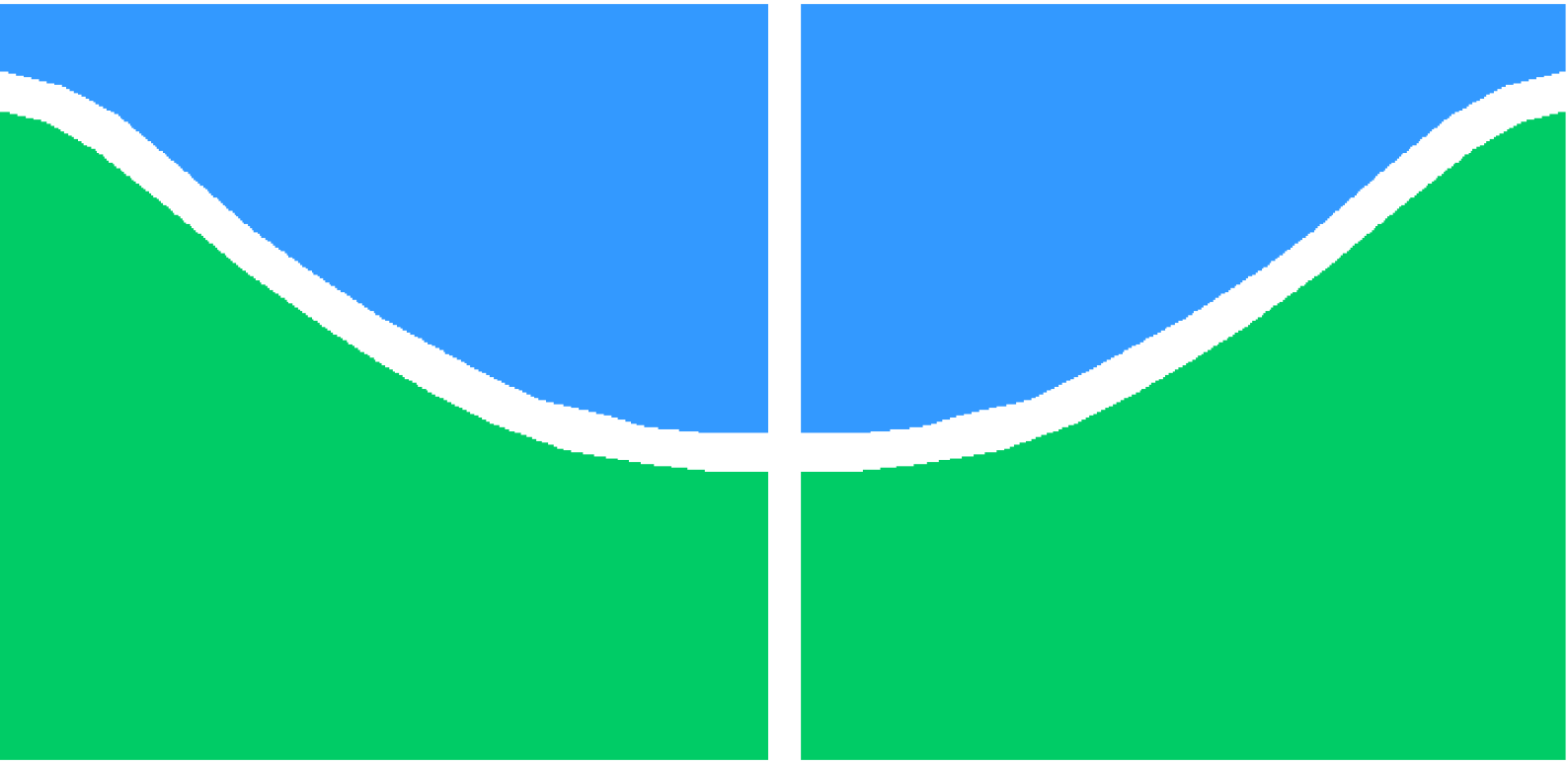

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Catálogo de Padrões Arquiteturais: Uma Visão Organizacional para Sistemas Multiagentes

Autor: Tainara Santos Reis
Orientador: Profª. Drª. Milene Serrano
Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF
2018

Tainara Santos Reis

Catálogo de Padrões Arquiteturais: Uma Visão Organizacional para Sistemas Multiagentes

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Profª. Drª. Milene Serrano

Coorientador: Prof. Dr. Maurício Serrano

Brasília, DF

2018

Tainara Santos Reis
Catálogo de Padrões Arquiteturais:
Uma Visão Organizacional para
Sistemas Multiagentes/ Tainara Santos Reis. – Brasília, DF, 2018-
131 p. : il. (algumas color.) ; 30 cm.

Orientador: Profª. Drª. Milene Serrano

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2018.

1. Sistemas Multiagentes. 2. Padrões Arquiteturais. I. Profª. Drª. Milene Serrano. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Catálogo de Padrões Arquiteturais:
Uma Visão Organizacional para
Sistemas Multiagentes

CDU 02:141:005.6

Tainara Santos Reis

Catálogo de Padrões Arquiteturais: Uma Visão Organizacional para Sistemas Multiagentes

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 04 de julho 2018:

Profª. Drª. Milene Serrano
Orientador

Prof. Dr. Maurício Serrano
Coorientador

Profª. Drª. Carla Silva Rocha Aguiar
Convidado

Brasília, DF
2018

Este trabalho é dedicado à todos aqueles que, mesmo duvidando de si mesmos, foram à luta e, portanto, venceram.

Agradecimentos

Agradeço aos meus pais, Geni Anastácia e Francisco Barbosa que me criaram para ser excelente na profissão que eu escolhesse. Agradeço ao meu irmão, Lucas Barbosa, por ouvir meus desabafos durante a graduação e por ter sido um exemplo de determinação e disciplina nos estudos. Agradeço às amigas que me acompanham há tantos anos, em especial, Cárita Gisele, Lays Cirqueira, Lorena Gianne e Marina Magalhães.

Agradeço aos meus colegas de curso por terem sido tão sensacionais e em muitos momentos, foram verdadeiros professores. Com meus colegas aprendi a importância e o significado de equipe, ética, responsabilidade e paciência. Em especial, agradeço aos amigos: Attany Araújo, Cleiton Gomes, Ebenézer Andrade, Jéssica Karine, Julliana Almeida, Keli Cristina, Lucas Rufino, Leonardo Cambraia, Martha Dantas, Marcelo Augusto, Paulo Markes, Phelipe Wener, Tâmara Barbosa, Thabata Granja e Sérgio Bezerra.

Agradeço aos meus orientadores, Milene Serrano e Maurício Serrano. Foi uma imensa alegria poder usufruir do conhecimento de ambos, sempre pacientes, dedicados, e compreensivos comigo. Agradeço a todos os professores que tornaram todo este processo a experiência mais importante da minha vida.

Agradeço, por fim, a mim mesma pela persistência e coragem; por me dedicar por incontáveis horas a este sonho; por acreditar que é possível construir algo valioso a partir da união de oportunidade, boa vontade e amor; por conseguir, mesmo diante de tantas dificuldades, finalizar esta etapa da minha vida.

Resumo

Sistemas Multiagentes (SMA) são um paradigma eficaz para projetar e desenvolver sistemas de software complexos. Entretanto, quando agentes interagem muito livremente ou de forma desorganizada, a dinâmica desses sistemas tornam-se caóticas. Uma alternativa, visando minimizar e auxiliar nessa questão, está na definição prévia de um padrão arquitetural adequado ao seu desenvolvimento. Utilizados corretamente, padrões arquiteturais podem aumentar a produtividade e a qualidade dos sistemas desenvolvidos. A disponibilização da descrição destes padrões é um instrumento para simplificar processos de análise e *design* de SMA; reduzindo o esforço associado e fornecendo suporte em âmbito técnico e estratégico no processo de desenvolvimento de software. Cabe destacar que os métodos e as técnicas de *design* organizacional podem ser utilizados como princípios arquiteturais para SMA, contribuindo para a redução da complexidade desses sistemas. Diante dessa problemática, o presente trabalho identifica, a partir da utilização da técnica de revisão sistemática, padrões arquiteturais organizacionais para SMA utilizados em diversos contextos. O principal objetivo desse trabalho consistiu na elaboração de um catálogo de padrões arquiteturais centrados no paradigma de SMA. Esse catálogo contém os padrões arquiteturais apresentados no nível de modelagem, de implementação bem como outros detalhes relevantes, permitindo apoiar o arquiteto de software no processo de escolha e na implementação de um ou mais padrões arquiteturais, usando-os como a *baseline* arquitetural/organizacional no desenvolvimento de SMA.

Palavras-chaves: Sistemas Multiagentes, Padrões Arquiteturais, Estruturas Organizacionais.

Abstract

Multiagent Systems (MAS) are an effective paradigm for designing and developing complexes software systems. Therefore, if agents interact very freely or in a disorganized and dynamic way, systems could become chaotic. One alternative, in order to minimize and assist in this matter, is in prior definition of an architectural pattern, suited to its development. Used properly, architectural standards can increase the productivity and quality of developed systems. The provision of the description of these standards and instrument to simplify processes of analysis and design of MAS; reducing associated effort and providing technical and strategic support in the software development process. It must be noted that organizational design methods and techniques can be used as a database for MAS, contributing to the reduction of system complexity. In order to contribute to solve this problem, the present work identifies, through the use of the systematic review technique, organizational architectural patterns for MAS used in different contexts. The main objective of this work was the elaboration of a catalog of architectural patterns focused on the MAS paradigm. This catalog contains the architectural standards presented at the modeling, implementation and other relevant details, assisting the software architect in the selection and implementation process of one or more architectural standards, using it as the architectural/organizational baseline in the development of MAS.

Key-words: Multiagent systems, Architectural patterns, Organizational Structures.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ACL	*Agent Communication Language*
AID	*Agent Identifier*
AMS	*Agent Management System*
DF	*Directory Facilitator*
FIPA	*Foundation for Intelligent Physical Agents*
JADE	*Java Agent DEvelopment Framework*
RS	Revisão Sistemática
SMA	Sistemas Multiagentes
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
UnB	Universidade de Brasília
MACRON	*Multiagent Architecture for Cooperative Retrieval ONline*

Sumário

1 Introdução

Este capítulo tem como objetivo contextualizar a pesquisa (Seção 1.1), apresentar a questão de pesquisa investigada, justificar a pertinência do estudo no tema (Seção 1.2), descrever os objetivos desse trabalho (Seção 1.3) e, por fim, apresentar a organização geral dessa monografia (Seção 1.4).

1.1 Contextualização

A utilização de agentes de software constitui uma abordagem adequada para a concepção e a construção de sistemas complexos e distribuídos (JUNIOR et al., 2003). Tais agentes conhecem os interesses dos usuários e podem agir de forma autônoma em prol dos mesmos. Ao invés de exercer controle completo, o papel dos usuários dá-se, preferencialmente, de forma cooperativa com os agentes de software, de modo que esses últimos possam auxiliar esses usuários no cumprimento de suas metas (GREEN et al., 1997).

Agentes de software despertam o interesse de diversos pesquisadores e empresas de software (GREEN et al., 1997). Trata-se de uma área que está emergindo rapidamente dentro da TI (JENNINGS; WOOLDRIDGE, 1996), como um paradigma poderoso para projetar e desenvolver sistemas de software complexos (ZAMBONELLI; JENNINGS; WOOLDRIDGE, 2001).

Sistemas Multiagentes (SMA) encontram espaço em uma série de domínios de aplicação, como: comércio eletrônico, *design* de interface, jogos e gestão de processos industriais e comerciais complexos (JENNINGS; WOOLDRIDGE, 1996). Green et al. (1997) apontam outras áreas de aplicação como controle de tráfego aéreo, *data mining*, recuperação e gestão da informação, educação e assistentes digitais pessoais (*Personal Digital Assistants*, PDAs).

Uma das armadilhas apontadas por Wooldridge (2009, pág. 246) ocorre quando agentes interagem muito livremente ou de forma desorganizada. A dinâmica de SMA é complexa e, portanto, pode se tornar caótica. Para descobrir o que acontecerá em seguida no sistema, muitas vezes, é necessário executar o sistema repetidamente. O número de agentes também influencia em termos de complexidade para gerir de forma eficaz o sistema.

Independentemente do paradigma adotado, é habitual que desenvolvedores comecem a codificar sem uma arquitetura formal clara e bem definida (RICHARDS, 2015). O resultado desta prática, frequentemente, é uma coleção de módulos de código-fonte

desorganizados, que não possuem papéis, responsabilidades e relacionamentos claros uns com os outros.

Segundo Richards (2015), aplicações que não possuem uma arquitetura formal são geralmente "acopladas, quebradiças, difíceis de mudar e sem uma visão ou direção clara". Questões básicas sobre implantação, manutenção e escalabilidade desses sistemas são difíceis de definir.

Uma alternativa é adotar padrões arquiteturais. Estes ajudam a definir as características básicas e o comportamento de uma aplicação (RICHARDS, 2015). O arquiteto de software deve conhecer as características, pontos fortes e fracos de cada padrão arquitetural, afim de optar por aquele que atenda às suas necessidades e aos seus objetivos comerciais específicos.

De acordo com Shaw (1996), descrições uniformes das arquiteturas, quando disponíveis, são instrumentos para simplificar esta escolha. Segundo Avgeriou e Zdun (2005), o processo de descrever, encontrar e aplicar padrões arquiteturais na prática ainda é largamente *ad-hoc* e não sistemático. Isto se deve a várias questões que ainda não foram resolvidas como, por exemplo, em relação à classificação ou catalogação de padrões que podem ser usados por arquitetos de software.

Cabe destacar que existe um paralelo entre a complexidade das organizações e os SMA. Organizações são entidades complexas formadas para superar várias limitações de instâncias individuais, tais como limitações cognitivas, físicas, temporais e institucionais (AART, 2004). Neste sentido, são utilizados conceitos, métodos e técnicas de *design* organizacional humano como princípios arquitetônicos para SMA (AART, 2004).

Uma organização fornece uma estrutura para as interações dos agentes através da definição de papéis, expectativas de comportamento e relações de autoridade (SYCARA, 1998). As organizações são, em geral, conceitualizadas em termos de sua estrutura, isto é, o padrão de informações e relações de controle que existem entre os agentes e a distribuição das capacidades de resolução de problemas entre eles (SYCARA, 1998).

Na resolução cooperativa de problemas, por exemplo, uma estrutura fornece a cada agente uma visão de alto nível de como o grupo soluciona problemas (SYCARA, 1998). Portanto, a estrutura organizacional impõe restrições sobre as formas como os agentes se comunicam e se coordenam. Neste contexto, os padrões arquiteturais para SMA baseados em estruturas organizacionais, podem fornecer soluções práticas para o desenvolvimento de SMA complexos, sendo assim, são objetos de estudo deste trabalho.

1.2 Justificativa

A definição prévia de um padrão arquitetural para o desenvolvimento de um sistema é relevante no desenvolvimento de software, como evidenciado na Seção anterior. Se usados corretamente, padrões arquiteturais podem aumentar a produtividade e a qualidade das aplicações desenvolvidas (JUNIOR et al., 2003).

Em vista disso, uma das formas de estruturar um SMA está em reduzir a complexidade do mesmo. Por conseguinte, a eficiência do sistema aumenta, bem como a solução do problema o qual o sistema busca resolver passa a ser mais precisa (WOOLDRIDGE, 2009, pág. 236).

Muitas arquiteturas para SMA têm sido propostas como comprovam os trabalhos de Kolp, Giorgini e Mylopoulos (2006) e Junior et al. (2003). Entretanto, ainda existem poucos estudos que reúnam tais propostas de forma voltada para a catalogação de padrões arquiteturais em nível organizacional de abstração.

Constatadas essas dificuldades, este trabalho procurou responder a seguinte questão: quais são os principais padrões arquiteturais em nível de estruturas organizacionais para SMA? Os padrões arquiteturais acordados nesse TCC são identificados, como também possuem modelagem e exemplos de cunho didático e prático. Portanto, podem servir como referências nos desenvolvimentos de SMA, da análise, ao design e à implementação desses.

Considerando esse panorama, a elaboração de um catálogo capaz de apoiar o arquiteto de software nas tarefas de escolha, modelagem e implementação de um padrão arquitetural organizacional para SMA, reduzindo o esforço e, provavelmente, até os custos associados, torna-se de uma valia considerável, tanto no âmbito técnico quanto estratégico no processo de desenvolvimento de SMA.

1.3 Objetivos

O objetivo geral deste trabalho foi definir um catálogo de padrões arquiteturais para SMA que se enquadram como estruturas organizacionais.

Visando atingir o objetivo geral, foram estabelecidos alguns objetivos específicos, os quais são apresentados a seguir:

1. Identificar modelos arquiteturais de SMA candidatos a padrões arquiteturais;
2. Selecionar modelos arquiteturais que se caracterizam como padrões arquiteturais;
3. Catalogar os padrões arquiteturais.

1.4 Organização dos capítulos

Esta monografia está organizada em sete capítulos, descritos brevemente a seguir:

- **Capítulo 1 - Introdução:** capítulo atual, no qual são apresentados o contexto, a justificativa, a questão de pesquisa bem como os objetivos geral e específicos;
- **Capítulo 2 - Referencial teórico:** descreve os conceitos associados à Engenharia de Software e ao domínio de interesse desse trabalho, no caso, Sistemas Multiagentes;
- **Capítulo 3 - Suporte tecnológico:** apresenta as ferramentas que suportaram as atividades de desenvolvimento de software, gerenciamento, documentação, dentre outras;
- **Capítulo 4 - Metodologia:** estabelece os procedimentos seguidos do início até a conclusão do trabalho;
- **Capítulo 5 - Proposta:** destinado à apresentação da proposta do trabalho como um todo, definindo a estrutura do catálogo de padrões arquiteturais, procedimentos de seleção de padrões de projeto e categorização dos mesmos;
- **Capítulo 6 - Resultados finais:** apresenta os resultados alcançados, incluindo os ciclos de pesquisa-ação realizados bem como a versão final do catálogo em detalhes, e
- **Capítulo 7 - Conclusão:** apresenta a conclusão a cerca dos resultados alcançados, bem como os possíveis trabalhos futuros.

2 Referencial teórico

Neste capítulo, são apresentadas as bases teóricas para a elaboração do catálogo, visando facilitar o entendimento dos termos utilizados. O capítulo está estruturado em seções. Nas seções 2.1 e 2.2 são descritos os conceitos de agentes e SMA, respectivamente. Nas seções 2.3 e 2.4, aborda-se uma temática mais específica de arquitetura de software bem como são apresentados os padrões arquiteturais.

Na Seção 2.5, estruturas organizacionais são apontadas como princípios arquiteturais que contribuem para a definição de padrões arquiteturais para SMA. As seções 2.6 e 2.7 trazem a definição de padrão de projeto e catálogo de padrões bem como a motivação para fazê-los e suas estruturas.

2.1 Agentes

Existem diferentes definições para agentes. Ainda assim, estas definições possuem conceitos básicos em comum como sugere McArthur et al. (2007): a noção de agente, seu ambiente e sua autonomia. Para o autor e para Jennings e Wooldridge (1996), uma definição flexível para agentes é que são entidades de software ou hardware capazes de reagir e tomar decisões de forma autônoma, com base nas alterações do ambiente o qual estão situados. Selker (1994) destaca que agentes são software que simulam uma relação humana, realizando tarefas que poderiam ser executadas por pessoas.

Agentes podem ser representados por seis características ortogonais, conforme ilustrado na Figura 1: adaptabilidade, autonomia, inteligência, mobilidade, persistência e cooperação. Tais características, atuando em conjunto, tornam o agente mais robusto e suscetível a mudanças (GRISS, 2001). Quanto maior for a área fechada no diagrama, mais aquele componente do sistema pode ser classificado como agente (GRISS, 2001).

Um agente pode ter seu comportamento alterado após sua implantação. O grau com que isto é possível denomina-se *adaptabilidade*. A *autonomia* de um agente, por sua vez, é definida como o grau com que é responsável pelo controle da sua própria *thread* e o grau com que pode prosseguir em sua meta individual independentemente de mensagens enviadas de outros agentes (GRISS, 2001).

O agente também se caracteriza por sua capacidade de raciocinar sobre suas metas e seus conhecimentos. O grau com que o agente possui esta capacidade denomina-se por *inteligência*. Conjuntamente, agentes possuem *mobilidade*, que se trata da habilidade de um agente passar da execução de contexto para outro. Ou seja, o agente pode continuar sua execução em um novo contexto enquanto mantém seu estado (GRISS, 2001).

