgArch());
		ts = cbt;
		CBEventsGoalListener cbe = new CBEventsGoalListener(this);
		cbt.addGoalListener(cbe);
		bbToImagination();
		plToImagination();
		//mind.printGraph();
		mind.showImagination();
	}

	Hypergraph<Concept, HyperConceptRelation> adaptationInput1(Trigger goal){
		Hypergraph<Concept, HyperConceptRelation> ret = new
			SetHypergraph<Concept, HyperConceptRelation>();
		//O trigger eh representado no grafo como um conceito, tendo em vista que nao
necessariamente ele
		//reflete uma crenca. Esse eh conceito eh assoaciado a um Conceito Plano como
"triggers"
		Concept trigger = new Concept(goal, true);

		Intention currentIntention = getTS().getC().getSelectedIntention();
		IntendedMeans im = currentIntention.getIM(goal, new Unifier());
		//plano que falhou
		Concept failPlan = new Concept(im.getPlan(), true);
		ret.addVertex(failPlan);

		/*Concept intention = new Concept(currentIntention, true);
		ret.addVertex(intention);*/
		//im.getCurrentStep();

		for (Literal b : getBB()){
			ret.addVertex(new Concept(b, true));
		}

		return ret;
	}

	private Set<Modification> agentMods(){
		SEBModification seb = new SEBModification(this);
		RBeliefModfication rb = new RBeliefModfication(0.2f);
		Set<Modification> ret = new HashSet<Modification>();
		ret.add(seb);
		ret.add(rb);
		return ret;
	}

	private Set<SimpleConceptRelator> agentFcs(){
		PlanTriggerEnabler pc = new PlanTriggerEnabler();
		BeliefStringSimilarity bc = new BeliefStringSimilarity();
		Set<SimpleConceptRelator> ret = new HashSet<SimpleConceptRelator>();
		return ret;
	}

	public void intentionFailure(Trigger goal){
```

```
                Intention currentIntention = getTS().getC().getSelectedIntention();
                IntendedMeans im = currentIntention.getIM(goal, new Unifier());
                //plano que falhou
                Plan fp = im.getPlan();

                OWLHelper o = new OWLHelper();
                OWLOntology i1 = o.adaptationInput1(goal, fp, bbToSet(getBB()));
                OWLOntology i2 = o.adaptationInput2( plToSet(getPL()), bbToSet(getBB()));
                //FALTAM AS FUNCOES DE COMPARACAO E MODIFICACAO
                Set<Modification> mods = agentMods();
                Set<SimpleConceptRelator> fcs = agentFcs();
                OWLBlender blender = new OWLBlender(i1, i2, mods, fcs, Blender.K.DOUBLE_SCOPE, new
RandomAlpha(), new IterationPhi(10), new RandomSop());
                OWLOntology ret = blender.blend();
                AgtBlendResult agresult = o.blendToAgent(ret);

                System.out.println("IMPRIMINDO AGT RESULT");
                Plan teste = null;
                for (Plan p : agresult.getPlans()){
                        System.out.println("P: " + p.getFunctor() + " " + p.getBody());
                        teste = p;
                }
                for (Literal l : agresult.getBeliefs()){
                        System.out.println("B: " + l.toString());
                        try {
                                brf(l, null, currentIntention);
                        } catch (RevisionFailedException e) {
                                e.printStackTrace();
                        }
                }
                System.out.println(" ADICIONANDO PLANO p/ META-EVENT");

                if (teste!=null){
                        System.out.println("CRIANDO FAIL LITERAL");
                        //Literal                failLiteral                =                    new
LiteralImpl(fp.getTrigger().getLiteral().getFunctor());
                        Literal failLiteral = new LiteralImpl(fp.getTrigger().getLiteral());

                        try {
                                failLiteral.addAnnot((ASSyntax.parseTerm("state(failed)")));
                        } catch (ParseException e1) {
                                e1.printStackTrace();
                                //gerar o evento de falha e seguir com o ciclo
                        }
                        System.out.println("CRIANDO FAIL TRIGGER");
                        Trigger   failstate=   new   Trigger(TEOperator.goalState,   TEType.achieve,
failLiteral);
                        //FAILSTATE=^!g1(verde)[state(finished)]
                        Plan adaptPlan;
                        try {
                                System.out.println("literal:  "+  failLiteral  +  "  failstate:  "  +
failstate);
                                adaptPlan    =    new    Plan(null,    failstate,    Literal.LTrue,
teste.getBody());
                                System.out.println("adapt paln: " + adaptPlan);
                                System.out.println("ADICIONADNO PLANO");
                                getPL().add(adaptPlan);
                                Trigger  failevent  =  new  Trigger(TEOperator.add,  TEType.achieve,
goal.getLiteral());

                                Trigger fail2 = new Trigger(TEOperator.add, TEType.achieve, goal);

                                generateGoalStateEvent(goal.getLiteral(),           goal.getType(),
GoalStates.failed, null);
                        }catch (JasonException e) {
                                e.printStackTrace();
                                //gerar evento de falha mesmo assim, ignorando que o plano nao foi
adicionado
                        }
                }
        }

        //copy from jason1.3.3 library
```

```
        private void generateGoalStateEvent(Literal goal, TEType type, GoalStates state, String
reason) {
            goal = goal.forceFullLiteralImpl().copy();
            Literal stateAnnot = ASSyntax.createLiteral("state", new Atom(state.toString()));
            if (reason != null)
                stateAnnot.addAnnot(              ASSyntax.createStructure("reason",              new
StringTermImpl(reason)));
            goal.addAnnot( stateAnnot );
            Trigger eEnd = new Trigger(TEOperator.goalState, type, goal);
            if (ts.getAg().getPL().hasCandidatePlan(eEnd)) {
                 System.out.println("evento sendo gerado");
                ts.getC().addEvent(new Event(eEnd, null));
            }
        }

        private Set<Plan> plToSet(PlanLibrary pl){
                HashSet<Plan> ret = new HashSet<Plan>();
                for (Iterator<Plan> iterator = pl.iterator(); iterator.hasNext();) {
                        Plan p = iterator.next();
                        ret.add(p);
                }
                return ret;
        }

        private Set<Literal> bbToSet(BeliefBase bb){
                HashSet<Literal> ret = new HashSet<Literal>();
                for (Iterator<Literal> iterator = bb.iterator(); iterator.hasNext();) {
                        Literal literal = iterator.next();
                        ret.add(literal);
                }
                return ret;
        }

        public void intentionFailureTESTE(Trigger goal){
                System.out.println("^^^^^^^^^^^^^^^failed goal^^^^^^^^^^^ : L: " + goal.getLiteral()
+ " O: "
                                +  goal.getOperator()  +"  T:  "  +  goal.getType()  +  "  E:  "  +
goal.getErrorMsg());
                System.out.println("^^^^^^^^ terms do failed goal: " + goal.getTerms() + " ");
                //System.out.println("+++++ acao atual em TS: " + ts.getC().getAction().toString());
                System.out.println("+++++ SI: id: " + ts.getC().getSelectedIntention().getId() + "
term: " +
                                ts.getC().getSelectedIntention().getAsTerm() + " " );
                //ts.getC().get
                System.out.println("+++++ SO: " + ts.getC().getSelectedOption());

                //lista de planos capazes de lidar com o trigger que falhou
                List<Plan> lp = getPL().getCandidatePlans(goal);

                Intention currentIntention = getTS().getC().getSelectedIntention();
                System.out.println("CI To string: "+ currentIntention.toString());
                System.out.println("-----");
                System.out.println(getTS().getC().toString());
                IntendedMeans im = currentIntention.getIM(goal, new Unifier());
                //nome do plano que falhou
                im.getPlan();
                im.getCurrentStep();

                //System.out.println("+++++       SI.       hasTrigger       goal:       "       +
currentIntention.hasTrigger(goal, new Unifier()));
                System.out.println("+++++ SI.IM.getPlan: " + im.getPlan().toString() + " cstep: " +
im.getCurrentStep().toString());

                //falhou, vamos mostrar isso no grafo, jah linkado com os respectivos conceitos na
mente do agnte
                Collection<Concept> cc = new HashSet<Concept>();
                cc.add(new Concept(im.getPlan(), true));

                //trigger da intencao que falhou
                cc.add(new Concept(goal, true));

                //planos aplicaveis dif plano q falhou.
                for (Plan p : lp){
```

```
                        if (!p.equals(im.getPlan()))
                                cc.add(new Concept(p, true));
                }
        }

        private void circumstanceHandle(Circumstance c){
                //intencao escolhida, focar o blend para o que estah sendo realizado
                Intention si = c.getSelectedIntention();
                //si.isFinished();
                //si.isSuspended();
                //si.getAsTerm();
                Stack<IntendedMeans> sim = si.getIMs();
                HashSet<Plan> plans = new HashSet<Plan>();
                HashSet<Trigger> triggers = new HashSet<Trigger>();
                for (IntendedMeans im : sim){
                        //im.getCurrentStep();
                        plans.add( im.getPlan() );
                        triggers.add( im.getTrigger() );
                }
                c.getApplicablePlans(); //if null, no avaliable options... blend away baby, blend
away
        }

        /**Nesse metodo posso acrescentar a natureza dinamica do agente √† rede.
         * Ou seja, aqui, a Circumstance atual pode ser inserida na rede.
         * A alternativa √© nao incluir explicitamente essa parte na rede, porem
         * utiliza-la durante a fusao apenas.
         * Intencao como uma hyperAresta
         *
         */
        public void reasoningCycleEnd(){

                Intention si = getTS().getC().getSelectedIntention();
                if (si != null){
                si.isFinished();
                si.isSuspended();
                si.getAsTerm();

                System.out.println("@@@ SI: " + si.getAsTerm() + " is fini: " + si.isFinished() +
                                " is susp: " + si.isSuspended());
                }
                /*
                Stack<IntendedMeans> sim = si.getIMs();
                for (IntendedMeans im : sim){
                        im.getCurrentStep();
                        im.getPlan();
                        im.getTrigger();
                }

                ActionExec ac = getTS().getC().getAction();
                ac.getActionTerm();
                ac.getIntention();
                ac.getResult();

                Queue<Event> qe = getTS().getC().getEvents();
                for(Event e : qe){
                        e.getIntention();
                        e.getTrigger();
                        e.isExternal();
                        e.isInternal();
                }

                getTS().getC().getApplicablePlans();

                getTS().getC().getPendingActions();

                getTS().getC().getPendingIntentions();

                getTS().getC().getRelevantPlans();*/


        }
```

```
	private void addLiteralToImagination(Literal bel){
		Concept f = new Concept(bel, true);
		if (mind.getNetwork().containsVertex(f)){
			//setar o estado para true
			mind.updateConceptStatus(f, true);
			//System.out.println("*****************ADD BEL SET STATUS "  + bel);
		}else{
			//adicionar e verificar as relacoes :-)
			mind.addConcept(f);
			everyRelation(bel);
			//System.out.println("*****************ADD BEL FULL " + bel);
		}
	}

	private void delLiteralFromImagination(Literal bel){
		Concept f = new Concept(bel, false);
		//modificou a mente, entao atualizar a imaginacao
		mind.updateConceptStatus(f, false);
		//System.out.println("*****************DEL BEL " + bel);
	}