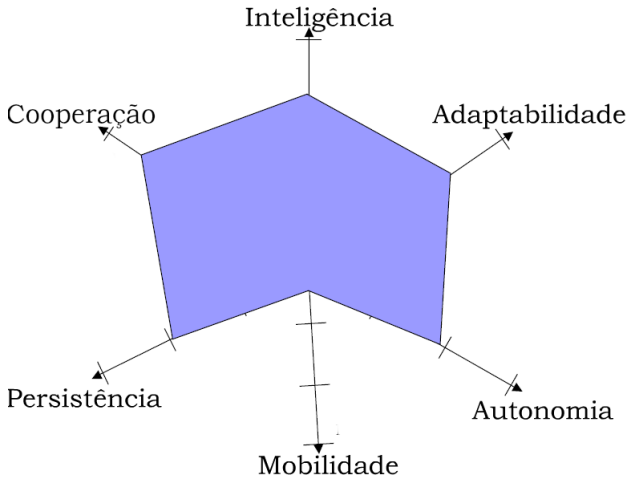

Figura 1 – Dimensões do agente. Fonte: Griss (2001). Traduzido.

A infra-estrutura oferecida aos agentes pode permitir que conservem tanto conhecimentos adquiridos quanto seu estado durante longos períodos de tempo. Pode contribuir também para a robustez dos agentes diante de possíveis falhas em tempo de execução. O grau com que a infra-estrutura cumpre estes critérios intitula-se *persistência* (GRISS, 2001).

Ademais, agentes possuem *cooperação* como um de seus atributos. Trata-se do grau em que agentes se comunicam e trabalham em cooperação uns com os outros para formar Sistemas Multiagentes (GRISS, 2001). Em sentido semelhante, Wooldridge e Jennings (1995) destacam outros atributos como habilidade social - caracterizada pela interação entre agentes através de linguagem de programação voltada para agentes, reatividade - diz respeito à responder em tempo hábil às mudanças que ocorrem no ambiente percebido pelo agente, e, por fim, proatividade - agentes possuem comportamento dirigido à meta, o que significa que possuem a habilidade de tomar iniciativas por conta própria.

2.2 Sistemas multiagentes

Sistemas Multiagentes (SMA) enquadram-se no cenário, onde coexistem dois ou mais agentes (MCARTHUR et al., 2007) operando juntos de modo a realizar um conjunto de tarefas para satisfazer um conjunto de metas (LESSER, 1999). Tais agentes comunicam-se tipicamente com troca de mensagens através da infra-estrutura de rede fornecida (WOOLDRIDGE, 2009, pág. 3).

Lesser (1999) aponta que agentes podem ser caracterizados como benevolentes/-cooperativos ou auto-interessados (*self-interested*). Agentes cooperativos trabalham para alcançar objetivos comuns, enquanto agentes auto-interessados possuem objetivos distintos do objetivo geral, mas ainda interagem para alcançar suas metas pessoais. Neste último caso, os agentes auto-interessados podem, através de troca de favores, coordenar-se com

outros agentes, a fim de que estes realizem alguma atividade para ajudar na conquista de seus próprios objetivos.

Segundo Wooldridge (2009), no caso mais geral, os agentes em um SMA representam ou atuam em nome dos usuários ou proprietários com diferentes objetivos e motivações. Para que interajam com sucesso, tais agentes exigem a capacidade de cooperar, coordenar e negociar uns com os outros, da mesma forma que as pessoas podem cooperar, coordenar e negociar com as outras na vida cotidiana (WOOLDRIDGE, 2009, pág. 3).

Como estabelece Serrano (2011 apud JENNINGS, 2000, pág. 42-43), sistemas multiagentes possuem basicamente seis elementos: (i) agentes, (ii) interação, (iii) organização, (iv) recursos, (v) esfera de influência e (vi) ambiente (Figura 2). A *interação* pode ocorrer entre os agentes e *ambiente*, bem como entre os agentes por meio de protocolos específicos. A *organização* situa-se dentro do ambiente, determinando quais tarefas serão executadas pelos agentes de modo a satisfazer os usuários.

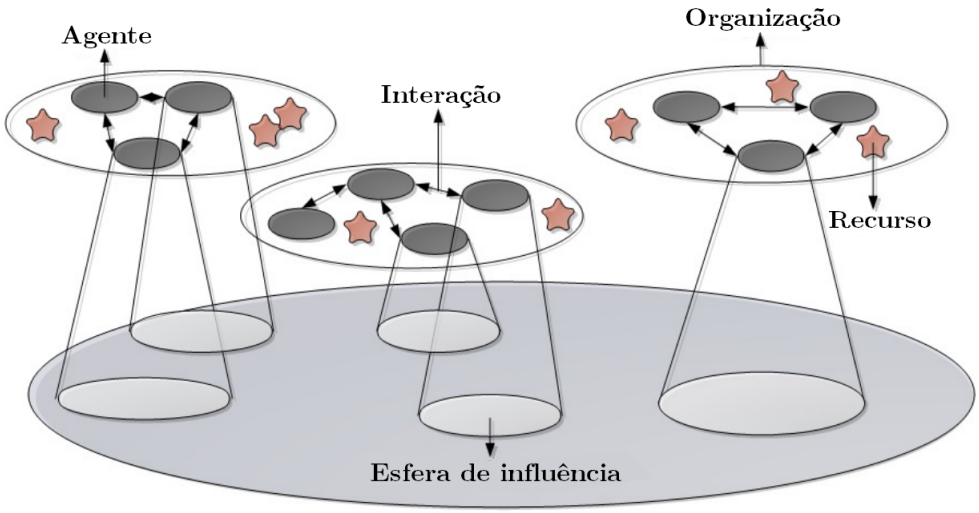

Figura 2 – Estrutura de SMA. Fonte: Jennings (2000 apud SERRANO, 2011) . Traduzido.

Para que concluam suas tarefas, os agentes e a organização podem usufruir dos *recursos* presentes no ambiente. Serrano (2011 apud JENNINGS, 2000, pág. 42-43) explica que cada elemento contém uma *esfera de influência* no ambiente em análise. Esta esfera representa a influência de cada elemento do ambiente, de modo a evidenciar sua importância para que os objetivos sejam alcançados com êxito.

2.3 Arquitetura de software

A decomposição de alto nível de um sistema em componentes principais, juntamente com uma caracterização de como esses componentes interagem, é chamada de arquitetura de software (VLIET; VLIET, 2007, pág. 279). Trata-se da descrição dos subsis-

temas e componentes de um sistema de software e as relações entre eles (BUSCHMANN; HENNEY; SCHMIDT, 2007).

Tais subsistemas e componentes são muitas vezes especificados por meio de diferentes pontos de vista para mostrar as propriedades funcionais e não funcionais relevantes de um sistema de software (BUSCHMANN; HENNEY; SCHMIDT, 2007). A arquitetura de um sistema de software, portanto, é uma estrutura em larga escala de sistemas de software que reflete as decisões iniciais e essenciais do projeto (VLIET; VLIET, 2007).

Vale ressaltar que arquitetura é uma abstração transferível de um sistema, ou seja, é uma base para a reutilização (VLIET; VLIET, 2007, pág. 280). Assim, abstrações de mais alto nível podem classificar-se em padrão arquitetural (VLIET; VLIET, 2007, pág. 282).

2.4 Padrão arquitetural

Avgeriou e Zdun (2005) definem padrões arquiteturais como soluções bem estabelecidas para problemas arquiteturais. Tais soluções ajudam a documentar as decisões de projeto de arquitetura bem como descrevem atributos de qualidade do sistema. Padrões arquiteturais facilitam a comunicação entre as partes interessadas - *stakeholders* - através de um vocabulário comum.

Padrões arquiteturais são considerados como pares problema-solução que ocorrem em um determinado contexto e se afetam por ele (AVGERIOU; ZDUN, 2005). Além disso, um padrão não apenas indica como uma solução resolve um problema, mas também torna explícito o "porquê"do problema ser resolvido; ou seja, qual a lógica por trás desta solução em particular (AVGERIOU; ZDUN, 2005).

A descrição dos padrões arquiteturais baseia-se no tripé problema-solução-contexto (AVGERIOU; ZDUN, 2005). Pode também ser ainda mais elaborado com maiores detalhes, como por exemplo, a lógica por trás da solução. Em particular, a descrição do problema foca nas causas que moldam o problema, enquanto a solução detalha como e se essas causas são resolvidas (AVGERIOU; ZDUN, 2005).

Há uma série de postulados para uma solução para que se qualifique como padrão. Por exemplo, este deve capturar a prática comum, ou seja, possuir pelo menos três utilizações conhecidas e, ao mesmo tempo, a solução deve ser não-óbvia (AVGERIOU; ZDUN, 2005).

Padrões podem ser usados em combinação, conforme explicam Shaw (1996), Avgeriou e Zdun (2005). Podem trabalhar em conjunto fornecendo pontos de vista complementares durante o projeto inicial ou elaborando um determinado componente de um padrão usando algum outro padrão (SHAW, 1996). Portanto, é possível que haja numerosas in-

terdependências entre padrões (AVGERIOU; ZDUN, 2005). Estas combinações podem ocorrer repetidamente, até que questões de arquitetura sejam resolvidas, dando lugar às técnicas de programação convencionais (SHAW, 1996).

No trabalho de Shaw (1996, pág. 5), a descrição de cada padrão inclui notas sobre:

- Problema: o problema em que o padrão busca solucionar;
- Contexto: aspectos da configuração, como o ambiente computacional, que restringem o uso do padrão;
- Solução: o modelo de sistema capturado pelo padrão, os componentes, os conectores e as estruturas de controle que compõem o padrão;
- Diagrama: uma figura, em notação apropriada, que mostra um padrão típico;
- Variantes significativas: notas sobre possíveis variações do padrão, e
- Exemplos: referências a exemplos ou *overviews* de sistemas, nos quais se aplicam o padrão.

2.5 Estruturas organizacionais

Teorias voltadas às estruturas sociais contribuem para os conceitos fundamentais de SMA como intencionalidade e sociabilidade (KOLP; GIORGINI; MYLOPOULOS, 2006). Dentre essas teorias, os pesquisadores voltaram-se para estruturas sociais que resultam de um processo de *design*: as teorias organizacionais nomeadas *Organization Theory* - que descrevem a estrutura e o *design* de uma organização - e *Strategic Alliances* - que modelam colaborações estratégicas de *stakeholders* organizacionais independentes.

De acordo com Aart (2004), as organizações são entidades complexas formadas para superar várias limitações de instâncias individuais como, por exemplo, limitações físicas, temporais, cognitivas e institucionais. Existe um paralelo entre a complexidade das organizações e SMA cabendo, portanto, a utilização de conceitos, métodos e técnicas de *design* organizacional humano como princípios arquitetônicos para SMA (AART, 2004).

Zambonelli, Jennings e Wooldridge (2001) indica que uma das vantagens em adotar uma perspectiva organizacional é tornar o projeto do sistema menos complexo e mais fácil de gerenciar do que abstrações mais tradicionais permitem.

Cada agente torna-se um local separado de controle, encarregado de cumprir seu papel e ser plenamente responsável por ele; incorporando, assim, a maior parte da funcionalidade de que necessita para cumprir o seu papel. Quando tomados em conjunto, esses pontos facilitam o processo de *design* por promover uma separação mais evidente

entre as dimensões de *design* intra-agente e inter-agente (ZAMBONELLI; JENNINGS; WOOLDRIDGE, 2001).

Em muitos casos, os SMA são destinados a apoiar e/ou controlar alguma organização do mundo real. Por exemplo, os SMA podem ser adotados para suportar o gerenciamento do fluxo de trabalho em uma equipe ou para ajudar a controlar as atividades de um leilão na internet. Nesses casos, um projeto de SMA baseado em organização reduz a distância conceitual entre o sistema de software e o sistema do mundo real que ele tem de suportar. Consequentemente, isto simplifica o desenvolvimento do sistema (ZAMBONELLI; JENNINGS; WOOLDRIDGE, 2001).

2.6 Padrão de projeto

Um padrão de projeto (ou *design pattern*) possui quatro elementos essenciais (JOHNSON et al., 1995):

- O nome do padrão: descreve o padrão que resolve determinado problema bem como facilita a documentação e a discussão sobre padrões pelos projetistas do sistema;
- O problema: descreve quando aplicar o padrão;
- A solução: descreve os elementos que compõem o *design*, as suas relações, responsabilidades e colaborações, e
- As consequências: os resultados, vantagens e desvantagens de aplicar o padrão.

O conceito de padrões de projeto refere-se às descrições de objetos e classes no catálogo de padrões apresentado por Gamma (1995). Um padrão de projeto define uma estrutura de projeto comum que o torne um projeto orientado a objetos reutilizável. Esta estrutura é composta por instâncias e classes com seus papéis, colaborações e a divisão de responsabilidades (GAMMA, 1995).

A fim de exemplificar um padrão de projeto, é apresentado a seguir, de forma resumida, o padrão *Abstract Factory* definido por Gamma (1995), com alguns dos elementos originalmente utilizados para sua descrição.

Nome do padrão: *Abstract Factory.*

Intenção: provê uma interface para a criação de famílias de objetos relacionados ou dependentes entre si, sem especificar suas classes concretas.

Estrutura: ilustrada na Figura 3.

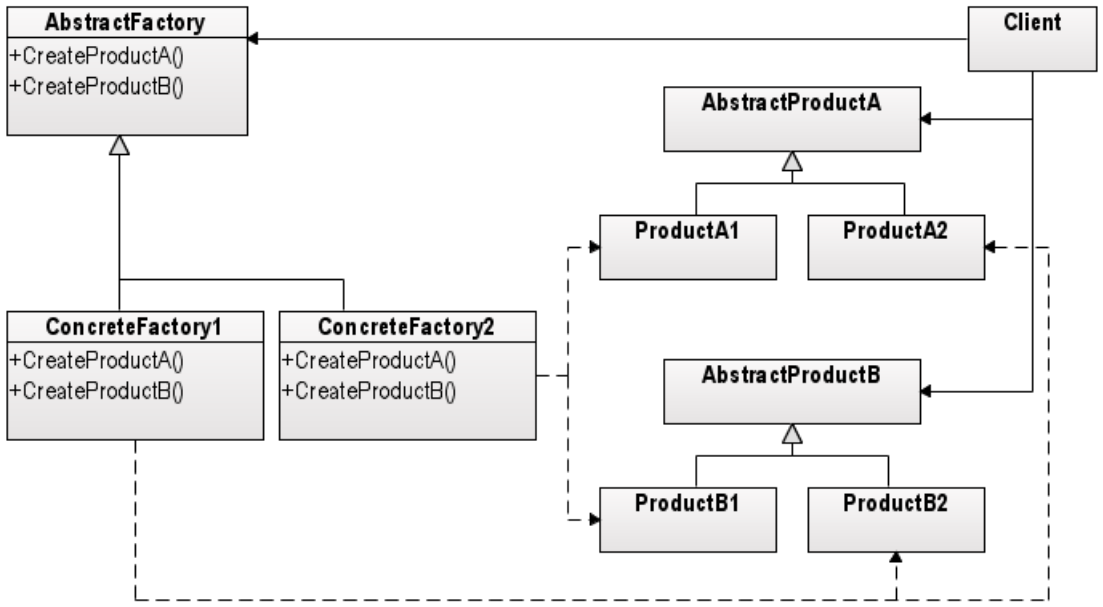

Figura 3 – Estrutura do padrão de projeto *Abstract Factory*. Fonte: Gamma (1995).

Consequências: isola classes concretas; facilita a troca de famílias de produtos; promove a consistência entre os produtos; dificulta a implementação de novos tipos de produtos.

Implementação: neste tópico, são apresentadas e descritas técnicas para implementar o padrão, basicamente: fábricas como *Singletons*, criação dos produtos e definição de fábricas extensíveis.

Exemplo de código: neste tópico, são inseridos trechos de códigos de modo a apoiar a explicação de como implementar o padrão, por exemplo: como instanciar uma *ConcreteFactory* a partir da *AbstractFactory* e, de modo semelhante, como instanciar um *Product* a partir de uma classe *AbstractProduct* utilizando as técnicas supracitadas.

Usos conhecidos: uso na criação de objetos de interface e uso no desenvolvimento de portabilidade entre sistemas *Windows*.

Padrões relacionados: o *Abstract Factory* pode ser implementado utilizando-se *Factory Method* ou *Prototype*. Uma *ConcreteFactory* geralmente é um *Singleton*.

Nesta pesquisa, de modo semelhante, o catálogo de padrões a ser apresentado busca reunir arquiteturas organizacionais exclusivamente para SMA e que propiciem a reutilização.

2.7 Catálogo de padrões

Um catálogo de padrões é um conjunto de padrões relacionados; por vezes vagamente ou informalmente relacionados. No catálogo, os padrões são subdivididos em categorias abrangentes, podendo incluir referências cruzadas entre os padrões (APPLETON, 1997). O catálogo de padrões pode incluir a apresentação das estruturas e organização do padrão, mas, normalmente, não apresenta algo além da estrutura e das relações mais facilmente identificáveis (APPLETON, 1997).

A exemplo de Gamma (1995), foram organizados vinte e três padrões de projeto orientados a objetos em um catálogo, como ilustra a Tabela 1. Este catálogo descreve boas soluções de software de uso genérico, ou seja, padrões que independem do domínio de aplicação. Os autores utilizaram dois critérios para classificar os padrões de projeto: (i) escopo e (ii) propósito. Assim, pôde-se comunicar e exibir as relações entre os padrões.

Tabela 1 – Classificação dos padrões de projeto GOF. Fonte: Gamma (1995).

		Propósito		
		Criacional	**Estrutural**	**Comportamental**
Escopo	**Classe**	*Factory Method*	*Adapter*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Builder Prototype* *Singleton*	*Adapter* *Bridge* *Composite* *Decorator* *Facade* *Flyweyght* *Proxy*	*Chain of Responsability* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

Portanto, em resumo, para caracterizar um catálogo de padrões, deve-se fornecer:

- Categorias de padrões;
- Critérios de classificação do padrão em categorias;
- Relacionamentos/referências cruzadas entre padrões.

2.8 Considerações parciais

Conceitos relacionados à Engenharia de Software como padrão de projeto, catálogo de padrões, arquitetura de software e padrão arquitetural são abordados neste capítulo

por serem conceitos frequentemente utilizados neste trabalho. Em suma, os principais tópicos acerca de SMA foram abordados de modo a servirem de embasamento para os próximos capítulos, auxiliar no entendimento do catálogo de padrões e de descrever aspectos importantes para sua elaboração. Dadas as possíveis definições de agentes e SMA, optou-se por agregar definições didáticas e concordantes entre si; bem como, a escolha dos autores dos trabalhos citados deu-se pela alta popularidade de seus trabalhos.

3 Suporte tecnológico

Este capítulo tem o objetivo de apresentar as principais ferramentas que foram utilizadas no desenvolvimento da monografia e na execução da pesquisa. Este capítulo está dividido em duas seções principais: a Seção 3.1 apresenta as ferramentas relacionadas a SMA, e a Seção 3.2 apresenta as ferramentas de apoio à elaboração do trabalho em geral.

3.1 Sistemas Multiagentes

Quanto ao desenvolvimento de SMA, foi adotada a plataforma JADE, descrita a seguir.

3.1.1 Plataforma JADE

O JADE[1] (*Java Agent DEvelopment Framework*) é uma ferramenta desenvolvida pela Telecom Itália em parceria com a Universidade de Parma, o qual, atualmente, é um projeto *open source* com licença LGPL (*Lesser General Public Licence*) (JADE, 2017). Trata-se de uma plataforma de software que fornece funcionalidades básicas de camada *middleware* (JADE, 2017). A versão utilizada foi a 4.5.0.

O JADE simplifica a abstração de agentes de software em contextos de aplicações distribuídas por meio da implementação de grande parte das especificações FIPA (BELLIFEMINE; CAIRE; GREENWOOD, 2007, pág. 30). O JADE contribuiu com a popularização das especificações FIPA ao fornecer um conjunto de ferramentas e abstrações de software que ocultam a implementação e as especificações dos programadores. Assim, pode-se implementar de acordo com as especificações, sem a necessidade de estudá-las (BELLIFEMINE; CAIRE; GREENWOOD, 2007, pág. 30).

Outro mérito da plataforma JADE é que ela implementa tais abstrações através da linguagem Java e, portanto, provê uma uma API simples e amigável; além de se tratar de uma linguagem orientada a objetos conhecida e multiplataforma (BELLIFEMINE; CAIRE; GREENWOOD, 2007, pág. 30).

Em suma, o JADE inclui (CAIRE, 2009):

- Um ambiente em tempo de execução para os agentes JADE, o qual deve estar ativo em um determinado *host* antes que um ou mais agentes possam ser executados nesse *host*;

[1] http://jade.tilab.com (último acesso: Junho 2017)

- Uma biblioteca de classes para implementar agentes, e
- Um conjunto de ferramentas gráficas que permite administrar e monitorar as atividades dos agentes em execução.

3.1.1.1 Arquitetura da plataforma JADE

Cada instância em execução do ambiente JADE é chamada de *Container* - ou contêiner-, pois pode conter vários agentes. O conjunto de contêineres ativos é chamado de *Platform*. Um contêiner se difere dos demais: o *Main container*. Este deve estar sempre ativo e ser o primeiro a iniciar-se na plataforma, de modo que todos os contêineres possam registrarem-se nele assim que se iniciam (CAIRE, 2009).

Caso outro *Main container* seja iniciado na rede, será de uma plataforma diferente para a qual novos contêineres normais podem registrar-se. Para ilustrar os conceitos supracitados, a Figura 4 apresenta um cenário de exemplo composto por duas plataformas JADE formada por três e um contêineres, respectivamente (CAIRE, 2009).

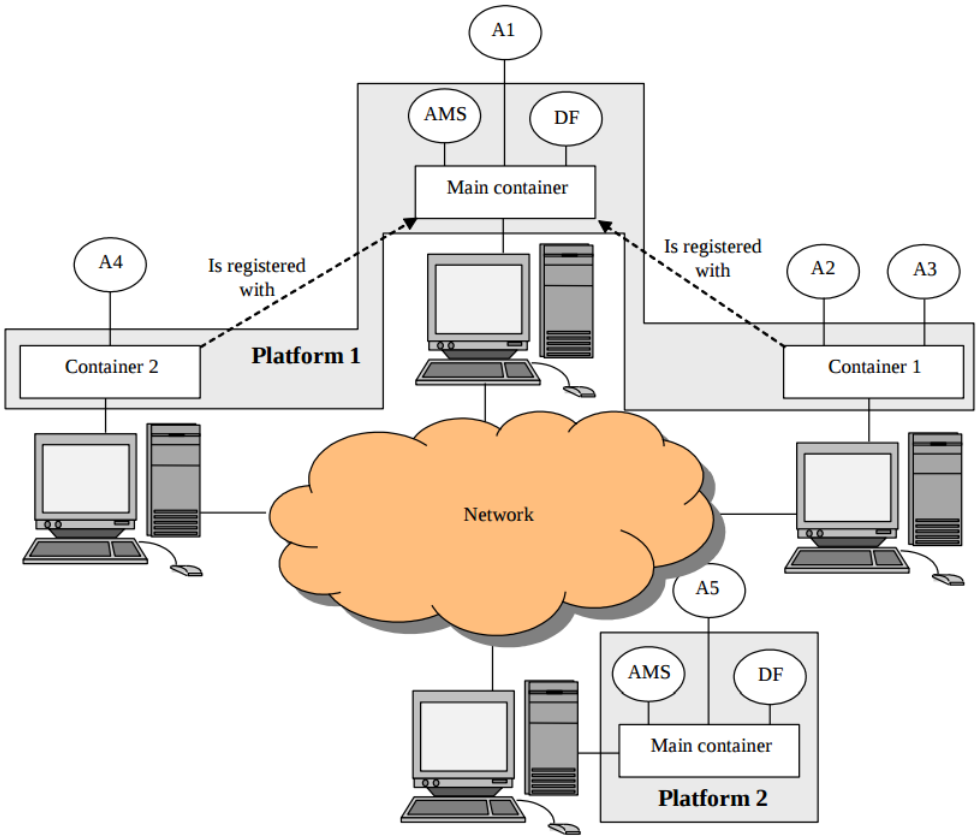

Figura 4 – Containers e plataformas. Fonte: Caire (2009).

Os agentes JADE são identificados com nomes exclusivos e, desde que conheçam o nome do outro, eles podem se comunicar independentemente da sua localização real, como, por exemplo, na Figura 4: agentes A2 e A3 no mesmo contêiner, agentes A1 e A2

em diferentes contêineres, mas na mesma plataforma e, agentes A4 e A5 em diferentes plataformas (CAIRE, 2009).

Além da capacidade de aceitar registros de outros contêineres, um *Main container* difere dos contêineres normais por possuir dois agentes especiais que se iniciam automaticamente: o AMS (*Agent Management System*) e o DF (*Directory Facilitator*) (CAIRE, 2009).

O AMS exerce controle de supervisão sobre o acesso e uso. Fornece o serviço de nomeação que garante que cada agente na plataforma registre-se com um nome exclusivo e válido, o AID (Agent Identifier) (BELLIFEMINE et al., 2002). Além disso, o AMS é capaz de controlar o ciclo de vida dos agentes em contêineres remotos (CAIRE, 2009).

Por sua vez, o agente DF fornece um serviço de *Yellow pages* - ou páginas amarelas - através do qual um agente pode encontrar outros agentes que fornecem os serviços que ele precisa para alcançar seus objetivos (CAIRE, 2009).

3.2 Ferramentas de apoio

As ferramentas a seguir foram utilizadas para auxiliar no gerenciamento e na implementação, bem como na documentação associada. Buscou-se aplicar os conhecimentos adquiridos no decorrer do curso de Engenharia de Software para apoiar o processo, tanto em tecnologia, quanto em práticas de desenvolvimento de software.

3.2.1 Bonita BPM

O Bonita BPM[2] é uma ferramenta *open-source* para gerenciamento de processos de negócios (BONITASOFT, 2016). Ela permite modelar, configurar e executar fluxos de trabalho de negócios utilizando a notação BPMN (*Business Process Management Notation*). No contexto desta monografia, foi utilizada para modelar processos relacionados à metodologia (Capítulo 4). A versão utilizada foi a 7.3.

3.2.2 Trello

A ferramento Trello[3] possibilita de modo fácil, gratuito e flexível o gerenciamento de projetos (TRELLO, 2017). O Trello auxilia no ganho de produtividade tanto por equipes grandes quanto de forma individual. Baseia-se no sistema de kanban[4], bastante utilizado no desenvolvimento com Scrum, metodologia utilizada neste traballho (TRELLO, 2017). O Trello foi utilizado para organizar as tarefas envolvidas durante a pesquisa, desenvolvimento e escrita deste trabalho.

[2] http://www.bonitasoft.com (último acesso: Junho 2017)
[3] https://trello.com (último acesso: Junho 2017)
[4] https://br.atlassian.com/agile/kanban (último acesso: Junho 2017)

3.2.3 Debian

O Debian[5] é um sistema operacional livre e *open-source* desenvolvido pelo Projeto Debian (DEBIAN, 2016). Os sistemas desenvolvidos pelo projeto utilizam, atualmente, o *kernel Linux* ou o *kernel FreeBSD*. O Debian 8 Jessie, licenciado pela *General License Public* (GPL), foi utilizado neste trabalho para o desenvolvimento de software.

3.2.4 Eclipse IDE

O Eclipse[6] é uma IDE utilizada, principalmente, para o desenvolvimento de aplicações Java (ECLIPSE FOUNDATION, 2016). Está licenciado sob uma licensa *open-source*, a *Eclipse Public License*. A versão 4.6.3 é utilizada neste trabalho.

3.2.5 Git

Git[7] é uma ferramenta para controle de versão distribuída sob a licença *GNU General Public License version 2.0*, uma licença *open source*. Traz benefícios como velocidade, garantia da integridade dos dados e suporte para fluxos de trabalho distribuídos e não-lineares (CHACON; STRAUB, 2014, pág. 31). Por estas razões, o Git versão 2.7.4 foi utilizado para versionamento do código fonte e para a parte escrita do TCC.

3.2.6 Github

Github[8] é um sistema *online* de hospedagem de código-fonte utilizando Git. Permite a navegalibilidade pelo código-fonte, possui *wikis*, integração com diversas ferramentas e gerenciamento de *issues* (GITHUB, 2016).

O GitHub foi utilizado para disponibilizar a implementação e documentação dos padrões para a comunidade, estando aberto à contribuições e consultas.

3.2.7 LaTeX

LaTeX[9] é um sistema tipográfico baseado na linguagem TeX. É adotado por diversas universidades, incluindo a UnB. Suas funcionalidades facilitam a produção de documentação técnica e científica. LaTeX está disponível como software livre bem como licenciado sob a LaTeX *Project Public License* (LPPL) na versão 1.3c (LATEX, 2016). O LaTeX foi escolhido para esta monografia por simplificar a formatação da documentação.

[5] https://www.debian.org/distrib (último acesso: Junho 2017)
[6] https://eclipse.org (último acesso: Junho 2017)
[7] https://git-scm.com (último acesso: Junho 2017)
[8] https://github.com (último acesso: Junho 2017)
[9] https://www.latex-project.org (último acesso: Junho 2017)

3.2.8 Papeeria

Papeeria[10] é um editor *online* para LaTeX. O uso é gratuito para a maior parte das funcionalidades. Trata-se de um software proprietário. Para utilizar o Papeeria é necessário somente um *web browser*. Possui o editor de código fonte e o compilador necessários para a documentação. Outro diferencial importante da ferramenta é a colaboração através do GitHub, permitindo controle de versão e sincronização em tempo real (PAPEERIA, 2016). Toda a documentação da monografia foi realizada utilizando o Papeeria.

3.2.9 Parsifal

A condução da Revisão Sistemática foi apoiada pela ferramenta Parsifal[11], cujo contexto é voltado para realização de revisões sistemáticas de literatura no contexto da Engenharia de Software (PARSIFAL, 2016). Parsifal é *online* e gratuito.

3.3 Considerações parciais

Este capítulo buscou apresentar as ferramentas e tecnologias utilizadas para apoiar o desenvolvimento desta pesquisa, desde a documentação do projeto, até seu desenvolvimento e pesquisa. Pode-se observar que foram selecionadas tanto ferramentas *open-source* quanto gratuitas.

[10] https://www.papeeria.com (último acesso: Junho 2017)
[11] https://parsif.al (último acesso: Junho 2017)

4 Metodologia

Este capítulo tem o objetivo de apresentar a metodologia bem como os fluxos que foram seguidos a fim de atingir os objetivos da pesquisa. A Seção 4.1 discute sobre metodologias de pesquisa científica para, em seguida, na Seção 4.2, serem apresentadas as escolhas metodológicas adotadas por este trabalho. As seções 4.3, 4.4 e 4.5 detalham os fluxos de condução do TCC como um todo, Revisão Sistemática e desenvolvimento, respectivamente.

4.1 Metodologias de pesquisa

O conhecimento científico diverge dos demais tipos de conhecimento devido à necessidade de adotar fundamentação e metodologias a serem seguidas. Além disso, baseia-se em *"informações classificadas, submetidas à verificação, que oferecem explicações plausíveis a respeito do objeto ou evento em questão"* (PRODANOV; FREITAS, 2013, pág. 22). Ou seja, é imprescindível determinar o método científico que possibilitou atingir esse conhecimento (PRODANOV; FREITAS, 2013, pág. 24).

O desenvolvimento do método científico, mediante sua aplicação formal e sistemática, possibilita a pesquisa científica (GIL, 2008). O objetivo fundamental da pesquisa é descobrir respostas para problemas mediante o emprego de procedimentos científicos (GIL, 2008, pág. 26).

Quanto aos objetivos, é possível classificar as pesquisas em três categorias (GIL, 2002, pág. 41): (i) exploratória, (ii) descritiva e (iii) explicativa. A Tabela 2 apresenta os objetivos das mesmas.

Com relação à abordagem, a pesquisa pode ser classificada em: (i) pesquisa quantitativa e (ii) pesquisa qualitativa. A pesquisa quantitativa considera que tudo pode ser quantificável; opiniões e informações podem ser traduzidas em números para que sejam classificadas e analisadas. Este tipo de pesquisa exige o uso de recursos e de técnicas estatísticas (PRODANOV; FREITAS, 2013). A pesquisa qualitativa investiga a relação dinâmica entre o mundo real e o sujeito, isto é, um vínculo inerente entre o mundo objetivo e a subjetividade do objeto de estudo que não pode ser traduzido em números.

Segundo Prodanov e Freitas (2013), do ponto de vista da natureza, a pesquisa pode ser classificada como: (i) básica e (ii) aplicada. Pesquisa básica tem por objetivo resultar em novos conhecimentos que sejam úteis para a ciência, porém sem a definição prévia de uma aplicação prática. A pesquisa aplicada busca resultar em conhecimentos com aplicação em soluções para problemas específicos.

Tabela 2 – Classificação da pesquisa científica quanto aos objetivos.

	Pesquisa Exploratória	**Pesquisa Descritiva**	**Pesquisa Explicativa**
Objetivo	Propiciar maior familiaridade com o problema, com o propósito de torná-lo mais explícito ou a construir hipóteses (GIL, 2002, pág. 41).	Registrar e descrever os fatos observados sem interferência do pesquisador (PRODANOV; FREITAS, 2013, pág. 52).	*"Identificar os fatores que determinam ou que contribuem para a ocorrência dos fenômenos"* (GIL, 2002, pág. 42).

Cabe ressaltar, que para obter os dados para a elaboração da pesquisa são adotados os procedimentos técnicos. Existem dois grandes grupos de procedimentos técnicos, segundo Prodanov e Freitas (2013, pág. 54): (i) aqueles que se utilizam das chamadas fontes de papel (pesquisa documental e pesquisa bibliográfica) e (ii) aqueles cujos dados são produzidos por pessoas (pesquisa participante, pesquisa *ex-post-facto*, o levantamento, pesquisa experimental, o estudo de caso e a pesquisa-ação).

A pesquisa-ação busca conectar a pesquisa à ação ou prática, o que significa que o conhecimento e sua compreensão são desenvolvidos como parte da prática (ENGEL, 2000). Neste contexto, o pesquisador torna-se participante da pesquisa, onde pode intervir com o fim de verificar se um novo procedimento é eficaz.

A pesquisa-ação é um tipo de pesquisa auto-avaliativa onde procura-se intervir na prática ainda no decorrer do próprio processo de pesquisa e não somente em sua etapa final (ENGEL, 2000). Além disso, a pesquisa-ação é cíclica: para que os resultados de fases anteriores da pesquisa sejam aprimorados, os resultados das fases anteriores são avaliados (ENGEL, 2000).

4.2 Escolhas metodológicas

A Tabela 3 apresenta, brevemente, as escolhas metodológicas definidas para este trabalho. O tipo de pesquisa adotado neste trabalho, quanto aos objetivos da pesquisa, é o exploratório. Em termos de abordagem, a pesquisa qualitativa foi a mais adequada a este trabalho. Quanto à natureza, a pesquisa classifica-se como aplicada, pois buscou gerar conhecimentos a serem utilizados na solução de problemas específicos. Quanto aos procedimentos, inicialmente, optou-se por pesquisa bibliográfica para permitir que o pes-

quisador conheça os estudos já realizados sobre o assunto (FONSECA, 2002, pág. 31).

Tabela 3 – Metodologia de pesquisa

Quanto aos objetivos	**Quanto à abordagem**	**Quanto à natureza**	**Procedimentos**
Pesquisa exploratória	Pesquisa qualitativa	Pesquisa aplicada	Pesquisa bibliográfica, Revisão Sistemática, Pesquisa-ação, *Scrum*

Foi utilizada Revisão Sistemática da literatura de modo a identificar possíveis arquiteturas para SMA. Para apoiar o desenvolvimento, foi feita uma adaptação da metodologia *Scrum*. Quanto à análise dos resultados da pesquisa foi adotado o procedimento de pesquisa-ação. Diante do exposto, para plena condução desse trabalho, foram desenhados quatro fluxos principais: (i) fluxo de atividades para a condução do TCC_1; (ii) fluxo de atividades para a condução do TCC_2; (Seção 4.3); (iii) fluxo de atividades de Revisão Sistemática (Seção 4.4), e (iv) fluxo para condução das atividades associadas ao desenvolvimento e/ou à implementação dos padrões arquiteturais (Seção 4.5.2). Cabe ressaltar que o fluxo de atividades para condução do TCC compreende os demais fluxos, conforme detalhado na próxima Seção.

4.3 Fluxo de atividades para condução do TCC

A condução do TCC foi modelada em duas etapas: a etapa referente ao TCC_1 (Figura 5) e a etapa referente ao TCC_2 (Figura 6).

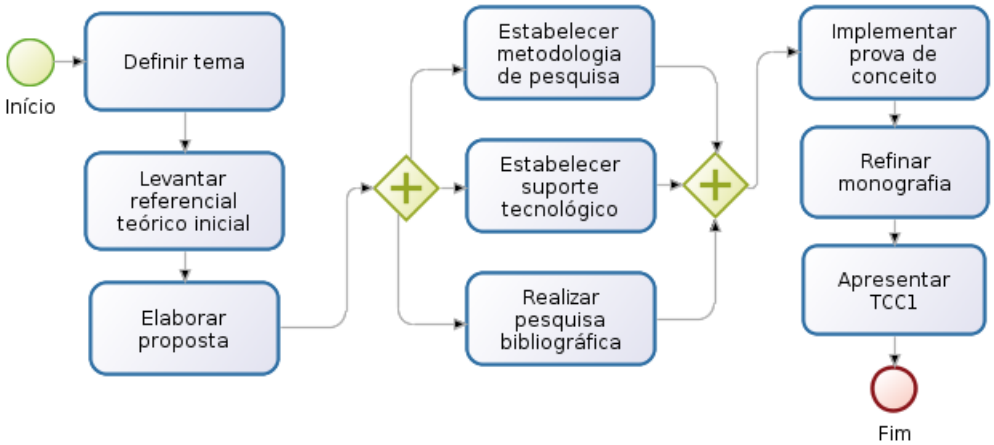

Figura 5 – Fluxo de atividades para o desenvolvimento do TCC_1.

As atividades descritas no fluxo para condução do TCC_1 (Figura 5) são apresentadas, resumidamente, a seguir:

- **Definir tema:** envolve a identificação da área de pesquisa, discussão sobre possíveis temas de relevância;
- **Levantar referencial teórico inicial:** o referencial teórico que fundamenta a proposta e consolida o tema é levantado nesta atividade;
- **Elaborar proposta:** fomentar a pesquisa de modo a estabelecer seu escopo, objetivos, justificativa, questões de pesquisa, metodologia e cronograma de atividades mais elementares. Tais aspectos podem sofrer modificações ao decorrer do trabalho;
- **Estabelecer metodologia de pesquisa:** com maior familiaridade do pesquisador sobre o tema, a metodologia de pesquisa pode ser consolidada e classificada;
- **Estabelecer suporte tecnológico:** trata-se da indicação das ferramentas a serem utilizadas, as quais apoiam o desenvolvimento da proposta;
- **Realizar pesquisa bibliográfica:** envolve a pesquisa de trabalhos científicos que forneçam embasamento para os objetivos pré-estabelecidos. O fluxo de atividades para Revisão Sistemática (Seção 4.4) foi adotado para esta atividade;
- **Implementar prova de conceito:** implementação de prova de conceito para que sejam identificados riscos potenciais que possam interferir na catalogação dos padrões. O fluxo, centrado em atividades de desenvolvimento e/ou implementação, adotado nesse trabalho, encontra-se descrito na (Seção 4.5.2);
- **Refinar monografia:** com base nos resultados obtidos até o momento, foi averiguado bem como documentado se os objetivos estabelecidos para cumprimento no TCC_1 foram atingidos. Isso permitiu refinar a parte escrita da monografia, e
- **Apresentar TCC_1**: consiste na apresentação do TCC_1 à banca examinadora.

As atividades descritas no fluxo para condução do TCC_2 (Figura 6) são apresentadas, brevemente, a seguir.

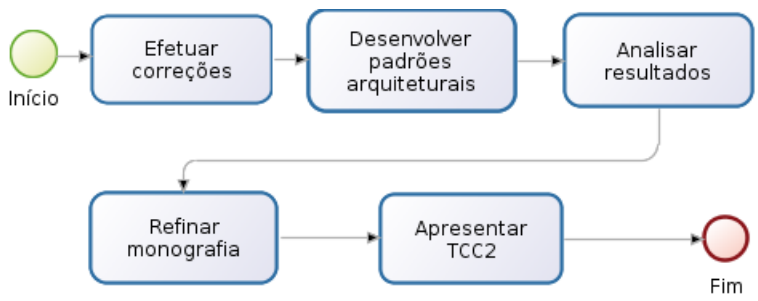

Figura 6 – Fluxo de atividades para o desenvolvimento do TCC_2.

- **Efetuar correções:** adequação da monografia às sugestões e críticas da banca examinadora;
- **Desenvolver padrões arquiteturais:** uma vez que os padrões arquiteturais já foram identificados, essa atividade compreende a modelagem, a implementação bem como a catalogação desses padrões. Vale ressaltar que a prova de conceito realizada, permitiu configurar o ambiente de desenvolvimento bem como acordar possíveis riscos e estabelecer caminhos seguros para a condução dessa atividade;
- **Analisar resultados:** foi adotado o procedimento de pesquisa-ação para avaliação de cada padrão arquitetural catalogado. Portanto, os passos relacionados à análise de resultados foram detalhados após o desenvolvimento dos padrões, de forma cíclica, atendendo, portanto, à característica inerente da pesquisa-ação; onde o pesquisador pode intervir na prática ainda no decorrer do próprio processo de pesquisa, baseando-se nos resultados obtidos até determinado momento. Os dados coletados, podem ser de natureza qualitativa ou quantitativa. Além disso, a avaliação dos padrões pode ocorrer unicamente por parte do pesquisador, ou por parte de desenvolvedores de SMA.
- **Refinar monografia:** com base nos resultados obtidos ao final do projeto, foi averiguado bem como documentado se os objetivos estabelecidos para cumprimento no TCC como um todo foram atingidos. Isso permitiu refinar a parte escrita da monografia;
- **Apresentar TCC_2:** apresentação do TCC_2 à banca examinadora, e
- **Efetuar correções finais:** possui como intuito realizar as correções e os refinamentos apontados pela banca após a apresentação do TCC_2.