	@SuppressWarnings("unchecked")
	@Override
	public List<Literal>[] brf(Literal beliefToAdd, Literal beliefToDel,  Intention i) throws
RevisionFailedException {
	List<Literal>[] temp = super.brf(beliefToAdd, beliefToDel, i);
	if (temp != null){
		List<Literal> ll = temp[0];//additions to the belief base
		if (ll != null){
			for (Literal l : ll){
				addLiteralToImagination(l);
			}
		}
		ll = temp[1];//additions to the belief base
		if (ll != null){
			for (Literal l : ll){
				delLiteralFromImagination(l);
			}
		}
	}
	return temp;
	}

	private void everyRelation(Literal l){
		beliefToImagination(l);//crencas x crencas
		for (Plan p : getPL()){
			planRelations(l, p); //crencas x planos : geral: trigger
			contextRelations(l, p); //crencas x contexto de planos : string
		}
	}

	void beliefToImagination(Literal literal){
		Set<SimpleConceptRelator>	crs	=	rf.compatibleSimpleRelator(Literal.class,
Literal.class);
		Iterator<Literal> i2 = getBB().iterator();
		while (i2.hasNext()) {
			Literal literal2 = (Literal) i2.next();
			Iterator<SimpleConceptRelator> ci = crs.iterator();
			while (ci.hasNext()) {
				SimpleConceptRelator conceptRelator = ci.next();
				System.out.println("--> " + conceptRelator.getClass().toString());
				//ConceptRelator<Literal,	Literal,	?>	notElegantCast	=
conceptRelator;
				try {
					//ConceptRelation crl = new ConceptRelation();
					HyperConceptRelation crl = new HyperConceptRelation();
					crl.setResult( conceptRelator.relateO(literal, literal2));
					crl.setLabel("b");
					crl.setSimpleRelator(conceptRelator);
					if (crl.getResult() != null)
					{
						System.out.println(mind.addRelation(new
Concept(literal, true), new Concept(literal2, true), crl));
```

```
                                        }
                                } catch (NotRelationalException e) {
                                        e.printStackTrace();
                                }
                        }
                }
        }

        private void bbToImagination(){
                for (Literal literal : getBB())
                        beliefToImagination(literal);
        }

        private void contextRelations(Literal l, Plan p){
                StringContains cf = new StringContains();
                try {
                        if (p.getContext() != null){
                                String    teste    =    (String)    cf.relateO(    l.getFunctor(),
p.getContext().toString());
                                if (teste != null){
                                        Concept c = new Concept(p, true);
                                        c.setGraphLabel(p.getTrigger().toString());
                                        HyperConceptRelation crl = new HyperConceptRelation();
                                        crl.setResult( teste );
                                        crl.setLabel("pc");
                                        crl.setSimpleRelator(cf);
                                        System.out.println(mind.addRelation(new Concept(l, true),
c, crl));
                                }
                        }
                } catch (NotRelationalException e) {
                        e.printStackTrace();
                }
        }

        private void planRelations(Literal l, Plan p){
                Set<SimpleConceptRelator>     scr     =     rf.compatibleSimpleRelator(Literal.class,
Plan.class);
                if (p.getSrcInfo().getSrcFile() == this.getASLSrc()){
                        Concept c = new Concept(p, true);
                        c.setGraphLabel(p.getTrigger().toString());
                        mind.addConcept(c);
                        for (SimpleConceptRelator sc : scr){
                                try {
                                        HyperConceptRelation crl = new HyperConceptRelation();
                                        crl.setResult( sc.relateO(l, p));
                                        crl.setLabel("p");
                                        crl.setSimpleRelator(sc);
                                        if (crl.getResult() != null && (Boolean)crl.getResult())
                                        {
                                                System.out.println(mind.addRelation(new
Concept(l, true), c, crl));
                                        }
                                } catch (NotRelationalException e) {
                                        e.printStackTrace();
                                }
                        }
                }
        }

        private void plToImagination(){
                for (Plan p : pl){
                        for(Literal l : bb){
                                planRelations(l, p);
                                contextRelations(l,p);
                        }
                }
        }


    }
```

# APPENDIX 2 <EVENT LISTENER FOR CB AGENT>

```
package asSemantics;

import agent.CBAgent;
import jason.asSemantics.Event;
import jason.asSemantics.GoalListener;
import jason.asSemantics.TransitionSystem;
import jason.asSyntax.Trigger;

public class CBEventsGoalListener implements GoalListener {

    private TransitionSystem ts;
    private CBAgent ag;

    public CBEventsGoalListener(CBAgent ag){
            this.ag = ag;
            this.ts = ag.getTS();
    }

    public void goalFailed(Trigger goal) {
            ag.intentionFailure(goal);
    }

    public void goalFinished(Trigger goal) {
            // TODO Auto-generated method stub

    }

    public void goalResumed(Trigger goal) {
            // TODO Auto-generated method stub

    }

    public void goalStarted(Event goal) {
            // TODO Auto-generated method stub

    }

    public void goalSuspended(Trigger goal, String reason) {
            // TODO Auto-generated method stub

    }

}
```

# APPENDIX 3 <EXAMPLE AGENT>

```
// Agent Chef in project JCBAgent

/* Initial beliefs and rules */
ing(funghi).
ing(porcini).
ing(brie).
ing(parmesao).
ing(padano).
ing(vbr).
ing(vtinto).
ing(vinag_arroz).
ing(vinag_maca).
ing(manteiga).
ing(nori).
ing(tomate_seco).
ing(arroz_canoli).
ing(arroz_arb).
ing(arroz_jap_curto).
ing(arroz_jap_longo).
ing(rucula).
ing(agriao).
ing(alface).
ing(acucar).
ing(sal).

panelaFogao(pan_ferro).

/* Initial goals */

!start.

/* Plans */

+!start : true <-
     +casa(laranja);
     +casa(verde);
     !g1(yada).

+!elaborarSushiTomSeco(A, V, S, AC, N): arrozJap(A) & nori(N) <-
     !elaborarSushizu(V,S,AC);
     !cozinharArrozSushi(A);
     ?tomateSeco(T);
     ?queijoCremoso(Q);
     ?rucula(R);
     ?cozido(A);
     ?suhizu(SZ);
     !prepararArrozSuhi(A,SZ);
     ?arrozSPronto(AS);
     .montarRolo(N,AS);
     .print("fim sushi").

+!elaborarSushizu(V,S,AC): vinagreArroz(V) & sal(S) & acucar(AC) <-
     .colPanela(V, 100);
     .colPanela(S, 20);
     .colPanela(AC, 20);
     ?panelaFogao(P);
     .mexer(P);
     .ferver(P);
```

```
    +sushizu(szArroz).

+!cozinharArrozSushi(A) : true <-
    .colPanela(A, 200);
    .colPanela(agua, 500);
    ?panelaFogao(P);
    .ferver(P,2);
    .fogoBaixo(P,18);
    .wait(20);
    +cozido(A).

+!prepararArrozSuhi(A,SZ) : cozido(A) & sushizu(SZ) <-
    .colApoio(A);
    .espalhar(apoio, SZ);
    .ventilar(apoio);
    .mexer(apoio);
    +arrozPronto(apoio).

+!cozinharRisoto(X, Y) : arroz(X) & caldo(Y) <-
    .colPanela(X);
    .colPanela(Y);
    ?panelaFogao(P);
    .ligarFogoMedio(P);
    .adicionarCaldo(P, Y, 50);
    +cozinhando(P);
    .wait(1);
    !adicionarIngredienteseMexer(P);
    .print("fim cozinhar rizo").

+!adicionarIngredientesMexer(P): cozinhando(P) <-
    ?ing(funghi);
    ?ing(brie);
    ?ing(vbr);
    ?ing(manteiga);
    .adicionaIng(P, vbr);
    .mexer(10);
    .adicionaIng(P, fungui);
    .mexer(2);
    .adicionaIng(P, brie);
    .mexer(2);
    .adicionaIng(P, manteiga);
    .print("fim do plano addIngMexer").
```

# APPENDIX 4 <MOFICATION FUNCTION FOR SEMANTICALY ENHANCED BELIEFS>

```
package reason;

import java.util.Random;
import java.util.Set;

import jasdl.bridge.JASDLOntologyManager;
import jasdl.bridge.factory.SELiteralFactory;
import jasdl.bridge.seliteral.SELiteral;
import jasdl.bridge.seliteral.SELiteralClassAssertion;
import jasdl.bridge.seliteral.SELiteralObjectPropertyAssertion;
import jasdl.util.exception.JASDLException;
import jasdl.util.exception.JASDLInvalidSELiteralException;
import jason.asSemantics.Agent;
import jason.asSyntax.ASSyntax;
import jason.asSyntax.Literal;
import jason.asSyntax.Term;
import jason.asSyntax.parser.ParseException;

import org.semanticweb.owlapi.model.OWLClass;
import org.semanticweb.owlapi.model.OWLClassExpression;
import org.semanticweb.owlapi.model.OWLDataFactory;
import org.semanticweb.owlapi.model.OWLNamedIndividual;
import org.semanticweb.owlapi.model.OWLOntology;
import org.semanticweb.owlapi.model.OWLOntologyCreationException;
import org.semanticweb.owlapi.model.PrefixManager;

import agent.OWLHelper;

import com.clarkparsia.pellet.owlapiv3.PelletReasoner;
import com.clarkparsia.pellet.owlapiv3.PelletReasonerFactory;

public class SEBModification implements Modification {

    private JASDLOntologyManager jom;
    SELiteralFactory selFactory;
    private OWLHelper ohelper;
    //private JASDLAgentConfigurator config;

    public SEBModification(Agent owner){
            jom = new JASDLOntologyManager(owner.getLogger());
            selFactory = new SELiteralFactory(jom);
            ohelper = new OWLHelper();
            //config = new JASDLAgentConfigurator(owner);
    }

    public Class getType(){
            return Literal.class;
    }

    public Object modify(Object con) throws NotModificableException{
            if (!(con instanceof Literal)){
                    throw new NotModificableException();
            }
            Literal concept = (Literal)con;
            try {
                    SELiteral se = selFactory.construct(concept);
```

```
                OWLOntology seOnto = ohelper.loadOntoFromFile(
                        OWLHelper.ONTO_DIR + se.getOntologyAnnotation() + ".owl");

                        PelletReasoner                          reasoner                        =
PelletReasonerFactory.getInstance().createReasoner( seOnto );
                reasoner.getKB().realize();

                PrefixManager pm = ohelper.getPM();
                OWLDataFactory df = ohelper.getDataFactory();
                Random r = new Random();

                        if (se instanceof SELiteralClassAssertion){
                                SELiteralClassAssertion sec = se.asClassAssertion();
                                String classname = sec.getLiteral().getFunctor();
                                OWLClass oc = df.getOWLClass(":"+classname, pm);
                                //agora pegar as instancias de oc
                                Set<OWLNamedIndividual>     si     =     reasoner.getInstances(oc,
false).getFlattened();
                                //sortear uma, substituir e retornar
                                if (!si.isEmpty()){
                                        OWLNamedIndividual      ch      =      (OWLNamedIndividual)
si.toArray()[r.nextInt(si.size())];
                                        int i = ch.getIRI().toString().indexOf(":");
                                        String iname = ch.getIRI().toString().substring(i+1);
                                        Term t = ASSyntax.parseTerm(iname);
                                        Literal                         ret                         =
ASSyntax.createLiteral(se.getLiteral().getFunctor(), t);
                                        ret.addAnnot(se.getOntologyAnnotation());
                                        return ret;
                                }
                                throw new NotModificableException();
                        }else if (se instanceof SELiteralObjectPropertyAssertion){
                                SELiteralObjectPropertyAssertion                seo                 =
se.asObjectPropertyAssertion();
                                //verifica se possui range definition
                                String functor = seo.getLiteral().getFunctor();
                                org.semanticweb.owlapi.model.OWLObjectProperty oa =
                                        df.getOWLObjectProperty(":"+functor, pm);
                                Set<OWLClassExpression> ranges = oa.getRanges(seOnto);
                                if (!ranges.isEmpty()){
                                        //possui range, obter algum individuo que se adeque a def.
                                        //escolhe uma def. aleatoreamente
                                        int i = r.nextInt(ranges.size());
                                        OWLClassExpression      c      =      (OWLClassExpression)
ranges.toArray()[i];
                                        Set<OWLNamedIndividual>   si   =   reasoner.getInstances(c,
false).getFlattened();
                                        //escolhe um individuo
                                        i = r.nextInt(si.size());
                                        OWLNamedIndividual      ind      =      (OWLNamedIndividual)
si.toArray()[i];
                                        Term t1 = seo.getLiteral().getTerm(0).clone();
                                        i = ind.getIRI().toString().indexOf(":");
                                        String iname = ind.getIRI().toString().substring(i+1);
                                        Term t = ASSyntax.parseTerm(iname);
                                        Literal                         ret                         =
ASSyntax.createLiteral(se.getLiteral().getFunctor(), t1, t);
                                        ret.addAnnot(se.getOntologyAnnotation());
                                        return ret;
                                }else{
                                        //nao possui range, entao inferir o tipo
                                        String uri = seo.getSubject().getURI().toString();
                                        int i = uri.indexOf("#");
                                        String iname = uri.substring(i+1);
                                        OWLNamedIndividual                subject                 =
ohelper.getDataFactory().getOWLNamedIndividual(":"+iname, pm);
                                        org.semanticweb.owlapi.model.OWLObjectProperty pa =

        ohelper.getDataFactory().getOWLObjectProperty(":"+seo.getLiteral().getFunctor(), pm);
                                        Set<OWLNamedIndividual>                objs                 =
reasoner.getObjectPropertyValues(subject, pa).getFlattened();
                                        //escolher um individuo aleatoriamente
                                        i = r.nextInt(objs.size());
```

```
                                        OWLNamedIndividual                object                =
(OWLNamedIndividual)objs.toArray()[i];
                                        Set<OWLClass>    classes    =    reasoner.getTypes(object,
false).getFlattened();
                                        //escolhe uma classe e entao um individuo dessa classe
                                        i = r.nextInt(classes.size());
                                        OWLClass c = (OWLClass) classes.toArray()[i];
                                        objs = reasoner.getInstances(c, false).getFlattened();
                                        i = r.nextInt(objs.size());
                                        OWLNamedIndividual                change                =
(OWLNamedIndividual)objs.toArray()[i];

                                        Term t1 = seo.getLiteral().getTerm(0).clone();
                                        i = change.getIRI().toString().indexOf(":");
                                        iname = change.getIRI().toString().substring(i+1);
                                        Term t = ASSyntax.parseTerm(iname);
                                        Literal                        ret                        =
ASSyntax.createLiteral(se.getLiteral().getFunctor(), t1, t);
                                        ret.addAnnot(se.getOntologyAnnotation());
                                        return ret;
                                }
                        }
                        else throw new NotModificableException();
                } catch (JASDLInvalidSELiteralException e) {
                        // provavelmente nao eh um SELit entao nao pode ser modificado
                        throw new NotModificableException(e);
                }catch (OWLOntologyCreationException e) {
                        throw new NotModificableException(e);
                } catch (ParseException e) {
                        throw new NotModificableException(e);
                } catch (JASDLException e) {
                        throw new NotModificableException(e);
                }
        }

    }
```

# APPENDIX 5 <RANDOM MOFICATION FUNCTION FOR BELIEFS>

```
package reason;

import java.util.Random;

import jason.asSyntax.ASSyntax;
import jason.asSyntax.Literal;
import jason.asSyntax.Term;
import jason.asSyntax.parser.ParseException;

public class RBeliefModfication implements Modification {

    Float chance;
    Random r;

    public RBeliefModfication(Float chance){
            this.chance = chance;
            r = new Random();
    }

    public Class getType(){
            return Literal.class;
    }

    public Object modify(Object c) throws NotModificableException{
            //selecao aleatoria do termo, uma vez,
            //tamanho do termo
            if (!(c instanceof Literal)){
                    throw new NotModificableException();
            }
            Literal concept = (Literal)c;
            Literal ret =  (Literal) concept.clone();
            if (r.nextFloat() < chance){
                    int termo = r.nextInt(ret.getArity());
                    Term t = concept.getTerm(termo);
                    try{
                            Term mod = null;
                            if (t.isString()){
                                    mod = ASSyntax.parseTerm(stringMod(t.toString()));
                            }else if (t.isNumeric()){
                                    Integer imod = integerMod(Integer.valueOf(t.toString()));
                                    mod = ASSyntax.parseTerm(imod.toString());
                            }
                            ret.setTerm(termo, mod);
                            return ret;
                    }catch(ParseException pe){
                            throw new NotModificableException(pe);
                    }catch(NumberFormatException e){
                            throw new NotModificableException(e);
                    }
            }
            return concept;
    }

    String stringMod(String c){
            //A-Z: 65-90
            //a-z: 97-122
```

```
            //quantos caracteres?
            char[] ret = c.toCharArray();
            int cs = r.nextInt(c.length());
            for (int i = 0; i < cs; i++) {
                    //qual caracter?
                    int ca = r.nextInt(c.length());
                    //numero ou caracter?
                    if (r.nextBoolean())
                            ret[ca] =  (char)((char)r.nextInt(26) + 65);
                    else
                            ret[ca] = (char)r.nextInt(9);
            }
            return String.copyValueOf(ret);
    }

    Integer integerMod(Integer c){
            return r.nextInt();
    }
}
```

# APPENDIX 6 <FUNCTOR COMPARISON FUNCTION>

```
package kr;

import jason.asSyntax.Literal;

public class EqualsFunctor extends BeliefStringSimilarity implements SimpleConceptRelator {

    @Override
    public Object relateO(Object t1, Object t2) throws NotRelationalException {
            if (t1 instanceof Literal && t2 instanceof Literal){
                    String f1 = ((Literal) t1).getFunctor();
                    String f2 = ((Literal) t2).getFunctor();
                    if (f2.equals(f1))
                            return f1;
            }
            throw new NotRelationalException();
    }

    @Override
    public boolean equals(Object obj) {
            if (obj == null)
                    return false;
            if ( this == obj )
                    return true;
        if ( !(obj instanceof EqualsFunctor) )
            return false;
        return obj.getClass().getName().equals(this.getClass().getName());

    }
}
```

# APPENDIX 7 <LITERAL INTERSECTION COMPARISON FUNCTION>

```
package kr;

import jason.asSyntax.Literal;

public class BeliefStringSimilarity implements ConceptRelator<Literal, Literal, String>,
SimpleConceptRelator {

	public Class<?> getConcept1Type() {
		// TODO Auto-generated method stub
		return Literal.class;
	}

	public Class<?> getConcept2Type() {
		// TODO Auto-generated method stub
		return Literal.class;
	}

	public Class<?> getReturnType() {
		// TODO Auto-generated method stub
		return String.class;
	}

	public String relate(Literal t1, Literal t2) throws NotRelationalException {
		if (t1.equals(t2))
			return null;
		return LiteralHelper.literalRelation(t1, t2);
	}

	public Class<?> getKRType(){
		return Literal.class;
	}



	public static void main(String[] args) {
		// TODO Auto-generated method stub
		BeliefStringSimilarity teste = new BeliefStringSimilarity();
		System.out.print(teste.getClass());

	}

	public Object relateO(Object t1, Object t2) throws NotRelationalException {
		if (!(t1 instanceof Literal) || !(t2 instanceof Literal))
			throw new NotRelationalException();
		return LiteralHelper.literalRelation((Literal)t1, (Literal)t2);
	}

	@Override
	public boolean equals(Object obj) {
		if (obj == null)
			return false;
		if ( this == obj )
			return true;
```

```
        if ( !(obj instanceof BeliefStringSimilarity) )
            return false;
        return obj.getClass().getName().equals(this.getClass().getName());
    }

    @Override
    public int hashCode() {
            // TODO Auto-generated method stub
            return this.getClass().getName().toString().hashCode();
    }

    @Override
    public String toString() {
            return this.getClass().getName();
    }