4.4 Revisão Sistemática

A maioria das pesquisas começa com uma revisão da literatura. No entanto, a revisão de literatura possui pouco valor científico (KITCHENHAM, 2004). Uma Revisão Sistemática (RS) pode ser adotada neste caso. Uma RS, é um meio de identificar, avaliar e interpretar boa parte das pesquisas disponíveis e, ao mesmo tempo, relevantes para um determinado foco (KITCHENHAM, 2004).

A RS sintetiza trabalhos existentes de acordo com uma estratégia de busca pré-definida. A estratégia de busca deve permitir que a integridade da pesquisa seja avaliada. Dentre as razões para realizar uma Revisão Sistemática da literatura, Kitchenham (2004) destaca:

- Resumir a evidência existente sobre algum tópico em particular;

- Identificar eventuais lacunas na pesquisa atual, a fim de sugerir novas áreas de investigação, e
- Fornecer um *framework* ou *background* para outras pesquisas.

A condução do processo de Revisão Sistemática pode ser entendida como uma abordagem em três fases, como apresentado na Figura 7. A primeira fase da pesquisa começa a partir de conceitos, que, de forma explícita e formalmente, representam o assunto em questão. São definidos os procedimentos que o pesquisador deve utilizar. Em seguida, é realizada a análise dos estudos que podem conter as informações e evidências sobre o tema específico da investigação (BIOLCHINI et al., 2005).

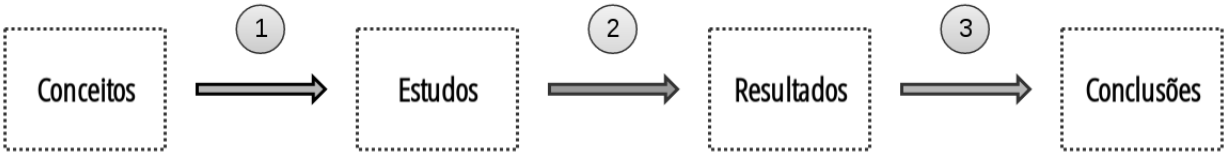

Figura 7 – Abordagem de Revisão Sistemática em três fases. Traduzido. Fonte: BIOLCHINI et al., 2005.

A segunda fase começa a partir destes estudos coletados. Neste momento é realizada a distinção de quais estudos são relevantes e irrelevantes para o propósito da investigação. São examinados em seus conteúdos, comparados entre si, e às vezes reagrupados em suas partes. Obtêm-se, assim, os resultados, que representam o surgimento de um novo tipo de prova (BIOLCHINI et al., 2005).

A terceira fase parte destes resultados, através da análise e síntese deste rearranjo de dados, e chega-se às conclusões. Estas implicam em adquirir novos conhecimentos sobre o problema em questão ou apoiar algum processo de decisão sobre o mesmo (BIOLCHINI et al., 2005).

O que motivou a adoção da técnica de RS neste trabalho foi o fato de fornecer de modo formal, através de um protocolo, ou seja, através de passos repetíveis, o *background* para a elaboração do catálogo. Foram identificados os modelos arquiteturais organizacionais para SMA existentes por meio da RS.

4.4.1 Fluxo de atividades para Revisão Sistemática

A Figura 8 ilustra o fluxo das atividades que orientou a execução da RS. Foi elaborado o protocolo de Revisão Sistemática definido no Apêndice A. O objetivo é disponibilizar os procedimentos e decisões do fluxo de atividades da RS realizado de modo que possa ser repetido por outros pesquisadores que obterão os mesmos resultados. A estrutura deste protocolo baseou-se no trabalho de Silva (2014).

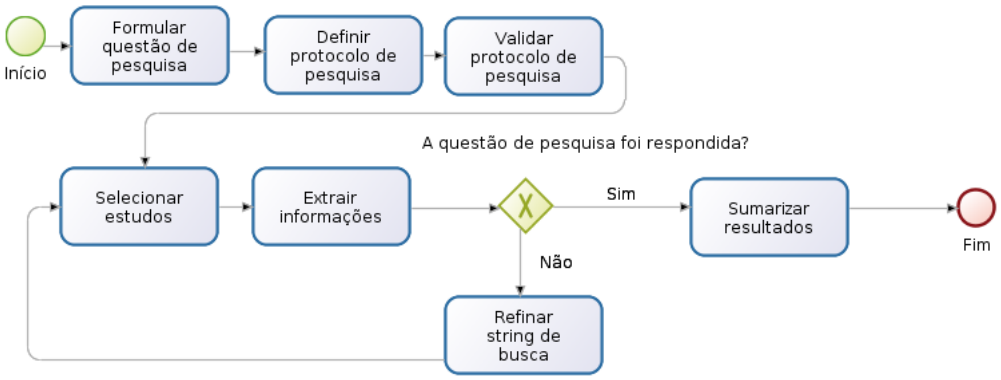

Figura 8 – Fluxo de atividades para Revisão Sistemática.

4.5 Metodologia de desenvolvimento

Foi adotada uma adaptação da metodologia Scrum para a atividade *"Desenvolver padrões arquiteturais"* descrita na Figura 6.

4.5.1 Scrum

O Scrum é um *framework*, no qual pode-se resolver problemas complexos de forma produtiva com o principal objetivo de gerar produtos de maior valor possível (SCHWABER; SUTHERLAND, 2016). Isto é favorecido empregando-se uma abordagem iterativa e incremental que promove controle de riscos e uma melhor previsão dos mesmos. O Scrum fundamenta-se no empirismo, afirmando que o conhecimento é desenvolvido pela experiência. São três os pilares que sustentam a implementação do processo empírico:

1. Transparência: aspectos significantes do fluxo devem ser visíveis para os responsáveis pelos resultados. A comunicação deve ser frequente e, consequentemente, estabelece confiança dos envolvidos;

2. Inspeção: a transparência torna a inspeção possível. Envolve examinação atenciosa e *feedbacks* constantes para tomar decisões com relação a adaptações no processo ou produto, e

3. Adaptação: a inspeção torna a adaptação possível. Adaptação diz respeito aos ajustes no produto em desenvolvimento ou no processo; o qual está sendo desenvolvido de acordo com os resultados da inspeção (RUBIN, 2012).

O objetivo do Scrum é entregar software de qualidade, tanto quanto possível, em intervalos curtos e fixos de tempo, intitulados *Sprints* (BEEDLE et al., 1999). Cada fase do ciclo de desenvolvimento de software - consideradas por Beedle et al. (1999) como requisitos, análise, projeto, evolução e entrega - é mapeada para uma *Sprint* ou conjunto de *Sprints* (BEEDLE et al., 1999).

Além da *Sprint*, existem outros eventos prescritos no Scrum: *Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospective*. Estes eventos são usados para criar regularidade de encontros do time bem como minimizar a necessidade de reuniões não definidas no Scrum. Cada um desses eventos requer que seja estabelecida uma duração máxima (BEEDLE et al., 1999).

Para definir se o que foi produzido é potencialmente entregável, o *Scrum Team* deve ter a definição de pronto - *Definition of Done* (DoD) - bem estabelecida. A definição de pronto é uma *checklist* de tipos de trabalho que o time deve realizar completamente para declarar o trabalho desenvolvido na *Sprint* como pronto (RUBIN, 2012).

Cada *Sprint* opera com um certo número de itens de trabalho, constituindo o *Backlog*. Os papéis responsáveis pelo processo representam o *Scrum Team*. Este é constituído por: *Product Owner*, *Development Team*, e um *Scrum Master* (BEEDLE et al., 1999).

4.5.2 Fluxo de atividades de desenvolvimento

Foram realizadas adaptações no *Scrum* devido ao fato de haver somente um membro na equipe para compor o *Scrum Team*. Portanto, foram utilizados: entregas a cada *Sprint*, cerimônia *Sprint Planning* e a utilização do *Definition of Done*. Foram adotados os procedimentos descritos na Figura 9. As *Sprints* tiveram duração de duas semanas, de modo que as estórias de usuário foram dadas como prontas seguindo a definição de pronto - ou *Definition of Done* (ou “DoD”).

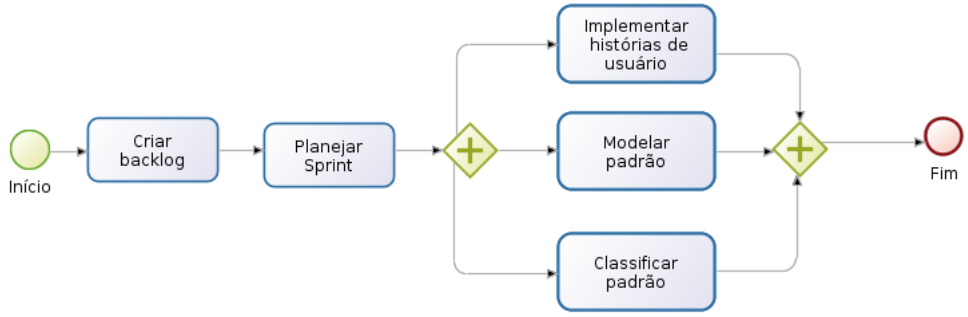

Figura 9 – Fluxo de atividades para o desenvolvimento de software.

4.6 Cronograma

Os cronogramas referentes aos TCC 1 e 2 são apresentados nas Tabelas 4 e 5, respectivamente.

Tabela 4 – Cronograma de atividades do TCC_1 realizado em 2017

Atividades	Fevereiro	Março	Abril	Maio	Junho
Definir tema	X				
Levantar referencial teórico inicial	X				
Elaborar proposta	X				
Estabelecer metodologia de pesquisa		X			
Estabelecer suporte tecnológico		X			
Realizar pesquisa bibliográfica		X	X		
Implementar prova de conceito			X	X	
Refinar monografia			X	X	
Apresentar TCC_1					X
Efetuar correções					X

Tabela 5 – Cronograma de atividades do TCC_2 realizado em 2018.

Atividades	Março	Abril	Maio	Junho	Julho
Desenvolver padrões arquiteturais	X	X	X		
Analisar resultados		X	X	X	
Refinar monografia				X	X
Apresentar TCC_2					X

4.7 Considerações parciais

Esse capítulo teve como intuito apresentar os métodos utilizados no desenvolvimento desse trabalho. Ele contém as escolhas metodológicas, bem como os fluxos de atividades modelados que guiaram a condução deste. Elaborou-se um protocolo de condução da Revisão Sistemática, e foi apresentada uma visão temporal das atividades, por meio de cronogramas.

5 Proposta

A seguir, será apresentada a proposta desse trabalho, em atendimento aos objetivos (Seção 1.3) acordados no Capítulo 1, Introdução. Procurou-se, ao longo do trabalho, concretizar essa proposta (lê-se, investigar, elaborar, implementar, avaliar e evoluir a mesma). Os resultados desse processo serão apresentados no Capítulo 6, Resultados Obtidos.

O presente capítulo está organizado em seções, visando apresentar uma visão geral do catálogo proposto, enfatizado como o mesmo foi construído (Seção 5.1). Em seguida, é apresentado um trabalho relacionado com o tema, identificado durante as pesquisas (Seção 5.2). Pretende-se, ao final deste capítulo, fornecer uma ideia mais concreta sobre a proposta deste trabalho.

5.1 O catálogo

Na Seção 2.7, foi abordado o conceito de catálogo de padrões de projeto bem como apresentados os itens que o mesmo deve fornecer. Com base nestes aspectos, o catálogo de padrões arquiteturais organizacionais para SMA proposto encontra-se estruturado em *categorias* de padrões.

Para que se classifique um modelo arquitetural como padrão arquitetural, esse deve atender aos critérios de classificação definidos na Seção 5.1.1. De modo semelhante, critérios foram definidos na Seção 5.1.2 para que se classifique um padrão arquitetural em determinada categoria de estrutura organizacional.

5.1.1 Critérios para padrões arquiteturais

Ao realizar a Revisão Sistemática, os trabalhos retornados e aceitos possuem modelos arquiteturais para SMA. Estes são candidatos a padrões arquiteturais, ou seja, os modelos necessitam de uma verificação apurada de suas características para serem classificados como padrões arquiteturais ou serem descartados. Assim, pode-se elaborar um catálogo contendo, genuinamente, padrões arquiteturais.

Nas Seções 2.4 e 2.5, são abordados conceitos de padrão arquitetural e estruturas organizacionais. Baseando-se nesses conceitos, os seguintes critérios foram estabelecidos para a classificação dos modelos arquiteturais identificados durante a Revisão Sistemática em padrão arquitetural organizacional:

1. Pode ser identificado, na descrição do modelo, o tripé problema-solução-contexto;

2. O modelo é representado por algum diagrama, em notação adequada, facilmente compreensível na comunidade de software;

3. O modelo captura a prática comum (ou seja, possui pelo menos três utilizações conhecidas e, ao mesmo tempo, a solução não é óbvia), e

4. O modelo baseia-se em estruturas organizacionais conhecidas.

5.1.2 Classificação de padrões arquiteturais em categorias e subcategorias

Malone (1990), Sycara (1998), Argente, Julian e Botti (2006) abordam, em seus trabalhos, estruturas de coordenação de SMA a partir do emprego da Teoria Organizacional. Após a análise das categorias de cada trabalho, observou-se que as mesmas possuíam o mesmo conceito, diferenciando-se apenas pelo nome e pelos autores dos trabalhos que as citam. São, portanto, equivalentes e sinônimos para este trabalho: *Market* e *Organic*; *Hierarchy* e *Mecanic*.

A seguir, tais categorias serão apresentadas em detalhe e fomentaram as categorias e subcategorias, as quais os padrões definidos no catálogo foram classificadas. A Figura 13 ilustra a estrutura das categorias e subcategorias.

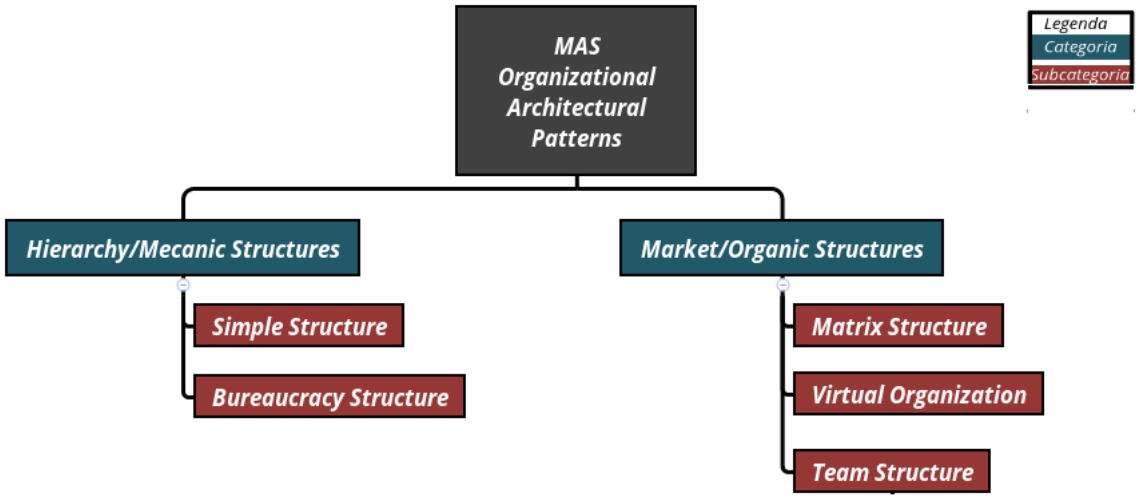

Figura 10 – Categorias e Subcategorias. Fonte: Autora.

5.1.2.1 Categorias *Mecanic* e *Organic*

Argente, Julian e Botti (2006) realizam uma revisão das organizações humanas mais conhecidas para que possam ser aplicadas no campo de SMAs. Tais organizações podem ser usadas como base para descrever papéis, padrões e conexões que colaboram com a definição de arquiteturas para SMAs. A Teoria Organizacional (*Organizational Theory*) analisa como funcionam as organizações, suas características principais, as características

mais relevantes de seus membros, seus papéis, as relações dos membros, a cadeia de comando, as regras e normas que regem a organização, dentre outros aspectos.

Nesse sentido, Argente, Julian e Botti (2006) definem dois tipos de organizações humanas: mecânicas e orgânicas. Nas organizações mecânicas, as tarefas são definidas com precisão, e são divididas em partes separadas e especializadas. Existe uma hierarquia de autoridade rígida. Os processos de conhecimento e raciocínio também são centralizados no topo da hierarquia. As comunicações são, principalmente, verticais (entre supervisores e subordinados). São três exemplos deste tipo de organização: *Simple Structure*, *Bureaucracy Structure* e *Matrix Structure*. A Figura 11 apresenta características de cada uma dessas organizações humanas.

		Human Organizations				
		Simple	Bureaucracy	Matrix	Team	Virtual
Organizational features	Centralized capture of decisions	√	√	√	X	X
	Work specialization	X	√	√	√	X
	Generalist members	X	X	X	√	√
	Departmentalization	X	√	√	X	X
	Span of control	√	X	√	X	X
	Formalization of tasks	X	√	√	X	X
	Coordination between specialists	X	√	√	√	X
	Authority	√	√	√	X	X
	Chain of command	X	√	√	X	X
	Several direct managers	X	X	√	X	X
	Department goals	X	√	X	X	X
	Organization goals	√	X	√	√	√
	Shared information	X	X	X	√	√
	Flexibility	X	X	X	√	√
	Business functions outsourcing	X	X	X	X	√

Figura 11 – Características de organizações humanas. Fonte: Argente, Julian e Botti (2006).

Nas organizações orgânicas, as tarefas são ajustadas e redefinidas por meio de trabalho colaborativo em grupos (ARGENTE; JULIAN; BOTTI, 2006). Existem menos níveis de autoridade e controle, de modo que o controle do conhecimento e das tarefas são distribuídos. Todos os membros devem contribuir para a tarefa comum do departamento. As comunicações são principalmente horizontais entre membros do mesmo departamento, ou mesmo entre diferentes departamentos. Desta forma, eles podem oferecer respostas rápidas e flexíveis. São exemplos deste tipo de organização: *Team Structure* e *Virtual Organizations*.

A subcategoria ***Simple Structure*** representa uma organização com poucos departamentos, onde um indivíduo centraliza a captura de decisões (ARGENTE; JULIAN; BOTTI, 2006). Esta estrutura apresenta um alto grau de controle, pois o gerente dirige um grande número de pessoal. Este tipo de organização é, frequentemente, empregado

em pequenas empresas, onde o gerente e proprietário são a mesma pessoa, bem como em grandes empresas quando o controle é centralizado em um indivíduo.

Esta organização é simples: as responsabilidades são claras, as comunicações são diretas e a captura de decisões e sua execução é rápida. No entanto, é recomendada somente para organizações pequenas, pois há pouca formalização e o gerente deve lidar com muitas informações. Já que tudo depende de uma única pessoa, se essa pessoa falhar ou tomar uma decisão errada, a empresa pode ser prejudicada (ARGENTE; JULIAN; BOTTI, 2006).

A subcategoria ***Bureaucracy Structure*** é caracterizada, principalmente, por tarefas operacionais e de rotina com alta especialização. Há também muitas regras e regulamentos formalizados. Por haver diversos departamentos, é ocasionado um baixo nível de controle, com o qual os gerentes controlam um pequeno grupo de pessoas ou departamentos (ARGENTE; JULIAN; BOTTI, 2006).

Há uma autoridade central e a tomada de decisões segue uma cadeia de comando. Entre suas vantagens, esta estrutura permite que as atividades padrão sejam realizadas de forma muito eficaz. Os especialistas são reunidos nos mesmos departamentos, facilitando as comunicações entre os funcionários. Graças à ampla presença de regras e regulamentos e à padronização das operações, a tomada de decisões é centralizada em gerentes executivos (ARGENTE; JULIAN; BOTTI, 2006).

No entanto, a alta especialização das tarefas pode criar conflitos em unidades ou departamentos. Os gerentes podem estar mais focados em alcançar seus objetivos individuais do que os objetivos gerais da organização (ARGENTE; JULIAN; BOTTI, 2006).

Além disso, os gerentes podem estar excessivamente concentrados em seguir as regras, o que dificulta suas decisões quando confrontados com novas situações. Portanto, esses tipos de organizações tem dificuldades em responder às mudanças externas (ARGENTE; JULIAN; BOTTI, 2006).

A subcategoria ***Matrix Structure*** possui dois supervisores: o gerente do departamento funcional e o gerente do produto. Portanto, existem duas cadeias de controle. Esse tipo de estrutura é muito comum em empresas de engenharia e gerenciamento de projetos. Facilita a coordenação entre funcionários quando há numerosas atividades complexas e interdependentes (ARGENTE; JULIAN; BOTTI, 2006).

Outra vantagem consiste na melhoria das comunicações e ampliação da flexibilidade. Também reduz a possibilidade dos membros se concentrarem nos objetivos individuais de seus departamentos mais do que nos objetivos gerais da organização. Da mesma forma, facilita a localização efetiva de especialistas (ARGENTE; JULIAN; BOTTI, 2006).