 }

package kr;

import jason.asSyntax.Literal;
import jason.asSyntax.Term;

import java.util.*;

public class LiteralHelper {

    public static boolean functorComparisson(String functor, String comp){
    return functor.contains(comp);
    }

    public static boolean termComparisson(Term t, Term tcomp){
    return t.equals(tcomp);
    }

    public static boolean listTermComparisson(List<Term> lt, Term tcomp){
    return lt.contains(tcomp);
    }

    public static Term listTermComparisson(List<Term> lt1, List<Term> lt2){
            Iterator<Term> it = lt1.iterator();
            while (it.hasNext()) {
                    Term t = it.next();
                    if (listTermComparisson(lt2, t))
                            return t;
            }
    return null;
    }

    public static String literalRelation(Literal l1, Literal l2){
            //se os literais forem iguais esse metodo retorna falso
            if (l1.toString().equals(l2.toString()))
                    return null;
            //comparacao de functor com functor
            String lf1 = l1.getFunctor();
            String lf2 = l2.getFunctor();
            if (lf1.equalsIgnoreCase(lf2) || lf1.contains(lf2) || lf2.contains(lf1)){
                    return lf1;
            }
    //comparacao do functor com o restante do literal
            String lf = l1.getFunctor();
            System.out.println("FUNCTOR LITERAL 1: " + lf);
            if (functorComparisson(lf, l2.toString())){
                    return lf;
            }

    lf = l2.getFunctor();

    if (functorComparisson(lf, l1.toString())){
            return lf;
    }
```

```
        //comparacao da lista de termos com o restante do literal
        List<Term> lt1 = l1.getTerms();
        List<Term> lt2 = l2.getTerms();
        Term r = listTermComparisson(lt1, lt2);
        if (r != null){
                return r.toString();
        }
        r = listTermComparisson(lt2, lt1);
        if (r != null){
                return r.toString();
        }
                return null;
       }

}
```

# APPENDIX 8 <EVENT SIMULATION COMPARISON FUNCTION>

```
package kr;

import jason.asSyntax.*;

public class PlanTriggerEnabler implements ConceptRelator<Literal, Plan, Boolean>,
SimpleConceptRelator {

	public Object relateO(Object t1, Object t2) throws NotRelationalException {
		if (t1 instanceof Literal && t2 instanceof Plan){
			return PlanHelper.triggersPlan((Literal)t1, (Plan)t2);
		}
		throw new NotRelationalException();
		//return null;
	}


	public Class<?> getConcept1Type() {
		return Literal.class;
	}


	public Class<?> getConcept2Type() {
		return Plan.class;
	}


	public Class<?> getReturnType() {
		return Boolean.class;
	}


	public Boolean relate(Literal t1, Plan t2) throws NotRelationalException {
		// TODO Auto-generated method stub
		return PlanHelper.triggersPlan(t1, t2);
	}


	public int hashCode() {
		// TODO Auto-generated method stub
		return this.getClass().getName().toString().hashCode();
	}


	public String toString() {
		return this.getClass().getName();
	}


	public boolean equals(Object obj) {
		if (obj == null)
			return false;
		if ( this == obj )
			return true;
	    if ( !(obj instanceof PlanTriggerEnabler) )
		    return false;
```

```
            return obj.getClass().getName().equals(this.getClass().getName());

        }

    }


    package kr;

    import jason.JasonException;
    import jason.asSyntax.Literal;
    import jason.asSyntax.Plan;
    import jason.asSyntax.PlanLibrary;
    import jason.asSyntax.Trigger;

    import java.util.HashMap;
    import java.util.List;
    import java.util.Map;

    public class PlanHelper extends LiteralHelper {

        protected static Map<Trigger, List<Plan>> relatedByTrigger(Literal search, PlanLibrary
plans) throws Exception{

                Map<Trigger, List<Plan>> relatedByTrigger = new HashMap<Trigger, List<Plan>>();

                Trigger test1 = new Trigger(Trigger.TEOperator.add, Trigger.TEType.belief, search);
            Trigger test2 = new Trigger(Trigger.TEOperator.add, Trigger.TEType.achieve, search);
            Trigger test3 = new Trigger(Trigger.TEOperator.add, Trigger.TEType.test, search);
            Trigger test4 = new Trigger(Trigger.TEOperator.del, Trigger.TEType.belief, search);
            Trigger test5 = new Trigger(Trigger.TEOperator.del, Trigger.TEType.achieve, search);
            Trigger test6 = new Trigger(Trigger.TEOperator.del, Trigger.TEType.test, search);

            /*Use Trigger class to simulate possible uses / applications of the belief.
             * Achievable, testeable, from a belief, when added, when deleted
             */
            relatedByTrigger.put(test1, plans.getCandidatePlans(test1));
            relatedByTrigger.put(test2, plans.getCandidatePlans(test2));
            relatedByTrigger.put(test3, plans.getCandidatePlans(test3));
            relatedByTrigger.put(test4, plans.getCandidatePlans(test4));
            relatedByTrigger.put(test5, plans.getCandidatePlans(test5));
            relatedByTrigger.put(test6, plans.getCandidatePlans(test6));

            return relatedByTrigger;
        }

        protected static boolean triggersPlan(Literal search, Plan p) {

                Trigger test1 = new Trigger(Trigger.TEOperator.add, Trigger.TEType.belief, search);
            Trigger test2 = new Trigger(Trigger.TEOperator.add, Trigger.TEType.achieve, search);
            Trigger test3 = new Trigger(Trigger.TEOperator.add, Trigger.TEType.test, search);
            Trigger test4 = new Trigger(Trigger.TEOperator.del, Trigger.TEType.belief, search);
            Trigger test5 = new Trigger(Trigger.TEOperator.del, Trigger.TEType.achieve, search);
            Trigger test6 = new Trigger(Trigger.TEOperator.del, Trigger.TEType.test, search);

            if (p.isRelevant(test1)!=null)
                return true;
            if (p.isRelevant(test2)!=null)
                return true;
            if (p.isRelevant(test3)!=null)
                return true;
            if (p.isRelevant(test4)!=null)
                return true;
            if (p.isRelevant(test5)!=null)
                return true;
            if (p.isRelevant(test6)!=null)
                return true;
            return false;
        }

    }
```

# APPENDIX 9<LEXICAL SIMILARITY COMPARISON FUNCTION>

```
package kr;

public class StringContains extends BeliefStringSimilarity implements SimpleConceptRelator {

    @Override
    public Class<?> getConcept1Type() {
            // TODO Auto-generated method stub
            return String.class;
    }

    @Override
    public Class<?> getConcept2Type() {
            // TODO Auto-generated method stub
            return String.class;
    }

    @Override
    public Class<?> getReturnType() {
            // TODO Auto-generated method stub
            return String.class;
    }

    @Override
    public Object relate0(Object t1, Object t2) throws NotRelationalException {
            if (t1 instanceof String && t2 instanceof String){
                    String s1 = (String) t1;
                    String s2 = (String) t2;
                    if (s2.contains(s1))
                            return s1;
                    //if (s1.contains(s2))
                    //      return s2;
                    return null;
            }throw new NotRelationalException();
            //return null;
    }

    @Override
    public boolean equals(Object obj) {
            if (obj == null)
                    return false;
            if ( this == obj )
                    return true;
        if ( !(obj instanceof StringContains) )
            return false;
        return obj.getClass().getName().equals(this.getClass().getName());

    }

}
```

# APPENDIX 10<RANDOM SOP>

```
package reason;

import java.util.Random;

import reason.Blender.OP;

public class RandomSop implements Sop {

    public OP sop() {
            Random r = new Random();
            float f = r.nextFloat();
            if (f <= 0.33 )
                    return Blender.OP.MODIFICATION;
            else if (f > 0.33 && f <= 0.66)
                    return Blender.OP.COMPLETION;
            else
                    return Blender.OP.COMPOSITION;
    }
}
```

# APPENDIX 11<ITERATION ϕ>

```
package reason;

public class IterationPhi implements Phi{

    int counter = 0;
    int lim;

    public IterationPhi(int lim){
            this.lim = lim;
    }

    public boolean phi(){

            return true;
    }

}
```

# APPENDIX 12<BLENDING INTERNAL ACTION>

```
package agent.ia;

import java.util.HashSet;
import java.util.Set;

import org.semanticweb.owlapi.model.OWLIndividual;
import org.semanticweb.owlapi.model.OWLNamedIndividual;
import org.semanticweb.owlapi.model.OWLOntology;
import org.semanticweb.owlapi.model.OWLOntologyCreationException;

import reason.Alpha;
import reason.Blender;
import reason.IterationPhi;
import reason.Modification;
import reason.OWLBlender;
import reason.RandomSop;
import reason.Sop;

import jason.asSemantics.DefaultInternalAction;
import jason.asSemantics.TransitionSystem;
import jason.asSemantics.Unifier;
import jason.JasonException;
import jason.asSyntax.*;
import kr.SimpleConceptRelator;

@SuppressWarnings("serial")
public class RecommendationBlend extends DefaultInternalAction {

    RecOWLBridge rec;

    public RecommendationBlend(){
            rec = new RecOWLBridge();
    }

    protected void checkArguments(Term[] args) throws JasonException {
       super.checkArguments(args); // check number of arguments
       if (! (args[0] instanceof ListTerm))
           throw JasonException.createWrongArgument(this,
                   "first argument must be a List of Terms representing the user model");
       if (! (args[1] instanceof VarTerm))
           throw JasonException.createWrongArgument(this,
                   "second argument must be Variable to hold the reference to the blend");
       if (! (args[1] instanceof StringTerm))
           throw JasonException.createWrongArgument(this,
                   "third argument must be StringTerm defining the k. See the API Doc for
options.");
       if (! (args[1] instanceof StringTerm))
           throw JasonException.createWrongArgument(this,
                   "fourht argument must be StringTerm defining the alpha function. See the
API Doc for options.");
       if (! (args[1] instanceof StringTerm))
           throw JasonException.createWrongArgument(this,
                   "fifth argument must be StringTerm defining if the constitutive principles
are applied or not.");
    }
```

```
        public Object execute(TransitionSystem ts, Unifier un, Term[] args) throws Exception {
            // execute the internal action
            ts.getAg().getLogger().info("executing internal action 'agent.ia.Blender'");

            checkArguments(args);

            ListTerm userModelB = (ListTerm) args[0];
            //1. criar o input 1 - OWL Ontology
            OWLOntology i1 = recI1(userModelB);
            //2. criar o input 2 - OWL Ontology
            OWLOntology i2 = recI2();
            //3. funcoes de comparacao
            Set<Modification> mods = recMods();
                 Set<SimpleConceptRelator> fcs = recRelators();
                 Alpha a = alphaByName(args[3].toString());
                 if (a == null)
                         throw new Exception("unknown alpha function");
                 String sopt = args[4].toString();
                 Sop sop = null;
                 if (!sopt.equals("none"))
                         sop = new RandomSop();
            OWLBlender      blender      =      new      OWLBlender(i1,      i2,      mods,      fcs,
Blender.K.valueOf(args[2].toString()),
                         a, new IterationPhi(10), sop);
                 OWLOntology retb = blender.blend();