No entanto, pode haver confusão na tomada de decisões, pois existem duas cadeias

de comando. Além disso, esta estrutura promove lutas de poder e pode criar tensão, que pode ser diminuída usando técnicas burocráticas, como, por exemplo, uma formalização maior das regras (ARGENTE; JULIAN; BOTTI, 2006).

A subcategoria ***Team Structure*** elimina barreiras departamentais e descentraliza a tomada de decisão. Equipes ou grupos representam um sistema com vários atores que possuem um objetivo comum: a realização da tarefa global do sistema (ARGENTE; JULIAN; BOTTI, 2006).

Esta tarefa é dividida em sub-tarefas, que são atribuídas aos membros mais qualificados do grupo. Além disso, os membros compartilham todas as informações e estão em constante comunicação uns com os outros. A coordenação entre atores é obtida usando decisões e planos mutuamente aceitos (ARGENTE; JULIAN; BOTTI, 2006).

A subcategoria ***Virtual Organization*** consiste em uma empresa que terceiriza suas principais funções comerciais. Várias redes de contato são criadas, o que permite que a empresa contrate funções comerciais que os gerentes acreditam que possam ser feitas por outras empresas de uma maneira melhor ou a um custo menor. Esta estrutura oferece flexibilidade, mas reduz o controle de gerenciamento em partes fundamentais da organização (ARGENTE; JULIAN; BOTTI, 2006).

5.1.2.2 Categorias *Markets* e *Hierarchies*

Malone (1990, pág. 61) considera três processos fundamentais de coordenação de SMA:

- **Ajuste mútuo:** trata-se do modo mais simples de coordenação, no qual dois ou mais agentes compartilham seus recursos visando atingir um objetivo comum. Nenhum agente tem controle prévio sobre os outros, e a tomada de decisões é um processo conjunto;
- **Supervisão direta:** ocorre quando dois ou mais agentes já estabeleceram um relacionamento em que um agente tem algum controle sobre os outros. Nesta forma de coordenação, o supervisor controla o uso de recursos compartilhados - como trabalho humano, tempo e dinheiro do processo de computador - pelos subordinados, e também pode prescrever certos aspectos de seus comportamentos, e
- **Padronização:** ocorre quando o supervisor estabelece procedimentos padrão para subordinados a serem seguidos em várias situações. Os procedimentos de rotina em empresas ou em software são exemplos de coordenação por meio da padronização.

Por meio desses processos fundamentais de coordenação podem ser elaborados sistemas de coordenação mais sofisticados (MALONE, 1990, pág. 63). São eles:

- ***Markets*/Mercados:** podem ser considerados uma forma de organização baseada em ajuste mútuo. Os agentes, cada um dos quais controla recursos escassos - como, por exemplo, trabalho, matéria-prima, bens e dinheiro, concordam em compartilhar alguns de seus respectivos recursos para alcançar o objetivo mútuo. Os recursos são trocados, com ou sem custos explícitos. Uma vez que um contrato foi feito, existe um acordo em que o "comprador"se torna o supervisor do fornecedor, e

- ***Hierarchies*/Hierarquias:** baseiam-se em processos de supervisão direta. Um grande grupo pode ser dividido em subgrupos se a maior parte da transferência de informações necessária pode ocorrer em subgrupos e se as poucas interações entre subgrupos podem ser tratadas por supervisores. Nesse caso, pode-se utilizar como estratégia a hierarquia. Os subgrupos podem ser coordenados por ajuste mútuo ou por controle hierárquico, dependendo do domínio do aplicativo e das características da tarefa.

Sycara (1998), de modo semelhante, apresenta *Market* e *Hierarchy* como exemplo de organizações que são exploradas na literatura. Complementando o conceito de *Hierarchies*, Sycara (1998) define que esta estrutura é formada por uma autoridade para tomada de decisão e controle concentrada em um único solucionador de problemas (ou grupo especializado) em cada nível da hierarquia. A interação é através da comunicação vertical do agente superior ao subordinado e vice-versa; de modo que agentes superiores exerçam controle sobre recursos e tomada de decisões.

Na estrutura *Market*, o controle é distribuído aos agentes que competem por tarefas ou recursos através de lances e mecanismos que envolvem contratos. Os agentes interagem através de uma variável, como o custo, que é usado para avaliar os serviços (SYCARA, 1998).

5.1.2.3 Critérios para classificação em categorias e subcategorias

Baseando-se nos conceitos abordados até o momento, foram elencados critérios para classificar as estruturas organizacionais dos padrões arquiteturais identificados para que sejam catalogados. Os critérios para classificar uma estrutura organizacional como *Hierarchy/Mecanic* são:

- Há uma autoridade - ou seja, um agente ou um grupo de agentes superior - para a tomada de decisão e o controle de recursos;

- A comunicação dá-se verticalmente; da autoridade para o subordinado e vice-versa, e

- A autoridade pode prescrever aspectos do comportamento dos subordinados.

Os critérios para classificar uma estrutura organizacional como *Markets/Organic* são:

- Nenhum agente possui controle sobre os demais;
- Dois ou mais agentes compartilham seus recursos visando atingir um objetivo comum;
- Os agentes competem por tarefas ou recursos através de lances ou contratos, e
- A tomada de decisões ocorre mutuamente.

5.1.3 Elementos para a descrição dos padrões arquiteturais

Baseando-se na literatura discutida na Seção 2.6 sobre padrões de projeto e os elementos fundamentais para descrevê-los, o catálogo proposto adota os seguintes elementos:

Nome do padrão: nome do padrão encontrado na literatura;

Referências: referências para os trabalhos que descrevem o padrão;

Categoria: classificação do padrão em *Market* - e subcategorias *Simple Structure* e *Bureaucracy Structure* -, e *Hierarchy* - com subcategorias *Matrix Structure*, *Virtual Organization* e *Team Structure*;

Problema: problema em que o padrão busca solucionar, descrevendo-se seus objetivos;

Solução: instruções descrevendo como o problema pode ser resolvido;

Protocolos associados: este elemento é opcional. Caso o padrão esteja implementado em alguma plataforma acessível, os protocolos ou bibliotecas associados serão referenciados;

Modelagem: diagramas disponibilizados pela literatura representando o padrão. A notação utilizada deve ser apropriada para SMA como, por exemplo, diagrama de sequência UML (LARMAN, 2002);

Exemplo: quando disponível na literatura, será apresentado um exemplo prático do padrão arquitetural, podendo conter trechos de código e figuras que torne o exemplo mais didático, e

Implementação: a descrição do padrão poderá conter mais um elemento: sua implementação. A implementação dependerá da viabilidade tecnológica para que seja cumprido e preenchido para cada padrão.

5.2 Trabalho relacionado

Durante a pesquisa para fundamentar a base teórica e formular os objetivos deste trabalho, pôde-se encontrar um trabalho semelhante, o qual será apresentado a seguir.

Junior et al. (2003) propõe uma coleção de padrões arquiteturais para SMA voltados a minimizar problemas de comunicação e cooperação entre agentes. A coleção baseia-se na extração de arquiteturas frequentemente utilizadas bem como na geração de novos padrões a partir da extensão ou composição de padrões existentes.

A metodologia utilizada para identificar os padrões baseia-se no estudo de Deugo, Weiss e Kendall (2001) - que aborda apenas padrões sob o contexto da coordenação - e na técnica "*pattern mining*". Por fim, os padrões foram agrupados em coletâneas, de modo a ficarem acessíveis a outros desenvolvedores.

A técnica "*pattern mining*"pode ser representada pela Figura 12 (JUNIOR et al., 2003). Observa-se que os padrões são extraídos de aplicações bem sucedidas para que, em seguida, possam ser agrupados em catálogos de modo a solucionar problemas maiores, originando *frameworks* que serão utilizados para desenvolver aplicações completas.

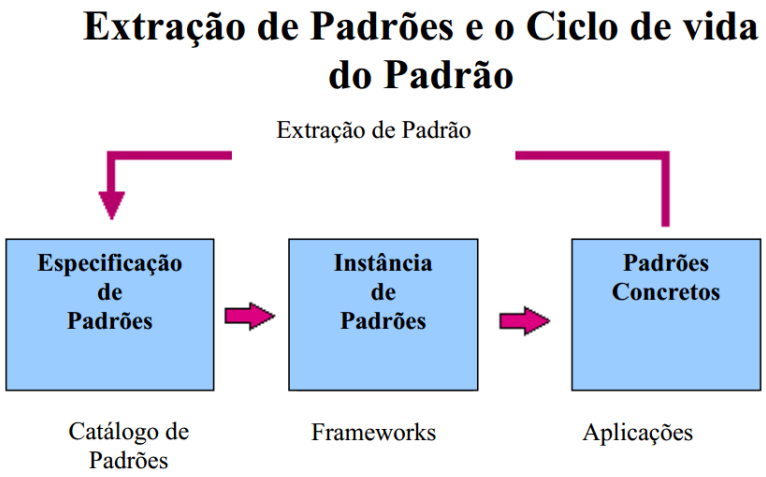

Figura 12 – Extração de padrões e o ciclo de vida de um padrão. Fonte: Junior et al. (2003 apud KOSKIMIES, 2002).

5.2.1 Comparação com o trabalho atual

Comparando em linhas gerais, a proposta deste trabalho e o trabalho descrito na Seção anterior, distinguem-se em vários aspectos. O trabalho de Junior et al. (2003) utiliza técnicas diferentes das utilizadas por este trabalho, o qual utiliza Revisão Sistemática de literatura para identificar os padrões arquiteturais, enquanto Junior et al. (2003) baseia-se no estudo de Deugo, Weiss e Kendall (2001) e utiliza a técnica "*pattern mining*"para identificar os padrões.

De modo semelhante, o trabalho de Junior et al. (2003) apresenta um catálogo

de padrões arquiteturais para SMA. Tais padrões são voltados para solucionar problemas de coordenação e comunicação. Neste contexto, estruturas organizacionais são soluções alternativas, porém, não chegam a ser abordadas no trabalho de Junior et al. (2003).

Apenas a notação UML é utilizada para descrever os padrões, em diagramas de pacote e de interação (JUNIOR et al., 2003). Utiliza-se, também, mensagens KQML para representar a comunicação entre agentes. No presente trabalho, a notação para modelagem dos padrões, bem como a modelagem dos padrões, relacionadas à sua implementação serão definidas a partir da literatura que define o padrão.

Tal aspecto, permite se adequar mais precisamente em atendimento às necessidades de cada padrão bem como colabora com representações mais didáticas possíveis para aqueles que consultarão e se apoiarão nos padrões arquiteturais oferecidos no catálogo. Outra diferença identificada consiste no fato de que o catálogo apresentado por Junior et al. (2003) não é organizado em categorias de padrões.

Esta estratégia, confere aos interessados no presente catálogo, maior conveniência em termos de facilidades de busca aos padrões. Adicionalmente, padroniza a evolução do catálogo por futuros colaboradores. Afinal, a proposta desse catálogo é ser um artefato vivo, evolutivo, permitindo, dentre outros aspectos, e se assim desejado, o acompanhamento de tendências tecnológicas ou mesmo o uso de notações mais genéricas possíveis para que o poder de reutilização do conteúdo seja ampliado. Lembrando que os únicos aspectos mais engessados do presente catálogo são, basicamente: os critérios para definição de categorias e subcategorias e a estrutura base para especificação dos padrões (Seção 5.1).

5.3 Considerações parciais

Esse trabalho consistiu na identificação de padrões arquiteturais organizacionais para SMA. Para isso, foi realizada uma Revisão Sistemática para encontrar e registrar diferentes modelos arquiteturais utilizados atualmente, sendo estes candidatos a padrões arquiteturais.

Com a identificação destes padrões, durante o TCC_2, os esforços voltaram-se para compor o catálogo, orientando-se por critérios específicos, uma proposta de classificação e uma estrutura para especificação e descrição - clara - para os padrões. Esse processo envolveu atividades de investigação, modelagem, implementação, análise e evolução. Os resultados obtidos com essas atividades são apresentados no próximo capítulo.

6 Resultados obtidos

Nesse capítulo, serão apresentados os resultados alcançados ao longo da realização desse trabalho, sendo: os resultados da Revisão Sistemática (Seção 6.1) e os avanços oriundos da concretização do catálogo como um todo, com foco em atividades de modelagem, implementação, análise e evolução (Seção 6.2).

6.1 Revisão sistemática

Nesta Seção, são relatados os resultados obtidos da Revisão Sistemática.

6.1.1 Primeiro ciclo de busca

Para o primeiro ciclo de busca, foram estudados os artigos que abordam diretamente o objeto de estudo. Estes estão listados no Apêndice A.2.4. Assim, obteve-se as palavras-chaves e sinônimos que, potencialmente, resultariam em uma *string* de busca satisfatória.

A *string* resultante deste processo foi: *"Multiagent System"AND ("Organization* Structure"OR "Organisation* Structure") AND "Architectur* Pattern"*. Entretanto, nas duas fontes de pesquisa utilizadas não foram retornados trabalhos. A *string* foi, então, descartada e repensada para o próximo ciclo de busca.

6.1.2 Segundo ciclo de busca

O segundo ciclo de busca foi fundamental para se obter uma visão prévia da pesquisa e dos rumos que a mesma deveria tomar. Possibilitou a identificação de modelos arquiteturais desenvolvidos para SMA candidatos a padrões arquiteturais. Em sua maioria, são modelos desenvolvidos para situações específicas como em contextos de suporte a *workshops*, *e-commerce*, computação em nuvem e infraestrutura.

Dentre os artigos rejeitados, muitos continham descrições acerca de modelos igualmente originados para situações específicas. Porém, foram rejeitados por se limitarem a seus contextos e não fazerem parte dos objetivos destes artigos, propondo estruturas reutilizáveis e/ou genéricas. Em suma, os resultados obtidos a partir desta busca são apresentados na Tabela 6.

Tabela 6 – Resultados obtidos com a *string* inicial

Fonte de pesquisa	***String* de busca**	**Nº de artigos retornados**	**Número de artigos aceitos**
IEEExplorer	*"Multiagent System"AND ("Organization* Structure"OR "Organisation* Structure")*	76	12
Science Direct	*"Multiagent System"AND ("Organization Structure"OR "Organisation Structure"OR "Organizational Structure"OR "Organisational Structure")*	37	5

Os artigos aceitos e obtidos pela fonte de pesquisa *IEEExplorer* são listados a seguir:

- *Embedding intelligent agents to enable physical robotic and sensor organizations* (MATSON, 2009);
- *Modelling Energy and Transport Infrastructures as a Multi-Agent System using a Generic Ontology* (DAM; LUKSZO, 2006);
- *Study on Cooperation Mechanism in Agent Organization* (PENG et al., 2006);
- *Collective Agents and Collective Intentionality Using the EDA Model* (FILIPE; FRED, 2007);
- *Computational Organization Theory* (WEISS, 2000);
- *Conflict Resolution within Multi-agent System in Collaborative Design* (LIU; CUI; HU, 2008);
- *Research on the Bidding Information Integrated System of Generation Group* (CHEN; XIAO; SONG, 2008);
- *Multi-agent Based Power Grid Data Integration and Sharing Platform* (WU; DUAN; SHI, 2009);
- *Research on Grid-Service Based Virtual Products Experience Environment Supporting Architecture in E-commerce Applications* (CHEN, 2009);

- *Using Multiagent Self-organization Techniques to Improve Dynamic Skill Searching in Virtual Social Communities* (MERCIER; OCCELLO; JAMONT, 2010);
- *A Generic Recursive Multiagent Model to Simplify Large Scale Multi-level Systems Observation* (HOANG; OCCELLO; JAMONT, 2011);
- *Multi-agent System to Support Creative Workshop* (GABRIEL et al., 2015).

Os artigos aceitos e obtidos pela fonte de pesquisa *Science Direct* são listados a seguir:

- *A low-level resource allocation in an agent-based Cloud Computing platform* (BAJO et al., 2016);
- *Comparison of tightly and loosely coupled decision paradigms in multiagent expedition* (XIANG; HANSHAR, 2010);
- *AEGIS: Agent oriented organisations* (UNLAND et al., 1995);
- *Expectation-oriented modeling* (NICKLES; ROVATSOS; WEISS, 2005);
- *Multi-Agent System Development Based on Organizations* (ARGENTE; JULIAN; BOTTI, 2006).

6.1.3 Aplicação dos critérios para padrões arquiteturais

A próxima etapa consistiu em extrair o conteúdo dos artigos aceitos para avaliar quais apresentavam modelos arquiteturais que se caracterizavam como padrões arquiteturais baseando-se nos critérios definidos na Seção 5.1.1. A Tabela 7 apresenta a relação de modelos identificados e os critérios que os mesmos atendem ou não atendem.

O critério de número 3 (Seção 5.1.1), que aborda a utilização do modelo por no mínimo três trabalhos conhecidos, foi o que mais dificultou a caracterização dos modelos enquanto um possível padrão arquitetural, como indicado na Tabela 7. Se um modelo divulgado na literatura não é reproduzido em um diferente contexto, isto é um indício de que ele é muito específico e, portanto, não é repetível para outros sistemas; não se enquadrando como um padrão arquitetural.

Tem-se como resultado da RS, bem como considerando a aderência das fontes investigadas aos critérios estabelecidos na Seção 5.1.1, a identificação de um artigo chave para a realização da presente pesquisa. Esse artigo, aliado aos estudos realizados em artigos similares, possibilitou a construção do catálogo conforme planejado. O trabalho de Argente, Julian e Botti (2006) resultou não somente na identificação de categorias e subcategorias de SMA, como também nos padrões arquiteturais organizacionais que se buscava. A Seção 2.7 apresenta o catálogo por completo.

Título do artigo	**Fonte**	**Critérios não atendidos**
Embedding intelligent agents to enable physical robotic and sensor organizations (MATSON, 2009)	*IEEExplorer*	3, 4
Modelling Energy and Transport Infrastructures as a Multi-Agent System using a Generic Ontology (DAM; LUKSZO, 2006)	*IEEExplorer*	3
Study on Cooperation Mechanism in Agent Organization (PENG et al., 2006)	*IEEExplorer*	2, 4
Collective Agents and Collective Intentionality Using the EDA Model (FILIPE; FRED, 2007)	*IEEExplorer*	1, 2
Computational Organization Theory (WEISS, 2000)	*IEEExplorer*	2
Conflict Resolution within Multi-agent System in Collaborative Design (LIU; CUI; HU, 2008)	*IEEExplorer*	3
Research on the Bidding Information Integrated System of Generation Group (CHEN; XIAO; SONG, 2008)	*IEEExplorer*	3, 4
Multi-agent Based Power Grid Data Integration and Sharing Platform (WU; DUAN; SHI, 2009)	*IEEExplorer*	3, 4
Research on Grid-Service Based Virtual Products Experience Environment Supporting Architecture in E-commerce Applications (CHEN, 2009)	*IEEExplorer*	3, 4
Using Multiagent Self-organization Techniques to Improve Dynamic Skill Searching in Virtual Social Communities (MERCIER; OCCELLO; JAMONT, 2010)	*IEEExplorer*	2, 3
A Generic Recursive Multiagent Model to Simplify Large Scale Multi-level Systems Observation (HOANG; OCCELLO; JAMONT, 2011)	*IEEExplorer*	1, 3
Multi-agent System to Support Creative Workshop (GABRIEL et al., 2015)	*IEEExplorer*	3
A low-level resource allocation in an agent-based Cloud Computing platform (BAJO et al., 2016)	*Science Direct*	3
Comparison of tightly and loosely coupled decision paradigms in multiagent expedition (XIANG; HANSHAR, 2010)	*Science Direct*	3, 4
AEGIS: Agent oriented organisations (UNLAND et al., 1995)	*Science Direct*	2, 3, 4
Expectation-oriented modeling (NICKLES; ROVATSOS; WEISS, 2005)	*Science Direct*	3, 4
Multi-Agent System Development Based on Organizations (ARGENTE; JULIAN; BOTTI, 2006)	*Science Direct*	Nenhum

Tabela 7 – Critérios não atendidos por artigo.

6.2 Catálogo de padrões arquiteturais organizacionais

O catálogo a seguir baseia-se no trabalho de Argente, Julian e Botti (2006) entitulado *Multi-Agent System Development Based on Organizations*, bem como em outros estudos realizados e conhecimentos adquiridos pela autora dessa monografia junto à massa de artigos investigados e apresentados na Seção anterior. A Figura 13 representa as classificações de cada padrão dentro de categorias e subcategorias definidas, principalmente, pelos autores do referido artigo.

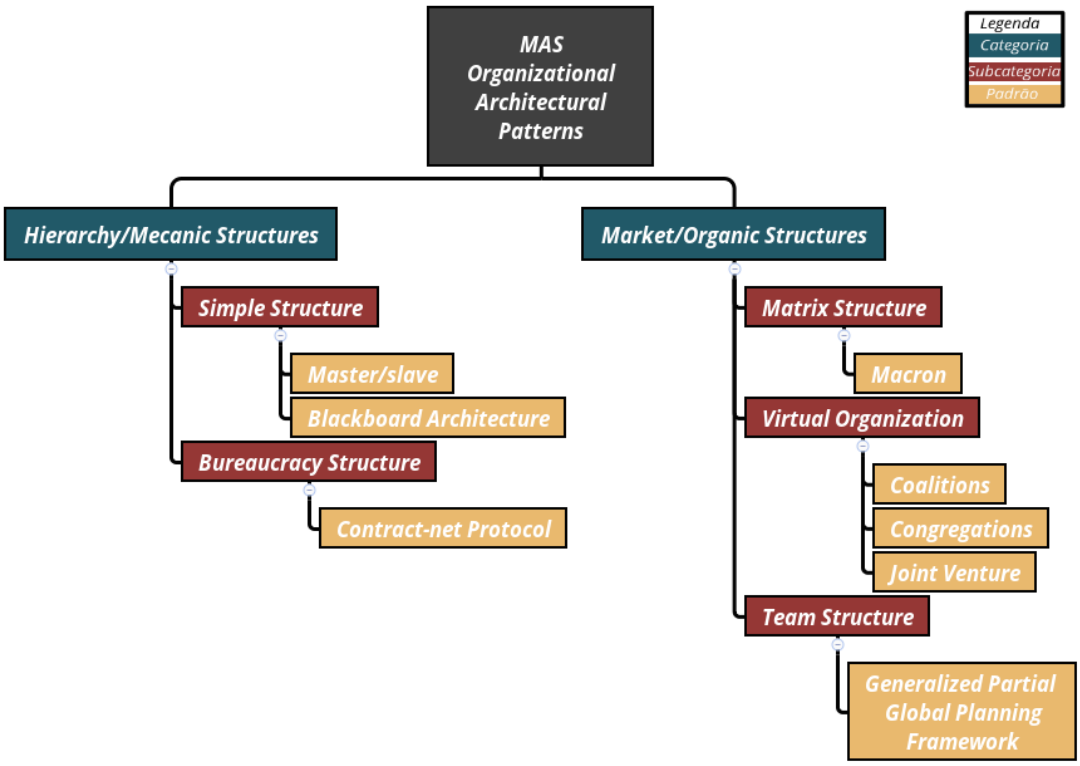

Figura 13 – *MAS - Organizational Architectural Patterns*. Fonte: Autora. Adaptado de: Argente, Julian e Botti (2006).

A implementação dos padrões pode ser encontrada no Apêndice B e no *GitHub*[1].

6.2.1 Considerações parciais

Neste capítulo foram abordados os resultados obtidos ao longo da realização desse trabalho. A Revisão Sistemática possibilitou que se identificasse os padrões arquiteturais organizacionais para SMA. A partir do trabalho de Argente, Julian e Botti (2006), obtido na Revisão Sistemática, pôde-se obter conhecimentos importantes para fomentar o catálogo. Por fim, o catálogo foi elaborado com foco em descrições teóricas e práticas, contendo modelagem, implementação e exemplos.

[1] https://github.com/tainarareis/MultiagentsSystemsCatalog (último acesso: Julho 2018

6.2.2 *Contract Net*

Este padrão foi apresentado como prova de conceito na primeira etapa de realização desse TCC. Para seu melhor entendimento, são apresentados os atos comunicativos FIPA (Seção 6.2.2.1). O *Contract Net* foi considerado como uma prova de conceito por permitir a condução de atividades críticas, inerentes a essa pesquisa, e que, potencialmente, poderiam resultar em insucessos, comprometendo a pesquisa como um todo.

O *Contract Net* é um protocolo que atende aos critérios definidos para caracterizar-se como padrão arquitetural organizacional. Trata-se de um protocolo para comunicação entre agentes voltado ao controle distribuído de tarefas em execução envolvendo negociação entre agentes. Seu detalhamento, enquanto padrão, é realizado a seguir, baseando-se no protocolo de comunicação *FIPA-Contract-Net*. Para maior entendimento dos atos comunicativos FIPA, a Seção 6.2.2.1 busca descrever os mesmos.

Nome do padrão: *Contract Net*.

Referências: Bellifemine, Caire e Greenwood (2007, pág. 23), Bellifemine et al. (2002, pág. 31).

Categoria: *Markets*.

Problema: um agente, o *Initiator* (ou iniciador), que deseja ter alguma tarefa executada por um ou mais agentes, os *Responders* (ou participantes), e ainda deseja otimizar uma função que caracteriza a tarefa. Esta função é comumente expressa, por exemplo, como custo, tempo até a conclusão, e a distribuição justa das tarefas (BELLIFEMINE; CAIRE; GREENWOOD, 2007).

Solução: para uma determinada tarefa, qualquer número de participantes pode responder com uma proposta; os demais devem recusar. As negociações prosseguem com os Participantes que propuseram. O Iniciador solicita um número m de propostas de outros agentes através da emissão de um convite à apresentação de propostas (*Call for Proposals*), que especifica a tarefa e quaisquer condições que o Iniciador coloca sobre a execução da tarefa.

Os participantes que recebem o convite à apresentação de propostas são vistos como potenciais contratados e são capazes de gerar um número n de respostas. Destes, j são o número de propostas para realizar a tarefa, especificada como proposta.

A proposta do Participante inclui as pré-condições que o Participante está definindo para a tarefa, que pode ser o preço, o tempo em que a tarefa será feita, dentre outras. Alternativamente, os Participantes podem se recusar a propor. Uma vez passado o prazo, o Iniciador avalia o valor recebido

Protocolos associados: o *FIPA-Contract-Net* está associado a este padrão e pode ser representado pela Figura 14.

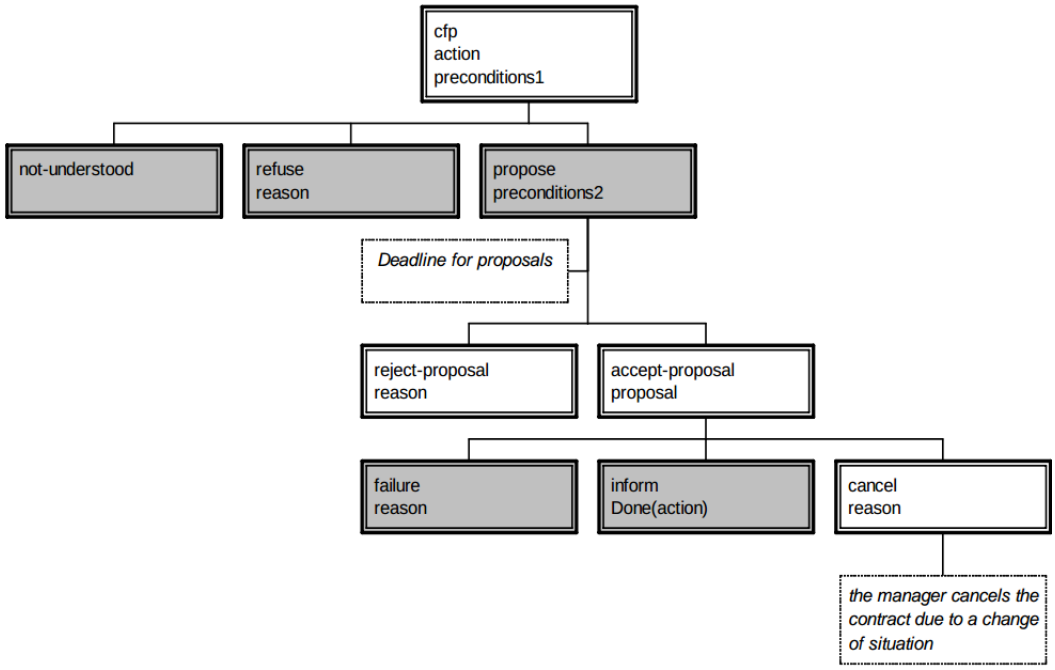

Figura 14 – Protocolo de Interação *FIPA-Contract-Net*. Fonte: Bellifemine et al. (2002, pág. 36)

O *FIPA-Contract-Net* permite que o *Initiator* envie um *Call for Proposal* (CFP) a um conjunto de *Responders*, avalie suas propostas e, por fim, realize sua escolha aceitando uma das propostas ou até mesmo rejeitando a todas (BELLIFEMINE et al., 2002).

A mensagem CFP contém a ação a ser realizada e, se necessário, condições de execução. Os *Responders* podem responder com as seguintes mensagens: *Propose* - sua proposta composta por pré-condições, custo e tempo -, *Refuse* para recusa - ou *Not-Understood* em caso de falhas de comunicação.

Após avaliar todas as propostas, o *Initiator* realiza sua escolha e informa quais foram rejeitadas e quais foram aceitas (através da mensagem *Accept Proposal*). Estes, assim que completam suas tarefas, respondem com *Inform* o resultado de sua ação - como finalizada ou como *Failure*.

O *Initiator* pode decidir cancelar o protocolo, enviando uma mensagem *Cancel*, antes da ação ter sido realizada e a última mensagem ter sido recebida.

O *FIPA-Contract-Net* é implementado por dois comportamentos: *ContractNetInitiator*[2] e *ContractNetResponder*[3]. O primeiro, atua sob o ponto de vista do *Initiator* e cuida do tempo limite de espera das propostas; além de prover os métodos *callback*

[2] http://jade.tilab.com/doc/api/jade/proto/ContractNetInitiator.html (último acesso: Junho 2017)
[3] http://jade.tilab.com/doc/api/jade/proto/ContractNetResponder.html (último acesso: Junho 2017)

para cada estado do protocolo. O comportamento *ContractNetResponder* atua sob o ponto de vista do *Responder*, e é responsável, principalmente, por avaliar a ação solicitada, enviar propostas ou recusar o envio de propostas.

Modelagem: este padrão pode ser representado pela Figura 15.

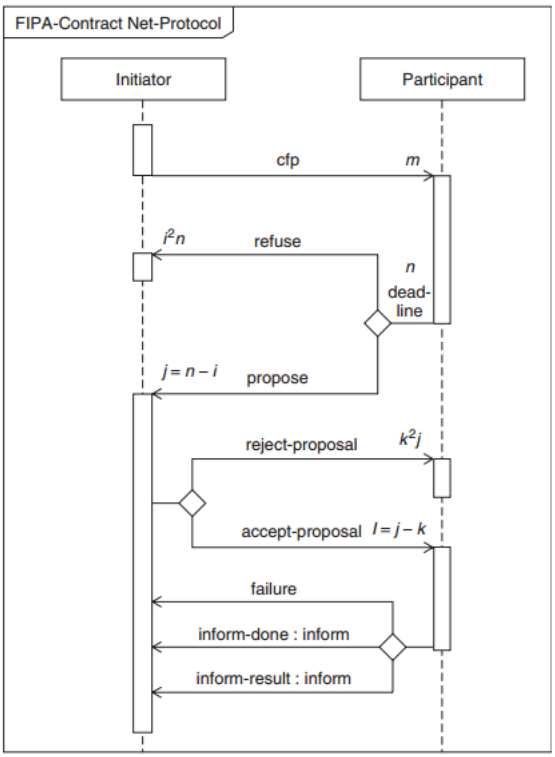

Figura 15 – Protocolo de interação *FIPA-Contract-Net*. Fonte: Bellifemine, Caire e Greenwood (2007, pág. 23).

Implementação: a fim de demonstrar o padrão *FIPA-Contract-Net*, é descrito o exemplo *Book Trading* fornecido pela plataforma JADE[4]. Todos os detalhes da implementação, configuração de ambiente e passos para execução são descritos no Apêndice B.1.

6.2.2.1 FIPA ACL

O FIPA-ACL baseia-se em mensagens que representam ações chamadas atos comunicativos (ou *communicative acts*), como apresenta a Figura 16. São definidos ao todo um conjunto de vinte e dois atos comunicativos (Tabela 8). Alguns dos atos mais comumente usados são *inform, request, agree, not understood*, e *refuse* . Estes capturam a essência da maioria das formas de comunicação básica (BELLIFEMINE; CAIRE; GREENWOOD, 2007, pág. 13). Segundo Bellifemine, Caire e Greenwood (2007), as normas FIPA estabelecem que um agente deve ser capaz de receber qualquer *communicative act* e, no mínimo, responder com uma mensagem *not-understood*, caso a mensagem recebida não possa ser

[4] http://jade.tilab.com/dl.php?file=JADE-examples-4.5.0.zip (último acesso: Junho 2017)

processada. Com base nestes atos comunicativos, o FIPA definiu um conjunto de protocolos de interação, consistindo cada um de uma sequencia de atos comunicativos para coordenar ações multi-mensagem, como o *FIPA-Contract-Net* (Seção 6.2.2) para o estabelecimento de acordos e vários tipos de leilões (BELLIFEMINE; CAIRE; GREENWOOD, 2007, pág 14).

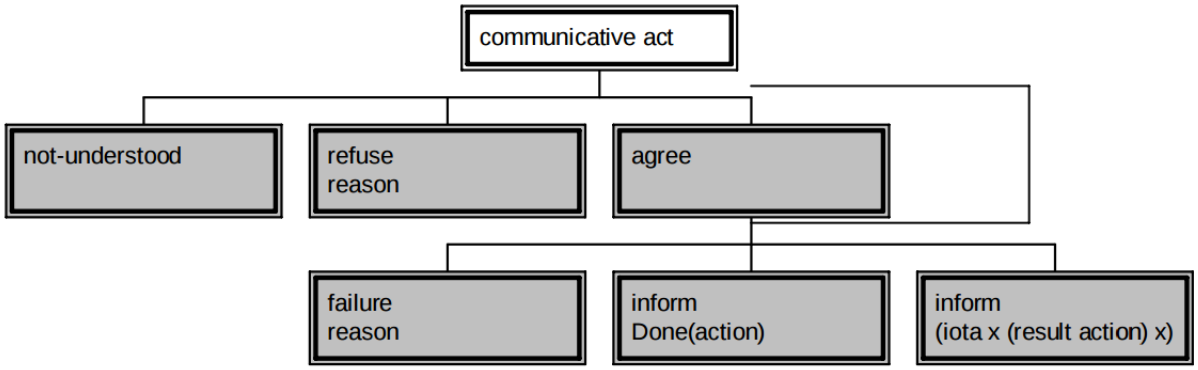

Figura 16 – *Communicative acts*. Fonte: Bellifemine et al. (2002, pág. 32).

6.3 *Master-Slave*

Este padrão apresenta um sistema regido por um mestre capaz de obter informações de agentes do grupo, criar planos e atribuir tarefas a agentes individuais, a fim de assegurar a coerência global.

Nome do padrão: *Master-Slave*.

Referências: Aridor e Lange (1998).

Categoria: *Simple structure*.

Problema: ocorre quando um agente mestre, (ou *Master*), precisa executar uma tarefa em paralelo com outras tarefas para as quais é responsável. Ocorre também quando este agente deseja executar uma tarefa em um destino remoto.

Solução: Quatro classes participam deste padrão. A Figura 17 ilustra suas relações estruturais.

- ***Master*:** define um esqueleto de um agente mestre, usando métodos abstratos para serem substituídos na classe *Concrete-Master*.
- ***Slave*:** define um esqueleto de um agente escravo, usando métodos abstratos para serem sobrescritos pela classe *ConcreteSlave*.
- ***ConcreteMaster*:** implementa métodos abstratos da classe *Master*.

Tabela 8 – Atos comunicativos FIPA. Traduzido. Fonte: Bellifemine, Caire e Greenwood (2007, pág. 19).

FIPA *communicative act*	**Descrição**
Accept Proposal	A ação de aceitar uma proposta previamente submetida para realizar ação.
Agree	A ação de concordar em realizar alguma ação, possivelmente no futuro.
Cancel	A ação de um agente informando outro agente de que o primeiro agente não tem mais a intenção de que o segundo agente execute alguma ação.
Call for Proposal	A ação de convocação de propostas para executar uma determinada ação.
Confirm	O remetente informa o receptor que uma dada proposição é verdadeira, onde o receptor é conhecido por ser incerto sobre a proposição.
Disconfirm	O remetente informa o receptor de que uma dada proposição é falsa, onde o receptor é conhecido por acreditar, ou acredita que é provável que, a proposição seja verdadeira.
Failure	A ação de dizer a outro agente que houve a tentativa de realizar uma ação, mas a tentativa falhou.
Inform	O remetente informa o receptor que uma dada proposição é verdadeira.
Inform If	Uma ação macro para o agente da ação para informar ao destinatário se uma proposição é verdadeira ou não.
Inform Ref	Uma ação macro que permite ao remetente informar o receptor de algum objeto acreditado pelo remetente para corresponder a um descritor específico, por exemplo, um nome.
Not Understood	O remetente do ato (por exemplo, i) informa o receptor (por exemplo, j) que percebeu que j realizou alguma ação, mas que i não entendeu o que j acabou de fazer. Um caso particular comum é quando i diz a j que i não entende a mensagem que j acaba de enviar para i.
Propagate	O remetente tem a intenção de que o receptor trate a mensagem incorporada como enviada diretamente para o receptor, e quer o receptor identifique os agentes denotados pelo descritor informado e envie a mensagem *propagate* recebida a eles.
Propose	A ação de perguntar a outro agente se uma determinada proposição é verdadeira ou não.
Proxy	O remetente deseja que o receptor selecione agentes de destino indicados por uma dada descrição e que ele envie uma mensagem incorporada para eles.
Query If	A ação de submeter uma proposta para executar uma determinada ação, dadas certas pré-condições.
Query Ref	A ação de perguntar a outro agente para o objeto referido por uma expressão referencial.
Refuse	A ação de recusar a execução de uma determinada ação, e explicando o motivo da recusa.
Reject Proposal	A ação de rejeitar uma proposta de realização de alguma ação durante uma negociação.
Request	O remetente solicita ao receptor que execute alguma ação. Um exemplo importante de usos do ato de *request* é solicitar ao receptor que execute outro ato comunicativo.
Request When	O remetente quer que o receptor execute alguma ação quando alguma proposição dada se torna verdadeira.
Request Whenever	O remetente quer que o receptor execute alguma ação assim que alguma proposição se tornar verdadeira e depois cada vez que a proposição se tornar verdadeira novamente.
Subscribe	O ato de solicitar uma intenção persistente de notificar o remetente do valor de uma referência e de notificar novamente sempre que o objeto identificado pela referência muda.

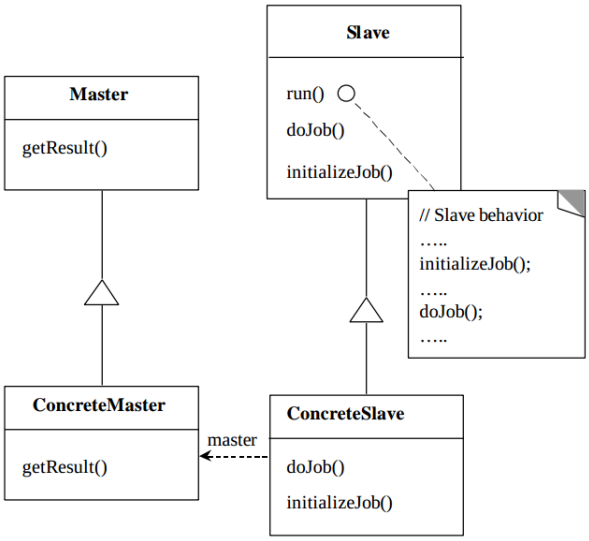

Figura 17 – Participantes do padrão *Master-Slave*. Fonte: Aridor e Lange (1998)

- **ConcreteSlave**: implementa métodos abstratos da classe *Slave*.

Primeiramente, o agente *Master* cria um agente *Slave*. Este, move-se para um *host* remoto e executa a tarefa para o qual foi designado. Em seguida, retorna com o resultado da tarefa para o agente *Master*.

Modelagem: este padrão pode ser representado pela Figura 18.

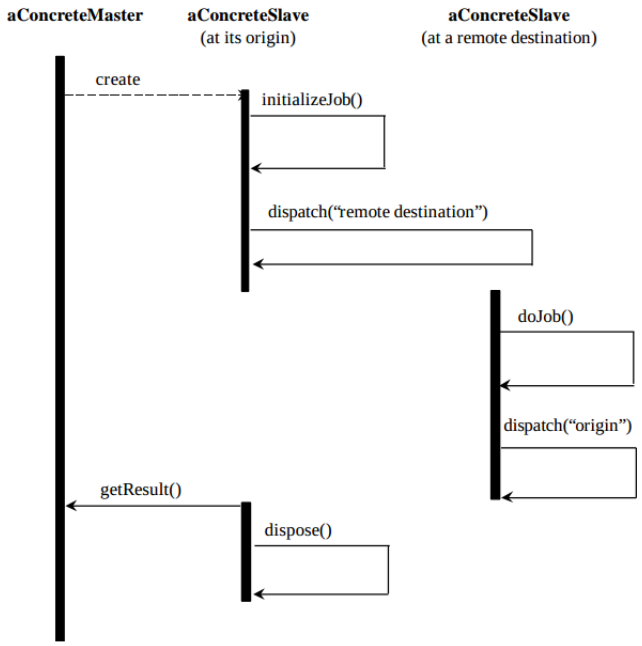

Figura 18 – Diagrama de sequência do padrão *Master-Slave*. Fonte: Aridor e Lange (1998).