            ListTerm ret = rec.ontoToListTerm(retb);
            return un.unifies(args[1], ret);
        }

        public OWLOntology recI1(ListTerm bs){
                 OWLOntology ret = rec.newOnto("r1");
                 OWLIndividual cs1 = rec.newIndividual("usermodel", rec.user, ret);
                 for (Term term : bs) {
                         if (term.isLiteral()){
                                 Literal l = (Literal)term;
                                 String functor = l.getFunctor();
                                 if (functor.equals("person") && l.getArity()==1){
                                         OWLIndividual                       p                       =
rec.userIndividual(l.getTerm(0).toString(), ret);
                                         rec.obpToIndividual(rec.hasPerson, cs1, p, ret);
                                 }else if (functor.equals("foafinterest") && l.getArity()==2){
                                         OWLIndividual                       p                       =
rec.userIndividual(l.getTerm(0).toString(), ret);
                                         OWLIndividual                       i                       =
rec.newIndividual(l.getTerm(1).toString(), rec.interest, ret);
                                         rec.obpToIndividual(rec.foafInterest, p, i, ret);
                                 }else if (functor.equals("topicInterest") && l.getArity()==2){
                                         OWLIndividual                       p                       =
rec.userIndividual(l.getTerm(0).toString(), ret);
                                         OWLIndividual                       i                       =
rec.newIndividual(l.getTerm(1).toString(), rec.interest, ret);
                                         rec.obpToIndividual(rec.foafTopicInterest, p, i, ret);
                                 }else if (functor.equals("item") && l.getArity()==1){
                                         OWLIndividual                       p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.item, ret);
                                         rec.obpToIndividual(rec.hasItem, cs1, p, ret);
                                 }else if (functor.equals("itemModel") && l.getArity()==1){
                                         OWLIndividual                       p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.itemModel, ret);
                                         rec.obpToIndividual(rec.hasItemModel, cs1, p, ret);
                                 }else if (functor.equals("ldactivity") && l.getArity()==1){
                                         OWLIndividual                       p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.ld_activity, ret);
                                         rec.obpToIndividual(rec.hasLearningActivity, cs1, p, ret);
                                 }else if (functor.equals("environment") && l.getArity()==1){
                                         OWLIndividual                       p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.environment, ret);
                                         rec.obpToIndividual(rec.hasEnvironment, cs1, p, ret);
                                 }else if (functor.equals("learningObjective") && l.getArity()==1){
                                         OWLIndividual                       p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.learningObjective, ret);
                                         rec.obpToIndividual(rec.hasLearningObjective,    cs1,    p,
```

```
ret);
                                }else if (functor.equals("goal") && l.getArity()==1){
                                        OWLIndividual                   p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.goal, ret);
                                        rec.obpToIndividual(rec.hasGoal, cs1, p, ret);
                                }else if (functor.equals("interest") && l.getArity()==1){
                                        OWLIndividual                   p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.interest, ret);
                                        rec.obpToIndividual(rec.hasInterest, cs1, p, ret);
                                }else if (functor.equals("accessibility") && l.getArity()==1){
                                        OWLIndividual                   p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.accessibility, ret);
                                        rec.obpToIndividual(rec.hasAccessibility, cs1, p, ret);
                                }else if (functor.equals("lipactivity") && l.getArity()==1){
                                        OWLIndividual                   p                       =
rec.newIndividual(l.getTerm(0).toString(), rec.lip_activity, ret);
                                        rec.obpToIndividual(rec.hasLIP_Activity, cs1, p, ret);
                                }else if (functor.equals("noattempts") && l.getArity()==1){
                                        rec.dpToIndividual(l.getTerm(0).toString(),
rec.hasNoOfAttempts, cs1, ret);
                                }else if (functor.equals("hasScore") && l.getArity()==1){
                                        rec.dpToIndividual(l.getTerm(0).toString(),   rec.hasScore,
cs1, ret);
                                }
                        }
                }
                return ret;
        }

        public OWLOntology recI2(){
                OWLOntology ret = rec.newOnto("r2");
                OWLIndividual cs2 = rec.newRecI2Individual("ri2");
                try {
                        Set<OWLNamedIndividual> is = rec.obtainLO();
                        for(OWLNamedIndividual i : is){
                                rec.obpToIndividual(rec.hasOA, cs2, i, ret);
                        }
                } catch (OWLOntologyCreationException e) {
                        e.printStackTrace();
                }
                return ret;
        }

        public Alpha alphaByName(String fname){
                if (fname.equals("metaphor"))
                        return new MetaphorAlpha();
                else if (fname.equals("counterpart"))
                        return new CounterPartAlpha();
                else if (fname.equals("surprise"))
                        return new SurpriseAlpha();
                return null;
        }

        public Set<Modification> recMods(){
                Set<Modification> ret = new HashSet<Modification>();
                OWLIModification om = new OWLIModification();
                ret.add(om);
                return ret;
        }

        public Set<SimpleConceptRelator> recRelators(){
                Set<SimpleConceptRelator> ret = new HashSet<SimpleConceptRelator>();
                WikiSim ws = new WikiSim(0.8);
                LexicSim ls = new LexicSim();
                HierarchySim hs = new HierarchySim();
                ret.add(ws);
                ret.add(ls);
                ret.add(hs);
                return ret;
        }

    }
```

# APPENDIX 13 <HIERARCHY-BASED COMPARISON FUNCTION>

```
package agent.ia;

import java.util.Set;

import org.semanticweb.owlapi.model.OWLClass;
import org.semanticweb.owlapi.model.OWLClassExpression;
import org.semanticweb.owlapi.model.OWLNamedIndividual;
import org.semanticweb.owlapi.model.OWLOntology;
import org.semanticweb.owlapi.reasoner.NodeSet;

import com.clarkparsia.pellet.owlapiv3.PelletReasoner;
import com.clarkparsia.pellet.owlapiv3.PelletReasonerFactory;

import kr.NotRelationalException;
import kr.SimpleConceptRelator;

public class HierarchySim implements SimpleConceptRelator {

    RecOWLBridge rec;
    Set<OWLOntology> ontos;

    public HierarchySim(Set<OWLOntology> onts){
            rec = new RecOWLBridge();
            this.ontos = onts;
    }

    public Class<?> getConcept1Type() {
            return OWLNamedIndividual.class;
    }

    public Class<?> getConcept2Type() {
            return OWLNamedIndividual.class;
    }

    public Class<?> getReturnType() {
            return Boolean.class;
    }

    public Object relateO(Object t1, Object t2) throws NotRelationalException {
            if (t1 instanceof OWLNamedIndividual && t2 instanceof OWLNamedIndividual){
                    OWLNamedIndividual i1 = (OWLNamedIndividual)t1;
                    OWLNamedIndividual i2 = (OWLNamedIndividual)t2;
                    for (OWLOntology o : ontos){
                            PelletReasoner                    reasoner                    =
PelletReasonerFactory.getInstance().createReasoner( o );
                            reasoner.getKB().realize();
                            //NodeSet<OWLClass> cls1 = reasoner.getTypes(i1, true);
                            NodeSet<OWLClass> cls2 = reasoner.getTypes(i2, true);
                            Set<OWLClassExpression> cle1 = i1.getTypes(o);
                            //Set<OWLClassExpression> cle2 = i2.getTypes(o);
                            for (OWLClassExpression cl1 : cle1){
                                    NodeSet<OWLClass>   ret   =   reasoner.getSuperClasses(cl1,
true);
```

```
                                        //tenho as superclasses de i1
                                        //agora verificar se alguma delas faz parte da definicao
de i2
                                        for (OWLClass oc : ret.getFlattened()){
                                                if (cls2.containsEntity(oc))
                                                        return true;
                                        }
                                }
                        }
                        return false;
                }
                throw new NotRelationalException();
        }

    }
```

# APPENDIX 14 <WORD SIMILARITY COMPARISON FUNCTION>

```
package agent.ia;

import java.util.Set;

import org.semanticweb.owlapi.model.OWLClassExpression;
import org.semanticweb.owlapi.model.OWLNamedIndividual;
import org.semanticweb.owlapi.model.OWLOntology;

import ponzo.nlp.wikipedia.similarity.WikiSimilarity;
import ponzo.nlp.wikipedia.similarity.WikiSimilarityFactory;

import kr.NotRelationalException;
import kr.SimpleConceptRelator;

public class WikiSim implements SimpleConceptRelator {

    RecOWLBridge rec;
    Set<OWLOntology> ontos;
    Double limit;

    public WikiSim(Set<OWLOntology> onts, Double limit){
            rec = new RecOWLBridge();
            this.ontos = onts;
            this.limit = limit;
    }

    public Class<?> getConcept1Type() {
            return OWLNamedIndividual.class;
    }

    public Class<?> getConcept2Type() {
            return OWLNamedIndividual.class;
    }

    public Class<?> getReturnType() {
            return Boolean.class;
    }

    public Object relateO(Object t1, Object t2) throws NotRelationalException {
            if (t1 instanceof OWLNamedIndividual && t2 instanceof OWLNamedIndividual){
                    OWLNamedIndividual i1 = (OWLNamedIndividual)t1;
                    OWLNamedIndividual i2 = (OWLNamedIndividual)t2;
                    Set<OWLClassExpression> cls1 = i1.getTypes(ontos);
                    Set<OWLClassExpression> cls2 = i2.getTypes(ontos);
                    WikiSimilarityFactory ws = WikiSimilarityFactory.getInstance();
                    for (OWLClassExpression cl1 : cls1){
                            if (!cl1.isOWLNothing() && !cl1.isOWLThing()){
                                    String c1 = cl1.getClassExpressionType().getName();
                                    for (OWLClassExpression cl2 : cls2){
                                            if (!cl1.isOWLNothing() && !cl1.isOWLThing()){
                                                    String                c2                =
cl2.getClassExpressionType().getName();
                                                    WikiSimilarity          result          =
ws.getWikiSimilarity(c1, c2);
                                                    if (result.getAverageResnik() > limit)
                                                            return true;
```

```
                                        }
                                    }
                                }
                            }
                            return false;
                        }
                        throw new NotRelationalException();
                    }