Exemplo: O exemplo a seguir baseia-se em um aplicativo que fornece uma GUI para inserir dados e exibir os resultados intermediários de uma tarefa específica a ser realizada remotamente.

Com um único agente para fornecer a GUI e executar essa tarefa, não será possível manter a GUI depois que o agente viajou de sua origem para um destino remoto. A alternativa é adotar um agente escravo para que se mova para outro destino, execute a tarefa atribuída, envie os resultados intermediários e, finalmente, retorne com o resultado da tarefa ao agente mestre. Este, por sua vez, exibe o resultado para seu cliente.

A ideia chave do padrão é usar classes abstratas, mestre e escravo, para localizar as partes invariantes de delegar uma tarefa entre agentes mestres e escravos, consistindo em três principais comportamentos: (i) despachar um escravo de um lado para outro para outros destinos; (ii) iniciar a execução da tarefa, e (iii) lidar com exceções ao executar a tarefa.

Os agentes mestres e escravos são definidos como subclasses de mestre e escravo. Apenas partes variáveis, como a descrição de como a tarefa deve ser executada e como o agente mestre deve lidar com o resultado da tarefa, são implementadas. Na prática, a classe *Master* possui um método abstrato *getResult()* para definir como lidar com o resultado da tarefa.

A classe *Slave* tem dois métodos abstratos, *initializeJob()* e *doJob()*, que definem as etapas de inicialização a serem executadas antes que o agente viaje para um novo destino e a tarefa a ser executada, respectivamente. Ambas as classes são definidas em termos desses métodos. O código a seguir mostra a classe *Slave* implementada como um *Aglet*.

```
1  public abstract class Slave extends Aglet {
2      Object result = null
3
4      public void onCreation(Object obj) {
5          // Called when the slave is created.Gets the
6          // remote destination, a reference to the master
7          // agent, and other specific parameters.
8      }
9
10     public void run () {
11         // At the origin:
12         initializeJob();
13         dispatch(destination); // Goes to destination
14         // At the remote destination:
15         doJob(); // Starts on the task.
16         result=...;
17         // Returns to the origin.
18         // Back at the origin.
19         // Delivers the result to the master and dies.
20         dispose();
21 }
```

Implementação: a fim de demonstrar este padrão, é utilizado um contexto de fechamento de caixa de uma loja. Todos os detalhes da implementação, configuração de

ambiente e passos para execução são descritos no Apêndice B.2.

6.4 *Blackboard*

O padrão arquitetural *Blackboard* (ou Quadro-Negro) tem sido amplamente utilizado para enfrentar problemas com características de incerteza (DONG; CHEN; JENG, 2005). Muitas vezes, não há solução algorítmica direta para problemas dessa natureza e a melhor solução trata-se de uma aproximação.

Esta arquitetura se baseia em um repositório central, onde agentes leem e escrevem em um quadro-negro, e um agente mestre controla o que é lido e escrito por estes agentes. As aplicações deste padrão podem ser encontradas em diferentes tipos de sistemas, como sistemas de comunicação, mobilidade e coordenação.

Nome do padrão: *Blackboard* ou Quadro-Negro.

Referências: Dong, Chen e Jeng (2005), Weiss (1999), Ito e Salleh (2000).

Categoria: *Simple Structure*.

Problema: é adotado em problemas com características de incerteza (DONG; CHEN; JENG, 2005), onde não há solução algorítmica direta para o problema e a melhor solução trata-se de uma aproximação. É adotado também para possibilitar o acesso paralelo a dados e a execução simultânea de tarefas.

Solução: Weiss (1999) apresenta a metáfora a seguir para demonstrar este padrão.

> "Imagine um grupo de especialistas humanos ou de agentes sentados ao lado de um grande quadro-negro. Os especialistas estão trabalhando cooperativamente para resolver o problema, usando o quadro-negro como local de trabalho para desenvolver a solução. A resolução de problemas começa quando o problema e a data inicial estão escritos no quadro-negro. Os especialistas observam o quadro-negro, procurando uma oportunidade para aplicar seus conhecimentos à solução em desenvolvimento. Quando um especialista encontra informações suficientes para fazer uma contribuição, ele registra a contribuição no quadro-negro. Esta informação adicional pode permitir que outros especialistas apliquem seus conhecimentos. Este processo de adicionar contribuições para o quadro-negro continua até o problema ter sido resolvido."(WEISS, 1999)

Esta metáfora captura uma série de características importantes deste padrão, cada uma das quais são descritas a seguir.

- **Independência da experiência**: os especialistas (chamados de fontes de conhecimento, *Knowledge Sources* ou KSs) não são treinados para trabalhar unicamente com esse grupo específico de especialistas. Cada um é especialista em

alguns aspectos do problema e pode contribuir para a solução independentemente da combinação particular de outros especialistas na sala.

- **Diversidade em técnicas de resolução de problemas**: o algoritmo interno de representação e inferência adotado por cada KS é ocultado;
- Representação flexível da informação do quadro-negro: não há quaisquer restrições prévias sobre as informações que podem ser colocadas no quadro-negro;
- **Linguagem de interação comum**: os KS devem ser capazes de interpretar corretamente as informações gravadas no quadro-negro por outros KSs. Pode haver tanto uma representação compreensível apenas entre alguns KSs e uma representação genérica compreensível por todos os KSs;
- **Ativação baseada em eventos**: os KSs são acionados em resposta ao quadro-negro e aos eventos externos. Os eventos do quadro negro incluem a adição de novas informações ao mesmo, uma alteração na informação existente ou a remoção de informações existentes. Cada KS informa o sistema de quadro-negro sobre o tipo de eventos em que ele está interessado. O sistema de quadro-negro grava esta informação e considera diretamente o KS a ser ativado sempre que esse tipo de evento vier a ocorrer;
- **Necessidade de controle**: há um componente de controle que é separado dos KSs e é responsável por gerenciar o fluxo da resolução de problemas. O componente de controle pode ser visto como um especialista na resolução de problemas. Ele analisa o benefício geral das contribuições possíveis pelos KSs para a resolução do problema em questão. Ao ser finalizada a execução de um KS, o componente de controle ativa o KS mais apropriado para execução. Ao ser disparado, o KS utiliza seus conhecimentos para avaliar a qualidade e a importância de sua contribuição. Cada KS disparado informa o componente de controle sobre a qualidade e os custos associados à sua contribuição, sem realmente realizar o trabalho para calcular a contribuição. O componente de controle usa essas estimativas para decidir como proceder, e
- **Geração de solução incremental**: KSs contribuem para a solução conforme apropriado, às vezes refinando, às vezes contradizendo, e às vezes iniciando uma nova linha de raciocínio.

Há, portanto, três componentes principais neste padrão (DONG; CHEN; JENG, 2005):

1. **Fontes de conhecimento (ou *Knowledge Sources*)**: cada KSs é um especialista na resolução de certos aspectos do problema geral. Uma vez que ele encontra a informação que precisa no quadro-negro, pode prosseguir sem qualquer ajuda de outras fontes de conhecimento;

2. *Blackboard* (ou **Quadro-negro**): o quadro-negro é a fonte de todos os dados em que a fonte de conhecimento irá operar e é o destino para todas as conclusões a partir de uma fonte de conhecimento, e

3. **Controle**: é um gerente que considera cada solicitação feita à fonte de conhecimento. O controle é responsável por abordar o quadro-negro em termos do que a fonte de conhecimento pode contribuir e o efeito que a contribuição pode ter sobre a solução a ser proposta. O controle tenta manter a resolução de problemas dentro do fluxo correto, de modo a garantir que todos os aspectos cruciais do problema recebam atenção e a equilibrar a importância estabelecida das contribuições dos diferentes especialistas. Os componentes de controle também são fontes de conhecimento, entretanto, se concentram no processo de solução de problemas de domínio específicos.

Modelagem: A Figura 19 representa a arquitetura básica deste padrão.

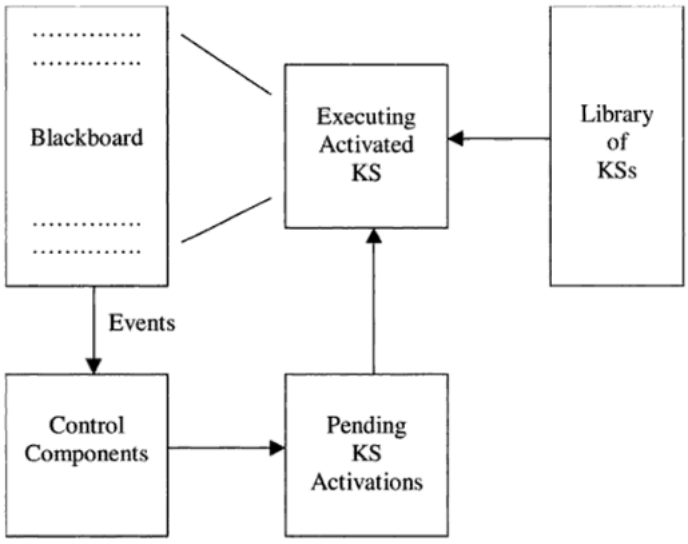

Figura 19 – A arquitetura básica de um sistema *Blackboard*. Fonte: Weiss (1999).

Exemplo: De modo a exemplificar uma utilização do padrão *Blackboard*, é apresentado o *Collaborative Supply Chain System* (CSCS) descrito por Ito e Salleh (2000). Trata-se de um sistema de gerenciamento de cadeia de suprimentos compreendido em uma rede integrada de membros, são eles: fornecedores, fábricas, armazéns, centros de distribuição e varejistas. Devido à complexidade desta rede, toda a cadeia de processos logísticos necessita ser gerenciada.

A colaboração entre os membros desempenha um papel crítico para implementar um sistema eficaz, mas sua implementação não seria fácil apenas com o mecanismo convencional de compartilhamento de informações. O intuito é proporcionar uma coordenação mais rápida e mais visível entre uma empresa, seus clientes e forne-

cedores. O CSCS é projetado para aumentar a velocidade e a certeza da oferta de suprimentos.

Conforme mostrado na Figura 20, cada agente interage e compartilha informações através de um quadro-negro. O quadro-negro regula e fornece informação a cada membro do sistema, são eles: *Supplier Agent* (SA), *Manufacturer Agent* (MA), *Distributor Agent* (DA) e *Customer Agent* (CA). A colaboração destes membros e a introdução de técnicas de negociação são desempenhadas por duas redes:

- *Intranet*: pode ser usada de modo a fornecer serviços de comunicação interna para obter melhores resultados do que os meios convencionais de acesso e transferência de dados. Por meio do *Manufacturer Agent*, os usuários acessam informações diárias, e a empresa pode facilmente exibir informações críticas de usuários externos. Basicamente, o *Manufacturer Agent* atua no controle de inventário regulando os níveis de inventário e negocia com os outros agentes para facilitar a circulação de materiais. A distribuição destes materiais é designada pelo *Distributor Agent*, e
- *Internet*: é uma base unificada composta por todos os membros do sistema e que permite que os usuários se comuniquem para as trocas de informações. Por exemplo, a *Internet* pode ser usada para oferecer a oportunidade de comparar fornecedores, escolher um fornecedor adequado e garantir que o fornecedor selecionado satisfaça os requisitos.

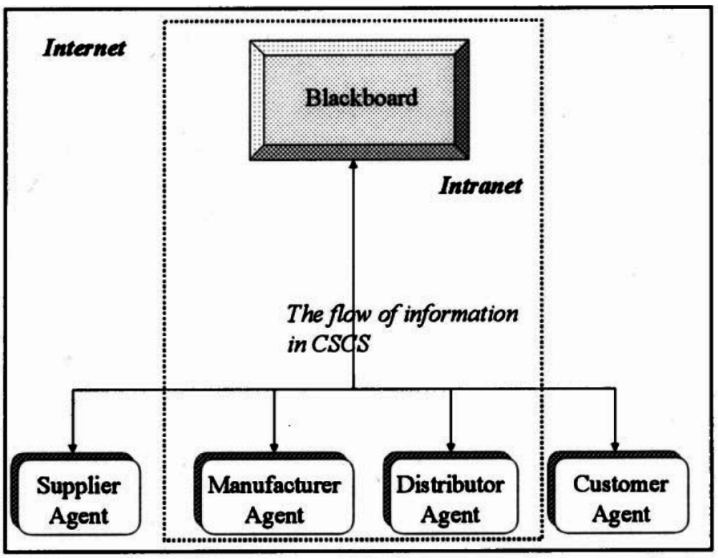

Figura 20 – Interação entre agentes no CSCS. Fonte: Ito e Salleh (2000).

As empresas são obrigadas a cumprir as ordens dos clientes mesmo quando é difícil fazê-lo. Elas devem responder às ordens de forma rápida e eficiente dentro do tempo limitado e disponível para atender às solicitações de suprimentos de seus clientes.

Ordens que surgem de modo imprevisto e com urgência, quando ocorrem, causam atraso na entrega e diminuem a eficácia dos membros do sistema. A colaboração dos membros do CSCS e a introdução de técnicas de negociação fornecem uma solução para esses problemas.

Suponha-se que para a reposição de peças e materiais, uma empresa publica um leilão no quadro-negro. Este leilão pode ser visualizado por toda a comunidade de fornecedores, caracterizando-se, portanto, como uma competição aberta.

Os agentes fornecedores, *Supplier Agents*, de cada empresa fornecedora, exploram a Internet para encontrar os leilões de seu interesse. Quando um agente encontra um leilão de seu interesse, ele indica sua respectiva companhia fornecedora.

Em seguida, o quadro-negro publica os lances apresentados por estas companhias. Os membros realizam revisões dos lances ou negociam com essas empresas. Logo após, o lance mais apropriado é selecionado com base em critérios de seleção.

Se por ventura um problema de entrega ocorre com um dos fornecedores, os agentes colaboram e negociam entre si para substituí-lo. O fornecedor alternativo é escolhido de acordo com os critérios de seleção preparados pelo fabricante (*Manufacturer Agent*). A Figura 21 mostra como esses tipos de atividades ocorrem quando o fornecedor selecionado não entregar os materiais no momento certo. Nessa atividade, o fornecedor em questão, o fabricante e o fornecedor alternativo são controlados pelo *Supplier Agent* 1 (SA1), *Manufacturer Agent* (MA) e *Supplier Agent* 2 (SA2), respectivamente.

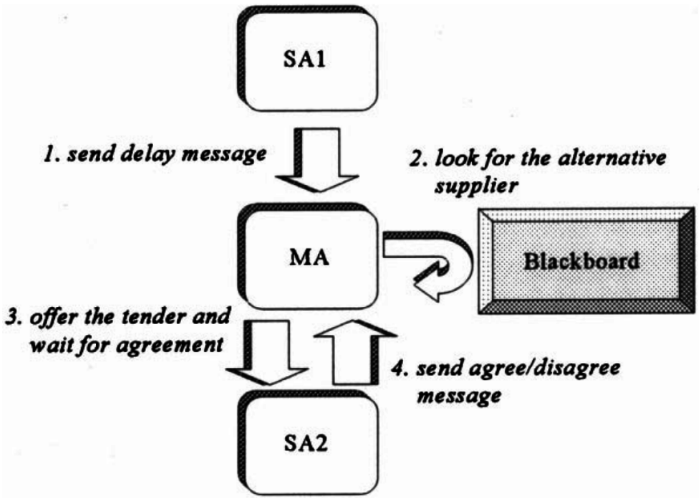

Figura 21 – Processo de colaboração e negociação para substituição de fornecedor. Fonte: Ito e Salleh (2000).

Quando o problema acontece, o SA1 envia uma mensagem de atraso para o MA. Depois de receber a mensagem de atraso, o MA refere-se ao BB para fornecer um fornecedor alternativo para a substituição. Se o SA2 for considerado o fornecedor

alternativo, o MA faz uma consulta ao SA2 e solicita a estimativa de custo da entrega.

Então, o SA2 cita o custo dos materiais e o tempo da entrega estimado e os envia de volta ao MA. Se o SA2 não aceitar a oferta, o MA executará o mesmo procedimento para outro fornecedor alternativo. E deste modo, são realizadas as atividades de colaboração e negociação entre esses agentes, ajudando fabricantes e fornecedores a resolver os problemas durante o processo de entrega.

Implementação: a fim de demonstrar o padrão *Blackboard*, é utilizado o contexto acima para simular a entrega de materiais. Todos os detalhes da implementação, configuração de ambiente e passos para execução são descritos no Apêndice B.3.

6.5 *MACRON*

O padrão MACRON (*Multiagent Architecture for Cooperative Retrieval ONline*) apresenta um sistema de aquisição de informações (DECKER et al., 1995). Nesta organização, é elaborado um planejamento único que gera sub-objetivos a serem alcançados por um grupo de agentes que cooperam entre si.

A cooperação entre agentes implica o gerenciamento de interdependências entre suas tarefas. Ao interagirem, os agentes integram e desenvolvem agrupamentos consistentes de informações de alta qualidade a partir de fontes heterogêneas distribuídas (DECKER et al., 1995).

Tais agentes compartilham informações um com o outro de modo a saberem (i) quais os planos que eles executam para alcançar seus sub-objetivos, (ii) em qual ordem executam tarefas, e (iii) quando as executam. Além disso, os agentes são livres para desenvolver diferentes planos de coleta de informações, a depender da quantidade de tempo que possuem para produzir uma resposta (DECKER et al., 1995).

Nome do padrão: MACRON, *Multiagent Architecture for Cooperative Retrieval ONline*.

Referências: Decker et al. (1995) e Shehory (1998).

Categoria: *Matrix structure*. Isso se deve ao fato de que este padrão possui uma partição bidimensional para unidades: unidades funcionais e unidades de resposta de consulta, a serem descritas a seguir.

Problema: ocorre quando um informação não pode ser simplesmente recuperada, mas adquirida através de um processo dinâmico, incremental e limitado pelos recursos disponíveis (DECKER et al., 1995).

Solução: O padrão MACRON possui os seguintes objetivos: (i) explorar interdependências entre problemas, (ii) gerenciar a incerteza inerente à busca, (iii) compensar inteligentemente a qualidade da solução devido às limitações de recursos, e (iv) explorar ou evitar redundância, conforme necessário (DECKER et al., 1995).

De acordo com Shehory (1998), o sistema consiste nos seguintes agentes ou grupo de agentes:

- ***Query-Manager*** (QM) ou gerente de consulta: recebe uma consulta de um usuário, desenvolve um plano inicial de levantamento de informações de alto nível, recruta agentes das *Functional Units* (FUs) necessárias através dos *Functional Managers* (FMs) para formar uma *Query-Answering Unit* (QAU) e monitora a execução do plano;
- ***Functional Unit*** (FU) ou unidade funcional: uma coleção de agentes que têm acesso a um determinado tipo de recursos de informação. Cada unidade funcional possui um *Functional Manager*;
- ***Functional Manager*** (FM) ou gerente funcional: a pedido de um *Query Answering Agent*, o agente FM atribui tarefas a agentes dentro de sua FU. Embora seja para o mesmo tipo de fontes de informação, os agentes dentro de um FU específica podem ter diferentes conhecimentos. O FM leva essas diferenças em conta ao planejar a tarefa a ser atribuída;
- ***Functional Agent*** (FA) ou agente funcional: planeja a recuperação de informações, solicita-as e as manipula;
- ***Low Level Information Agents*** (LLIA) ou agentes de informação de baixo nível: são simples agentes, cada um especializado em um tipo específico de informações de recuperação. Tal agente recupera as informações solicitadas por um FA, e
- ***Organization Chart Manager*** (OCM) ou gerenciador gráfico da organização: é uma memória organizacional onde os agentes podem procurar informações sobre a disponibilidade e as capacidades de outros agentes. Novos agentes são adicionados ao OCM quando eles se juntam ao sistema dinamicamente, durante o tempo de execução. QMs, FMs e agentes funcionais todos consultam o OCM para localizar os agentes apropriados aos quais eles têm que delegar tarefas.

Na Figura 22, são apresentados os componentes do MACRON e as relações entre eles. Alguns detalhes foram omitidos pelo autor, como a comunicação bidirecional de agentes FA com o OCM e de agentes FM com agentes FAs em sua unidade funcional. Cabe ressaltar que uma vez que o QAU foi formado, o monitoramento das FAs é feito pelo QM responsável pela consulta e não pelo FM.