                }
```

# APPENDIX 15 <RESUMO EM PORTUGUÊS>

O presente resumo em português apresenta os objetivos e principais resultados e contribuições desta Tese.

## Introdução

Como nosso título sugere, estamos interessados em aplicar criatividade a uma estrutura de agentes. Entretanto, também podemos considerar a influência de uma estrutura de intencionalidade (representada pelo agente) no raciocínio criativo. Ambas perspectivas são estudadas neste trabalho. Neste projeto consideramos agência como uma conceitualização advinda da filosofia que representa a habilidade humana de decidir e atuar autonomamente. Esse conceito é adotado em ciências sociais, psicologia, ciências cognitivas, biologia, economia e ciência da computação. No âmbito da computação, agência é um campo de pesquisa pertencente à Inteligência Artificial o qual desafia os pesquisadores a fornecer teorias e modelos computacionais para autonomia.

Criatividade está relacionada ao processo realizado por seres humanos para criar novos conceitos e artefatos – no sentindo mais amplo possível. Como um processo, a criatividade é estudada principalmente pelas neurociências, ciências cognitivas, psicologia e filosofia. Como uma propriedade (atribuída a algo concreto ou abstrato), ela é tipicamente estudada pelas ciências sociais e apreciação de artes. Ambas visões (de processo e de propriedade) são objetos de estudo da criatividade computacional.

Assim como na maioria dos sub-campos da Inteligência Artificial, agência e criatividade computacionais concentram muitos esforços de pesquisa nas ligações multi e inter disciplinares, geralmente resultando em modelos e observações úteis para todas as áreas envolvidas. Nosso roteiro de pesquisa é motivado pela idéia de que modelos de IA devem servir como meios para auxiliar no desafio do entendimento da mente humana. Desta forma, estamos interessados no estudo da habilidade humana de utilizar experiências e conhecimento prévio para lidar com situações novas, criando possibilidades e soluções a partir do momento em que o problema é apresentado.

Assim, nossa abordagem para estudar esse comportamento é através de um modelo de criatividade como um mecanismo de raciocínio integrado a uma estrutura de agentes. Considerando a criatividade como um processo representado por inferência computacional é possível posicionar o trabalho numa perspectiva de Representação de Conhecimento e Raciocínio (*Knowledge Representation & Reasoning*) da IA (HARMELEN; LIFSCHITZ; PORTER, 2007). Grande parte dos trabalhos em IA podem ser categorizadas em termos de como representam seu conhecimento e realizam inferências a partir do mesmo (também considerados como aspectos estáticos e dinâmicos). Além disso, cada uma destas categorias é, geralmente, resultado de estudos acerca de como nós utilizamos o conhecimento em nossas mentes.

Por exemplo, trabalhos pioneiros em IA foram fortemente influenciados por filosofia e matemática, resultando em uma das abordagens mais bem sucedidas da área: raciocínio baseado em lógica (HOFWEBER, 2009; COLMERAUER, 1985; KOWALSKI, 1986). Atualmente, trabalhos em lógica computacional e suas aplicações ainda fornecem importantes resultados para a IA e computação em geral. Os trabalhos

mais recentes em lógica procuram melhorar o desempenho dos motores de inferência e, ao mesmo tempo, manter e até melhorar sua expressividade. Outra linha de pesquisa em lógica computacional trabalha com formalismos lógicos para representar conhecimento temporal, causal, ontológico, probabilístico, espacial, entre outros.

Entretanto, percebendo que nem tudo pode ser modelado com lógica, ou que lógica não é adequada para todo o tipo de conhecimento, paradigmas híbridos ou totalmente diferentes em sua concepção foram desenvolvidos. Este é o caso dos frames (MINSKY, 1974) e redes semânticas (QUILIAN, 1968) que possuem suas fundamentações em descobertas da psicologia e foram desenvolvidos para acomodar naturalmente inferências em hierarquias e classificação. Por outro lado, redes Bayesianas (PEARL, 1985), originárias da estatística, são modeladas computacionalmente para representar causalidade e propagação de evidências em grandes correntes de conhecimento. Em oposição direta às abordagens simbólicas, redes neurais (MCCULLOCH; PITTS, 1943) constituem a abordagem mais eminente das representações sub-simbólicas (conexionismo). Seguindo inspirações das neurociências e biologia, as redes neurais fornecem diversas formas para implementar aprendizagem automática ou semi-automática.

Neste projeto, seguimos desenvolvimentos das ciências cognitivas na Fusão Conceitual – tradução do autor – (*Concept Blending*) (FAUCONNIER; TURNER, 1998), considerada como uma habilidade humana, inata e sub-consciente, que integra conhecimento gerando novas conceitualizações. Segundo esta teoria a integração de conceitos ocorre de acordo com princípios constitutivos e governantes. Os princípios constitutivos definem as regras e operações realizadas para formar um modelo de rede conceitual organizado de acordo com a vivência humana do mundo. Assim, diferentes perspectivas organizacionais podem ser aplicadas a um mesmo conceito ou agrupamento de conceitos. Por exemplo, perspectivas emocionais, sensoriais, temporais, causais, entre outras.

Portanto, o processo de fusão pode ser visto como o processo pelo qual a rede de conceitos é expandida, em analogia com a nossa imaginação, que gera novos conceitos e interpretações a partir do que vivenciamos no mundo. Já os princípios governantes restringem as operações constitutivas na rede visando enfatizar as relações vitais, as quais representam um conjunto subjetivo de propriedades vitais para os seres humanos. De acordo com a teoria da fusão, estas relações estão presentes em praticamente todas nossas atividades diárias, tais como acordar, caminhar, tomar café, trabalhar, estudar, namorar, etc. As relações vitais especificadas por Fauconnier e Turner (2002) são: mudança, causa e efeito, identidade, intencionalidade, singularidade, tempo, espaço, representação, parte-todo, propriedade, categoria, analogia, papel, similaridade.

Dentro da criatividade computacional, a utilização da Fusão Conceitual (FC) para especificar o raciocínio criativo posiciona nosso trabalho em modelos gerais de criatividade humana. Modelos específicos são desenvolvidos para simular atividades criativas tais como composição musical (MARTINS, 2004; MARTINS; MIRANDA, 2006; PEARCE; MÜLLENSIEFEN; WIGGINS, 2008), pintura (COLTON, 2008), escrita de poesia (VEALE; HAO, 2008; HERVÁS, R. et al. 2007) e piadas (BINSTEAD, K. et al. 2006). Em modelos gerais de criatividade, a lacuna de conhecimento que estudamos é a consideração de conhecimento prático, juntamente com o teórico, durante o processo de raciocínio criativo. Desta forma, em termos mais teóricos, procuramos integrar raciocínio criativo em uma estrutura cognitiva mais ampla, permitindo que criatividade interaja com intencionalidade. Para alcançar esse

modelo computacional, propomos a utilização de constructos da teoria de agentes, indo em direção à criatividade baseada em agência.

O escopo de agência considerado nesse trabalho é o de agentes cognitivos que implementam sistemas intencionais. Especificamente, nós adotamos a abordagem de crenças, desejos e intenções de Bratman (*Belief, Desire and Intention* – BDI) como a teoria intencional que fundamenta o presente trabalho. Bratman (1987) pode ser considerado como um dos trabalhos mais influentes em agentes autônomos. Sua teoria BDI está enraizada na filosofia da mente e psicologia popular ou de senso comum (*folk psychology*). Nessa teoria, o conhecimento que o agente possui sobre o mundo é representado pelas crenças, os desejos representam como o agente quer que o mundo seja e as intenções especificam desejos os quais o agente está comprometido em atingir. Este paradigma para especificar intencionalidade inspirou o desenvolvimento de várias arquiteturas de agentes (KUMAR; SHAPIRO, 1994; MORLEY; MYERS, 2004; D'INVERNO, M. et al. 2004), linguagens (DASTANI, M. et al.. 2003; RAO; GEORGEFF 1991; RAO, 1996) e variantes da teoria em si (GOVERNATORI; ROTOLO, 2008; CHOLVY, 2004; BROERSEN, J. et al. 2001). A maioria destes trabalhos observa os agentes como sistemas com recursos limitados (*resource-bounding agency*), visão introduzida por Bratman (1998). A fundamentação filosófica e psicológica da teoria, juntamente com uma arquitetura prática (porém restritiva) são as principais razões para o sucesso da abordagem.

Considerando o desenvolvimento de agentes BDI, as principais abordagens seguem a linha de sistemas de raciocínio procedimental (Procedural Reasoning Sytems – PRS) (GEORGEFF; LANSKY, 1987; RAO; GEORGEFF, 1991). Tais sistemas adotam a abstração de planos, os quais permitem a expressão de conhecimento procedimental dentro da estrutura do agente (geralmente programado com linguagens declarativas). Nesse contexto, os planos especificam receitas pré-definidas para lidar com configurações de mundo particulares (definidas nas pré-condições dos planos). Assim, planos funcionam como heurísticas para reduzir o espaço de busca por uma opção viável, possibilitando ao agente realizar seu raciocínio prático – raciocínio para a realização de ações (WOOLDRIDGE, 1995) – em tempo hábil. Entretanto, ao mesmo tempo que a limitação de recursos permite ao agente interagir rapidamente com seu ambiente, ela também restringe as possibilidades de ação. Wooldridge (1995, 2000) argumenta que alcançar um equilíbrio entre raciocínio e atuação constitui-se como um dos principais desafios da pesquisa e desenvolvimento de agentes.

As pesquisas mais recentes para lidar com esse desafio seguem diferentes perspectivas, tais como o uso de emoções (JIANG; VIDAL; HUHNS, 2007; STEUNEBRINK; DASTANI; MEYER, 2007), normas (DASTANI; TINNEMEIER; MEYER, 2009; GANGEMI, 2008; CONTE; ANDRIGHETTO; CAMPENNÍ, 2009) e aprendizagem (SUBAGDJA; SONENBERG; RAHWAN, 2009; FUJITA, 2009; SEN; AIRIAU, 2007; SHOHAM; POWERS; GRENAGER, 2007; STONE, 2007). Nós limitamos nosso escopo à abordagens que aumentem a utilização do conhecimento do agente permitindo que o mesmo adapte-se a situações não previstas em sua biblioteca de planos (planejamento e aprendizagem de agentes).

Finalmente, nossa observação de agência criativa refere-se ao uso de raciocínio criativo para aumentar a aplicabilidade do conhecimento do agente (adaptação), inspirado na forma como nós, humanos, entendemos e vivenciamos o mundo.

## Objetivos

Resumindo nosso argumento introdutório, a motivação deste trabalho está no impacto dos modelos computacionais no entendimento da inteligência humana, especificamente, nossa habilidade de utilizar experiências prévias para lidar com novas situações. Esta pesquisa localiza-se na intersecção entre criatividade computacional e agência cognitiva. Dentro de criatividade computacional, posicionamos este trabalho modelos computacionais da criatividade humana seguindo a teoria da fusão conceitual. Na perspectiva de agentes, o trabalho posiciona-se em arquiteturas e linguagens BDI. Imbuídos de nossa motivação e contexto, o problema de pesquisa apresentado pode ser sintetizado em duas questões de pesquisa, norteadoras do presente trabalho:

Q1. Como a criatividade pode ser modelada computacionalmente e produzir conhecimento prático e teórico?

Q2. Como a criatividade apóia a utilização de conhecimento e experiências prévias como meios para a adaptação a situações imprevistas?

A partir dessas questões, definimos um objetivo principal e os respectivos objetivos intermediários para o presente trabalho:

- **Propor um modelo computacional para criatividade;**
    a. Especificar uma representação da fusão conceitual que considere conhecimento prático e teórico.
    b. Propor a utilização da representação de criatividade previamente definida como um mecanismo de adaptação para apoiar uma estrutura de agentes.

## Método

A definição do método de pesquisa se dá através de decisões em diferentes níveis hierárquicos, iniciando com a escolha da filosofia de pesquisa, estratégia de pesquisa até o nível mais operacional que envolve técnicas de coleta e análise dos dados. Saunders, (2006) ilustra tal idéia através do diagrama da "cebola de pesquisa", em que os níveis hierárquicos são representados por camadas da cebola. A seqüência de camadas, iniciando pela mais externa é: a filosofia de pesquisa, escolhas (indutiva ou dedutiva), estratégias, horizonte de tempo (transversal ou longitudinal) e técnicas e procedimentos. As decisões nos níveis mais externos (filosofia de pesquisa e estratégia) orienta as definições nos níveis subseqüentes.

Os paradigmas positivista e a fenomenológico constituem dois extremos das filosofias de pesquisa. O primeiro considera que pesquisador é externo ao mundo, tendo uma percepção imparcial e objetivo sobre o mesmo. O segundo considera que o mundo é socialmente construído, a partir da percepção e interação entre os diversas partes envolvidas. Ou seja, não é possível desassociar o pesquisador do fenômeno investigado. Entre esses dois extremos, entretanto, é possível identificar outras filosofias de pesquisas tais como a pragmática. Nessa filosofia, a validade é avaliada em função da sua utilidade e funcionamento em contexto práticos (KAZANEN, KARI; ARTO, 1993).

Após o posicionamento em relação as filosofias de pesquisa, é definida a estratégia de pesquisa, que estrutura o trabalho de pesquisa, estabelecendo a forma com que a evidência empírica vai ser coletada e analisada. Alguns exemplos de estratégias de pesquisas são: estudos de caso, pesquisa-ação, *surveys* e experimentos. Cada estratégica possui pontos fortes em relação às virtudes de uma boa teoria , assim como fragilidades.

A estratégia de pesquisa adotada no presente trabalho será a pesquisa construtiva (*constructive research* ou *design research*). Essa estratégia se caracteriza pela solução de um problema de relevância prática e teórica através de uma solução, na forma de modelos, artefatos físicos, diagramas, planos, etc. Ao invés de produzir conhecimento teórico, cientistas da pesquisa construtiva produzem e aplicam conhecimento científico de forma a criar artefatos efetivos (MARCH; SMITH, 1995).

Conforme Kazanen, Kari e Arto (1993), essa estratégia de pesquisa é utilizada em diversas áreas tais como matemática, medicina, etc. Em matemática, a elaboração algoritmos pode ser entendida como exemplo de construções. Da mesma forma, a criação de linguagens de programação em Ciência da Computação e de medicamentos ou novos tratamentos na área médica, podem ser considerados, respectivamente, como construções ou artefatos. Essa estratégia está alinhada aos pressupostos epistemológicos da filosofia de pesquisa pragmática, uma vez que a qualidade do conhecimento gerado é avaliado em função de sua utilidade.

March e Smith (1995) sugerem que o processo da pesquisa construtiva constitui-se por duas etapas fundamentais: construir (construir um artefato para um propósito específico) e avaliar (testar o funcionamento do artefato). Kasanen, Kari e Arto (1993) sugere um processo ampliando, integrando outras quatros etapas às anteriores. Na primeira etapa é identificado um problema com relevância pratica e teórica. A segunda etapa refere-se a investigação e compreensão do tema a ser trabalhado, geralmente através da revisão de literatura e estudos empíricos. A terceira etapa compreende a construção da solução, na forma de um artefato físico, modelo, etc. Tal etapa é fundamental no desenvolvimento dessa estratégia.

Segundo Kasanen, Kari e Arto (1993), caso não seja possível criar uma solução, não há sentido em prosseguir o estudo. A quarta etapa envolve a implementação e teste da solução. Na quinta etapa são apresentadas as conexões entre a solução desenvolvida e o referencial teórico, assim como a contribuição da teoria no desenvolvimento da solução. Por fim, na ultima etapa é examinando o escopo de aplicabilidade da solução.

## Desenvolvimento da Pesquisa

Seguindo a estratégia de pesquisa construtiva, organizamos nosso trabalho em quatro fases. Cada fase é constituída por etapas de pesquisa específicas bem como respectivos produtos. A seguir, cada fase é descrita em termos de suas etapas e produtos, contextualizando sua importância para a pesquisa como um todo. Durante a descrição a seguir as etapas são denominadas E.F*x*, onde E refere-se à etapa, F representa a fase e *x* uma ordenação numérica das etapas de P.

### Fase A

A primeira fase refere-se às duas etapas inicias da pesquisa construtiva (KAZANEN, KARI; ARTO, 1993): identificação de um problema de relevância prática e científica e entendimento do tema de investigação. Tais etapas serão desenvolvidas sobretudo a partir da revisão de literatura. Os principais focos de investigação através de revisão de literatura são criatividade computacional (E.A1), fusão conceitual (E.A2), representação de conhecimento e raciocínio (E.A4) e agência (E.A5). Com base nos estudos em criatividade computacional e fusão conceitual, especificamos um conjunto de requisitos resumindo a fusão em termos de suas características, constructos e processos (E.A3). Dado um entendimento de representação de conhecimento e agencia, propomos um mapeamento entre os requisitos previamente estabelecidos e possíveis formas para

representá-los computacionalmente (E.A6). Tal mapeamento é o principal resultado da fase A, servindo como um guia para o desenvolvimento da próxima fase.

**Fase B**

A segunda fase do desenvolvimento da pesquisa compreende a terceira etapa da pesquisa construtiva (KAZANEN, KARI; ARTO, 1993): inovar, construir uma solução (modelo). Nesta pesquisa, esta é a fase mais importante uma vez que é nela que nosso modelo é especificado. O desenvolvimento do modelo parte da limitação de seu escopo (E.B1). São especificadas algumas limitações devido ao amplo escopo de aplicabilidade da fusão conceitual. De acordo com Fauconnier (1998), a teoria da fusão visa explicar como os humanos integram conceitos conhecidos para construir novos conceitos. Entretanto, essa integração pode ser aplicada virtualmente a qualquer idéia que possamos conceber.

Portanto, um modelo computacional completo da fusão requer representações multidimensionais (e.g. conceitualização simbólica, percepções emocionais e sensoriais associadas, contextualização e relações episódicas) e associações diversas referentes a um único conceito. Mesmo quando consideradas sozinhas, cada uma destas dimensões constitui um tema de pesquisa em si. Desta forma, limitamos o escopo de projeto do modelo em termos de quais constructos e processos serão considerados e exatamente quais características de cada um farão parte do modelo.

A definição dos paradigmas de representação (E.B2) está diretamente ligada às limitações de escopo. Esta etapa é apoiada pelo mapeamento fornecido em E.A4, especificando em detalhe o papel de cada representação para o modelo como um todo. A seguir, é realizada a especificação do modelo – dadas as limitações e paradigma de representação – que define a integração entre fusão conceitual e uma estrutura de intencionalidade (E.B3). Nesse nível, o modelo define como a fusão é representada e como o processo de fusão lidará com as estruturas intencionais BDI (também representadas). Considerando o enfoque em adaptação de agentes, a especificação deve fornecer estruturas adicionais (frames específicos e modelos de redes conceituais) e disparadores para apoiar a adaptação. A formalização do modelo deverá ser escrita na linguagem de semântica operacional estrutural (*Structural Operational Semantics* – SOS) (PLOTKIN, 1981, 2004).

Finalmente, a especificação será implementada (E.B4) utilizando o *framework* de agentes Jason (BORDINI; WOOLDRIDGE; HüBNER, 2007). Representações de conhecimento necessárias para o processo de fusão conceitual devem ser integradas através de interfaces de programação (*Application Programming Interfaces* – API) – quando disponíveis – e, caso contrario, serão adotadas alternativas tais como a interface nativa do Java (*Java Native Interface* – JNI) ou notação de objetos Java simplificada (*Java Simplified Object Notation* – JSON).

Em conseqüência das etapas da fase B, o resultado é uma arquitetura de agentes com um mecanismo de fusão conceitual integrado. Nesse ponto, a arquitetura suportará raciocínio criativo com base na fusão conceitual. Através da especificação de como as estruturas intencionais podem ser utilizadas durante a fusão (E.B5) damos um importante passo em direção aos objetivos do trabalho. A fundamentação para alcançar o segundo objetivo intermediário também é fornecida pela arquitetura resultante da fase B.

**Fase C**

Esta fase desenvolve o quarto estágio da pesquisa construtiva: demonstrar que a solução funciona. Apesar da implementação de E.B5 já mostrar que ao menos algumas partes da solução funcionam, nós observamos a implementação como uma estrutura que precisa ser preenchida com conteúdo para fazer sentido. Portanto, na fase C nós descrevemos e implementamos dois estudos de caso para considerar os aspectos práticos da arquitetura.

O primeiro estudo de caso visa estudar como a arquitetura se comportará durante situações que exijam adaptação (E.C1). Uma vez que o mecanismo de fusão é implementado no nível arquitetural do Jason, temos a possibilidade testá-lo com agentes já desenvolvidos, com um mínimo de esforço nas configurações dos mesmos. Considerando também os mecanismos de depuração fornecidos pelo Jason, podemos observar como a fusão interfere no raciocínio prático em termos de disponibilidade de opções em momentos que a adaptação faz-se necessária. Tais situações podem ser simuladas diretamente com o depurador do Jason ou a partir do ambiente do agente, dependendo de sua implementação. Situações que requerem adaptação são consideradas sob duas perspectivas: falha na execução de intenções e falta de planos aplicáveis para lidar com determinadas configurações de mundo. Ao completarmos esse caso de uso o nosso segundo objetivo intermediário é atingido, uma vez que poderemos verificar em quais condições o mecanismo de adaptação funcionou.

A etapa seguinte desta fase utiliza a arquitetura desenvolvida para implementar um agente cognitivo para recomendação de conteúdos educacionais a partir do histórico do usuário e da especificação dos metadados dos recursos disponíveis (E.C2). Um estudo completo em sistemas de recomendação necessita de, no mínimo, uma análise estatística do sucesso das recomendações e uma comparação com algoritmos que implementem o mesmo tipo de recomendação (baseada em conteúdo). Tal estudo não faz parte deste projeto de pesquisa. Aqui, estamos mais interessados em verificar a utilização do mecanismo criativo como primitivas de linguagem (implementadas como ações internas no Jason). O contexto deste estudo de caso está associado aos resultados de pesquisa do projeto OBAA (Objetos de Aprendizagem Baseados em Agentes), financiado pela FINEP e executado pelo Grupo de Inteligência Artificial da UFRGS e UNISINOS. Nesse contexto de aplicação, o qual visa fornecer conteúdo educacional nas plataformas de TV Digital, dispositivos móveis e Web, o raciocínio criativo pode ser aplicado para surpreender o usuário ou para estabelecer relações entre conteúdos seguindo um padrão de associações diferente das abordagens tradicionais (baseadas em dedução, generalização).

Ambos estudos de caso representam uma prova de conceito (E.C3) da arquitetura, fornecendo conteúdo para que o mecanismo de criatividade possa ser testado. Logo, o produto da fase C conclui a parte de desenvolvimento deste projeto uma vez que os objetivos são analisados através dos estudos.

**Fase D**

Na última fase do projeto, apresentamos as conexões entre o modelo desenvolvido e o referencial teórico. Além disso, o escopo de aplicabilidade da solução é examinado (KAZANEN; KARI; ARTO, 1993). Desta maneira, as duas etapas da fase D propõem uma análise do modelo (E.B6) e das provas de conceito (E.C3) considerando um referencial de criatividade computacional, agentes e, finalmente, uma perspectiva integrada dos campos. A primeira etapa foca na arquitetura (E.D1), enquanto a segunda estuda aspectos levantados a partir dos estudos de caso (E.D2). Um dos produtos desta

fase são observações e uma discussão da teoria da fusão conceitual estimulada pelo desenvolvimento do modelo (E.D3). Nós acreditamos que o modelo pode contribuir especialmente no que se refere a representação e utilização da intencionalidade – considerada uma relação vital na teoria de fusão. O último produto desta pesquisa posiciona nossos resultados como etapas iniciais para uma teoria de agentes criativos (E.D4).

## Resultados

De acordo com a estrutura de nossa pesquisa, o resultado central da tese é a especificação de um modelo da fusão conceitual que define explicitamente as regras necessárias para representar uma tipologia da fusão. A partir desse modelo, realizamos dois estudos para verificar a utilização da especificação como um mecanismo de raciocínio e também como parte integrante de uma arquitetura de agentes.

A especificação da fusão conceitual foi desenvolvida utilizando a notação de semântica operacional, para definir as regras do processo, e teoria dos conjuntos, para representar abstratamente os elementos manipulados durante a fusão. O modelo foca nos princípios constitutivos e na operação geral da fusão. Portanto, as relações vitais e os princípios governantes não fazem parte deste modelo uma vez que eles requerem uma visão mais ampla de cognição, fugindo do escopo deste trabalho. Adotamos semântica operacional e conjuntos para estabelecer um modelo abstrato da fusão, possibilitando que o mesmo seja concretizado em diferentes arquiteturas e paradigmas computacionais.

Os elementos fundamentais da fusão são conceitos, espaços conceituais, esquemas organizacionais e o *blend* (resultado da fusão). Em nosso modelo, esses componentes são definidos como componentes de conjuntos e suas características individuais especificadas através de anotações em lógica descritiva estabelecendo um modelo terminológico. Os princípios constitutivos da teoria são especificados como elementos de um conjunto de operações e, para cada um deles (modificação, preenchimento e composição) foram definidas regras – em semântica operacional – que representam as respectivas manipulações conceituais realizadas.

O processo da fusão conceitual, essencialmente, ocorre através da aplicação seletiva dos princípios constitutivos em determinados conceitos de dois espaços conceituais. A seleção dos conceitos, das operações e de quais elementos constituirão o *blend* denomina-se projeção seletiva. A teoria da fusão conceitual ainda não avançou o suficiente para definir completamente como a projeção seletiva é realizada. Em nossa opinião, são necessários mais estudos e uma integração maior com a área de neurociências. Apesar disso, modelamos a projeção seletiva como duas funções de seleção dependentes de contexto (seleção conceitual e operacional). Outro aspecto que interfere na projeção seletiva são as associações entre os espaços de entrada e o espaço genérico. As associações caracterizam relações de qualquer natureza entre conceitos de diferentes espaços. Já o espaço genérico representa os itens comuns entre os espaços de entrada.

Juntamente com a projeção seletiva, é definida uma função para a condição de parada do processo e um conjunto de elementos representando a tipologia de fusão a ser adotada. Esse conjunto caracteriza a configuração do processo de fusão e, somado aos elementos previamente apresentados, estabelece uma especificação inicial para a realização da fusão. Dada a especificação inicial, são aplicadas as regras para

estabelecimento das associações entre os espaços e, em seguida, do espaço genérico (ambas definidas com semântica operacional e manipulação de elementos terminológicos).

Uma importante funcionalidade desta modelagem da fusão é a definição do processo com base na tipologia de *blends* definida pelos autores da teoria (FAUCONNIER; TURNER, 1998). Originalmente, são definidos quatro tipos de *blends*, todos eles relacionados à forma como os conceitos são manipulados. A especificação definida nesta tese representa os quatro tipos. Todo a execução da fusão parte de uma configuração onde é definido qual dos tipos que guiará o processo. Portanto, o processo em si depende da tipologia adotada e, em nosso modelo, a tipologia original foi modelada possibilitando que a fusão seja realizada segundo quatro perspectivas diferentes.

Para fornecer contextos de utilização e testar o modelo de fusão, desenvolvemos dois estudos. O primeiro deles visou utilizar a fusão conceitual como um mecanismo de adaptação para agentes BDI. Sendo assim, o modelo de fusão foi implementado computacionalmente e integrado à uma arquitetura AgentSpeak de agentes, fornecida pelo framework Jason. A integração da fusão em nível arquitetural possibilita que agentes já desenvolvidos utilizem adaptação através de fusão sem a necessidade de mudanças no código original. Além da integração, desenvolvemos um agente específico para realizar os testes e também modelamos as terminologias de domínio sob a forma de representações descritivas em linguagem OWL (*Web Ontology Language*). O motor de fusão conceitual implementado utiliza a sintaxe OWL como representação descritiva dos conceitos.

O contexto de adaptação também propõe um domínio a ser modelado em termos dos componentes da fusão conceitual, onde o *blend* resultante define uma nova linha de ação que o agente pode seguir, dada uma situação de falta de opções ou falha. Desta forma, especificamos um modelo para definir automaticamente as entradas e a configuração do processo de fusão a partir da visão de mundo atual do agente, representada por suas crenças, desejos e intenções. Essa especificação também foi implementada no framework Jason. Outros elementos necessários para a execução da fusão (funções associativas, princípios constitutivos, funções de seleção e de configuração) também foram especificados e implementados. De acordo com nossa estratégia de pesquisa, o estudo de adaptação serviu, fundamentalmente, para testar o modelo definido como parte de uma estrutura cognitiva composta por um sistema intencional apoiado por criatividade (representada pela fusão conceitual).

Nosso segundo estudo utiliza o mecanismo de fusão conceitual como um motor de raciocínio utilizado para definir o conteúdo de recomendações educacionais. Assim, o resultado concreto deste estudo é um sistema de recomendação baseado em criatividade implementado sob o paradigma de agentes. A implementação da fusão permaneceu a mesma do estudo de adaptação, entretanto, foi definida uma nova interface de acesso, possibilitando a definição de todos os elementos da fusão em tempo de execução. Para tanto, essa interface foi implementada como uma ação interna do framework Jason, podendo ser vista também como um operador de linguagem.

De uma maneira similar à adaptação, a recomendação também fornece um domínio de aplicação que foi modelado em termos dos elementos da fusão conceitual. Sendo assim, foi definida uma representação OWL do domínio, as respectivas funções de comparação e seleção. Este estudo possibilitou testar o modelo e a implementação das

funções de comparação como interfaces para ferramentas externas (uso da Wikipedia para calcular a similaridade entre palavras e do ConceptSpace para a similaridade conceitual). Ademais, testamos também o uso do mecanismo de fusão para fornecer recomendações com base em metáforas, a partir da seleção da tipologia adequada e de restrições nas entradas.

Com relação ao estudo de adaptação, este estudo diferenciou-se por utilizar a fusão conceitual como uma forma de raciocínio, deixando para o desenvolvedor a decisão de como modelar e utilizar o mecanismo. A implementação do sistema de recomendação como um agente BDI que representa seu usuário e utiliza a fusão para construir as recomendações segue nossa estratégia de pesquisa para verificar o uso da fusão conceitual como um raciocínio criativo.

## Contribuições

Considerando as questões de pesquisa deste projeto, contextualizamos nossas contribuições sob três perspectivas principais: cognição, agência inteligente e criatividade computacional. De acordo com uma abordagem *bottom-up*, primeiro posicionamos as contribuições na área de agentes inteligentes. De fato, esse posicionamento está associado à segunda questão de pesquisa deste trabalho: Como a criatividade apóia a utilização de conhecimento e experiências prévias como meios para a adaptação a situações imprevistas?

Como um processo, a criatividade nos possibilita criar novas idéias, úteis ou inúteis, reais ou surreais. Assim, essa habilidade permite que sejamos capazes de inovar, criar mais conhecimento e solucionar novos problemas. Atribuindo essa habilidade à agentes inteligentes visamos permitir que os mesmos raciocinem criativamente a partir de seus conhecimentos e percepções do ambiente. Desta forma, possibilita-se também que os agentes criem soluções (em termos de ações) alternativas para novas situações. Geralmente, abordagens para adaptação em agentes fundamentam-se em generalização e indução. A abordagem aqui proposta propõe o estudo da criatividade como um raciocínio para apoiar a adaptação.

Voltando nosso foco para a primeira questão de pesquisa "como a criatividade pode ser modelada computacionalmente e produzir conhecimento prático e teórico?", contextualizamos as contribuições para a área de criatividade computacional. Modelos computacionais de criatividade humana têm focado nos aspectos teóricos do conhecimento. Portanto, tais modelos focam na geração de conceitos teóricos sem considerar as especificidades do conhecimento prático (relacionado à ações). Comparando este projeto com os modelos atuais, o modelo aqui proposto difere-se justamente na consideração de conhecimento prático. Ademais, comparando-se com o estado-da-arte em modelos genéricos de criatividade, nosso modelo de fusão diferencia-se por especificar a tipologia da fusão definida por Fauconnier e Turner. A abordagem de um modelo abstrato e de sua concretização computacional separadamente possibilita também que nosso modelo seja integrado a outras arquiteturas.

Finalmente, em um nível mais teórico e abstrato, o modelo resultante deste projeto pode ser utilizado para estudar a teoria de fusão conceitual em si. Neste aspecto, o resultado desta pesquisa pode discute em maior detalhe a importância da intencionalidade para a fusão e também aponta a necessidade de integração com outros aspectos da cognição humana. Percebemos este trabalho como um passo inicial em

direção a um estudo mais aprofundado da cognição relacionada à criatividade através de modelos computacionais.