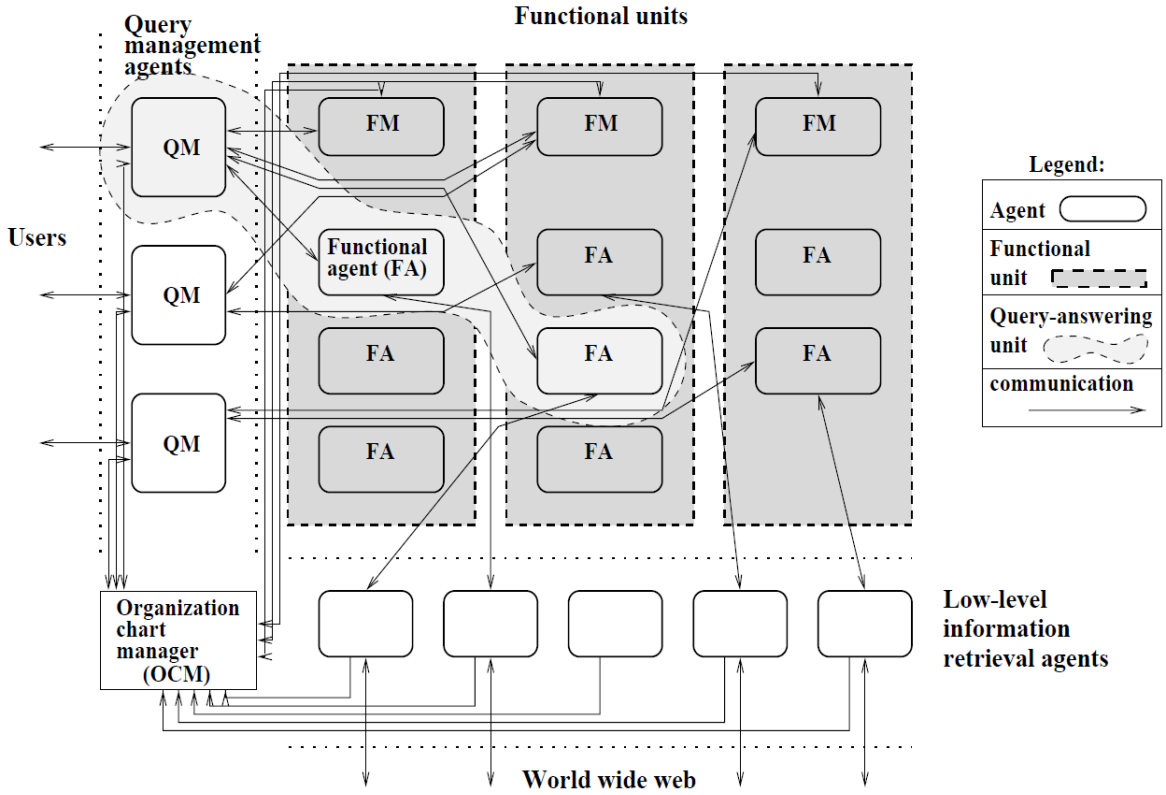

Figura 22 – Organização MACRON. Fonte: Shehory (1998).

Do ponto de vista organizacional, os agentes no MACRON formam uma organização matricial, consistindo em unidades funcionais e de resposta de consulta como apresentado na Figura 23. Em uma organização matricial humana, as pessoas pertencem a um grupo funcional de longo prazo e curto prazo. Por exemplo, em um ambiente de desenvolvimento de software, existe um grupo de *designers* de interface, um grupo de desenvolvedores de software e outro grupo de *designers* de *hardware*, consistindo em grupos de longo prazo. De modo dinâmico, pessoas destes grupos são alocadas para formar um grupo para um projeto de curto prazo. Tais organizações proporcionam flexibilidade ao sistema, em ambientes com mudanças incertas e dinâmicas, permitindo ainda a atribuição e o rastreamento de recursos limitados entre os agentes.

Modelagem: este padrão pode ser representado pela Figura 22.

Exemplo: o exemplo a seguir baseia-se em um sistema (Figura 23) capaz de recuperar *reviews* de usuários na internet a cerca de um determinado produto.

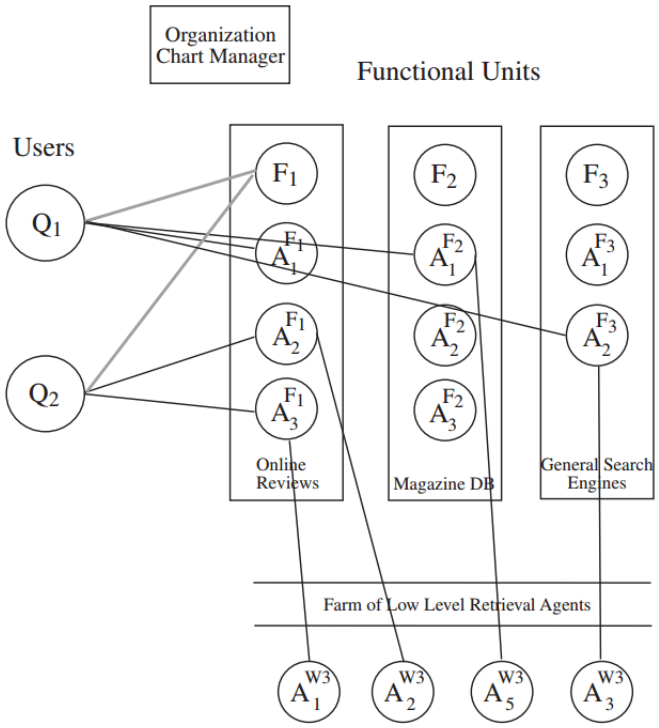

Figura 23 – Exemplo de estrutura MACRON. Fonte: Decker et al. (1995).

Na Figura 24, o agente QM pode expandir o objetivo inicial *Get Review* para *Get Online Reviews* e *Get Published Reviews*. Ele, então, se comunica com agentes FM para obter *Online Reviews* e *Published Reviews*. Os agentes FM, por sua vez, alocam os objetivos *Get Online Reviews* e *Get Published Reviews* aos agentes das respectivas unidades funcionais (FUs).

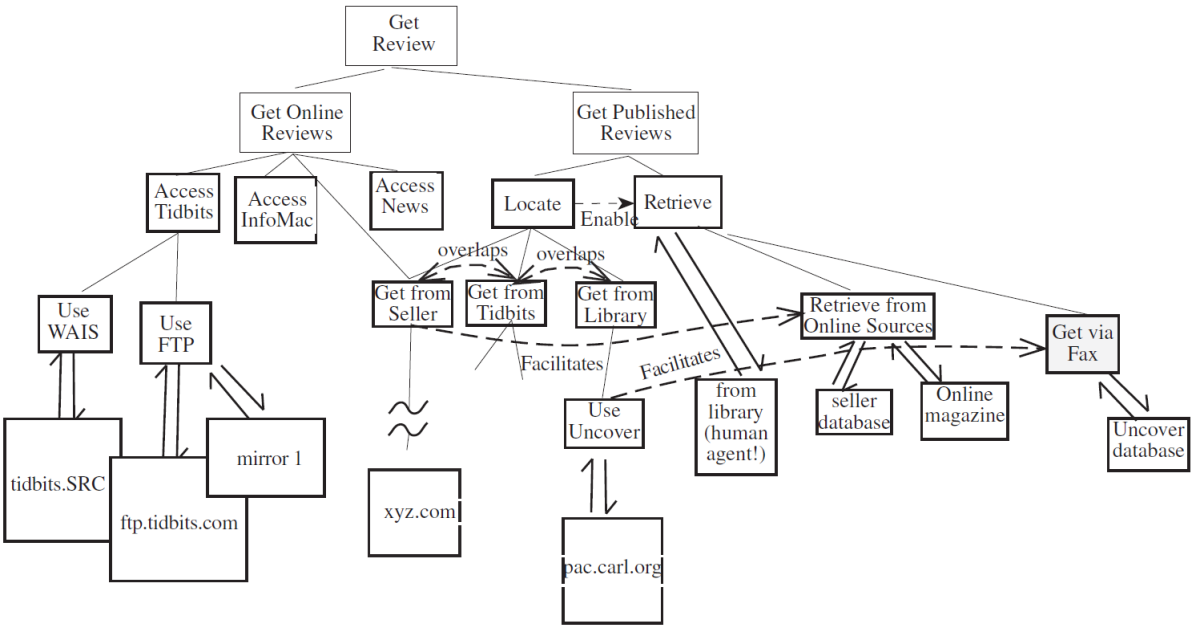

Figura 24 – Árvore de tarefas para recuperar *reviews* sobre um produto. Fonte: Decker et al. (1995).

Suponha-se que cada objetivo é alocado para um agente. Os agentes expandem seus objetivos para alcançar as tarefas primitivas de nível mais baixo, como *Use Wais*, *Use FTP* e *Retrieve Online Magazine*. O agente de consulta verifica o OCM para encontrar o endereço do FM cujas FUs possam obter desempenhar a tarefa *Get Online Reviews*.

Para melhor compreensão da utilidade da estrutura MACRON, a seguir ela será comparada, brevemente, à duas abordagens alternativas. Em uma abordagem de organização de **agentes com funções fixas**, teríamos equipes de agentes, cada equipe consistindo de pelo menos um membro de cada FU. Essas equipes fixas causam vários problemas: como todas as consultas não precisarão de todos os recursos funcionais, alguns membros da equipe ficarão ociosos. Por outro lado, se um membro da equipe necessário estiver temporariamente indisponível, uma consulta poderá falhar. Em terceiro lugar, à medida que forem introduzidos componentes de aprendizagem nos agentes, as equipes começam a diferenciar e desenvolver diferentes especializações em cada uma de suas áreas funcionais. Se o ambiente é dinâmico, pode-se esperar que surjam novas consultas que seriam mais bem respondidas por combinações de agentes experientes que diferem das equipes fixas oferecidas.

Uma segunda abordagem para se comparar com a estrutura MACRON, seria o outro extremo da abordagem anterior, com uma variedade de **agentes sem funções fixas**, onde um agente de consultas pode reunir uma equipe de agentes para lidar com uma consulta específica. Neste caso, não há recursos desperdiçados, e não há problemas se um único agente estiver temporariamente indisponível. Este sistema é mais flexível do que a organização funcional fixa.

No entanto, essa abordagem falha em fornecer métodos simples para reunir a melhor equipe possível, especialmente em um sistema cooperativo. São duas importantes perguntas que o agente responsável pelas consultas deve responder:

- Qual dos agentes que anunciam um serviço ele deve realmente entrar em contato e utilizar seu serviço?
- À medida que os agentes aprendem e se diferenciam, como eles podem comparar suas habilidades relativas sem se conhecerem?

Na organização MACRON, a escolha de um agente funcional é de responsabilidade exclusiva dos gerentes funcionais; que atuam como facilitadores especializados e inteligentes. O MACRON permite o melhor das duas abordagens anteriores: a flexibilidade de montar uma equipe eficiente e a capacidade de atribuir e rastrear de forma inteligente e dinâmica esses recursos do agente.

Implementação: a fim de demonstrar o padrão MACRON, é utilizado o contexto do exemplo acima. Todos os detalhes da implementação, configuração de ambiente e passos para execução são descritos no Apêndice B.4.

6.6 *Generalized Partial Global Planning*

O padrão *Generalized Partial Global Planning* (GPGP) atua na coordenação de tarefas locais de cada agente considerando objetivos locais e não locais (DECKER; LESSER, 1992).

Nome do padrão: *Generalized Partial Global Planning* (GPGP) ou Planejamento Global Parcial Generalizado.

Referências: Weerdt e Clement (2009), Decker e Lesser (1992).

Categoria: *Team Structure.*

Problema: é adotado para possibilitar a coordenação distribuída; que pode ser descrita como o agendamento das atividades locais de cada agente considerando preocupações e restrições não locais (DECKER; LESSER, 1992).

Solução: Para abordar o padrão GPGP, será primeiramente apresentada a estrutura PGP exemplificada pela Figura 25. O *Agent 1* possui o objetivo global de completar a tarefa *A1*, e duas sub-tarefas concorrentes, *B1* e *C1*. O *Agent 2* possui o objetivo de completar a tarefa *A2*, e duas sub-tarefas *B2* e *D2* (DECKER; LESSER, 1992).

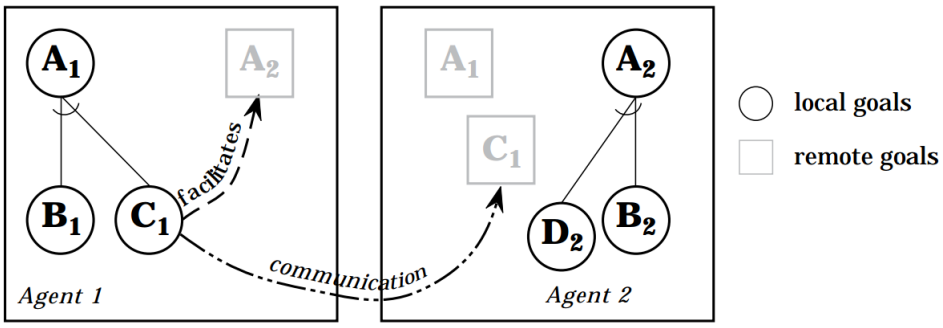

Figura 25 – Exemplo de estrutura PGP. Fonte: Decker e Lesser (1992).

No algoritmo PGP, os agentes *Agent 1* e *Agent 2* vão trocar seus objetivos de mais alto nível, chamados *System Goals*. Havendo ciência desta informação, o *Agent 1* pode determinar que sua sub-tarefa local C1 facilita a tarefa A2 (DECKER; LESSER, 1992).

Isto indica que a tarefa C1, de algum modo, é útil para que se complete A2. Informações sobre o status da sub-tarefa C1 são enviadas ao *Agent 2* permitindo-o

rearranjar sua programação para lidar com informações futuras. Similarmente, o fato de C1 ser uma tarefa facilitadora, torna-a preferível. Por exemplo, se o *Agent 1* possuir duas tarefas concorrentes, sendo uma delas *C1*, e não havendo outra razão para dar preferência à outra, o algoritmo PGP agendará *C1* para ocorrer primeiro (DECKER; LESSER, 1992).

A estrutura do PGP não foi projetada apenas para permitir a comunicação de informações necessárias para o planejamento global, mas também para o projeto e análise de algoritmos de coordenação para agentes que operam em ambientes com características diferentes daquelas para as quais o PGP foi projetado . Por exemplo, podem haver agentes heterogêneos - que podem ter diferentes critérios locais de solução de problemas -, agentes dinâmicos - que possuem várias estratégias e métodos diferentes disponíveis para a realização de metas e um conjunto de compensações entre eles - e em tempo real - agentes que podem ter prazos rígidos ou flexíveis (DECKER; LESSER, 1992).

O Planejamento Global Parcial (PGP) é uma abordagem flexível à coordenação distribuída que permite aos agentes responder dinamicamente à sua situação atual. O GPGP tenta estender a abordagem PGP comunicando informações mais abstratas e organizadas hierarquicamente, detectando de maneira geral as relações de coordenação necessárias aos mecanismos de planejamento global parcial e separando o processo de coordenação do planejamento local (DECKER; LESSER, 1992).

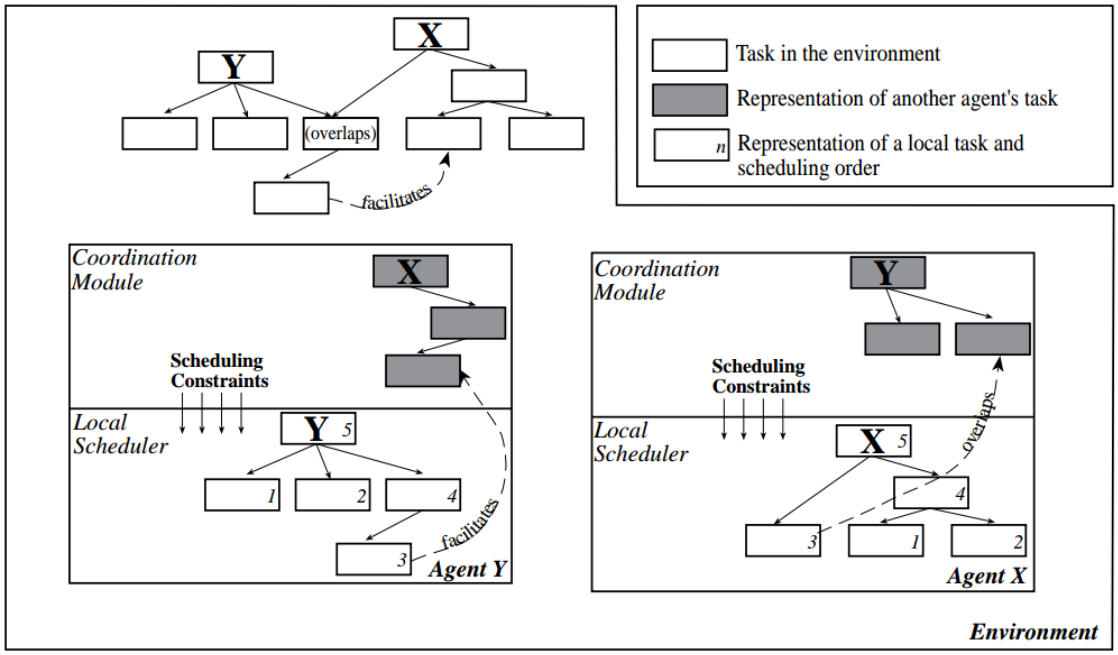

Figura 26 – Exemplo de estrutura GPGP. Fonte: Decker e Lesser (1992).

A estrutura do padrão GPGP pode ser exemplificada pela Figura 26. Basicamente, a informação flui pelo ambiente através do mecanismo de coordenação do agente,

Coordination Module, e do mecanismo de agendamento local, *Local Scheduler*. Primeiro, um determinado ambiente e/ou domínio de tarefa, em conjunto com um determinado agente, induz determinados relacionamentos de coordenação (*RCs*) entre tarefas nesse ambiente (DECKER; LESSER, 1992).

Cada agente segue algum algoritmo de coordenação que detecta ou até mesmo hipotetiza *RCs* e reage de acordo. Em seguida, este algoritmo produz certos comportamentos, por exemplo, (i) a criação e refinamento de restrições no agendamento local, e (ii) a negociação e criação de estruturas de dados internas (DECKER; LESSER, 1992).

Modelagem: Este padrão pode ser exemplificado pela Figura 26.

Exemplo: De modo a exemplificar uma utilização do padrão, é apresentado o caso abordado no artigo *Coordinated Hospital Patient Scheduling* (DECKER; LI, 1998) e exemplificado pela Figura 27.

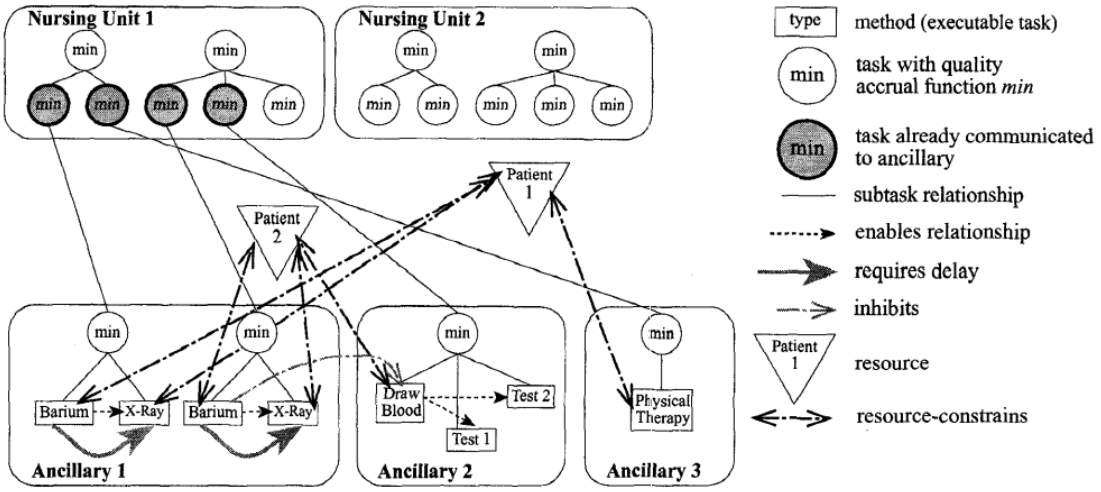

Figura 27 – Exemplo de agendamento de um paciente. Fonte: Decker e Li (1998).

Os pacientes de um hospital geral residem em unidades organizadas por ramos de medicina, como ortopedia ou neurocirurgia. Todos os dias, os médicos solicitam que determinados testes e/ou terapias sejam realizados como parte do diagnóstico e tratamento de um paciente. Os testes são realizados por departamentos auxiliares separados, independentes e dispostos em diferentes locais do hospital. O departamento de radiologia, por exemplo, presta serviços de raio-x e pode receber pedidos de várias unidades diferentes no hospital.

Além disso, cada teste pode interagir com outros testes em relacionamentos, tais como: autorizar, requisitar, atrasar, impedir outros testes. Essas relações entre tarefas indicam quando a execução de uma tarefa altera as características de outra tarefa e, neste caso, é principalmente a duração. A partir dessa visão da estrutura

da tarefa, o problema de agendamento do hospital tem as seguintes características peculiares:

- Tarefas não têm redundância. Cada teste só pode ser feito por um único departamento auxiliar;
- As funções de acúmulo de qualidade de tarefas não executáveis são sempre mínimas. Isto porque aqui todos os testes precisam ser feitos;
- A qualidade final não é importante: um teste é concluído ou não é, e
- Agentes diferentes, representando diferentes *Nursing Units* (unidades de enfermagem) ou diferentes *Ancillary Units* (unidades auxiliares), podem usar diferentes regras para fazer programações locais. Por exemplo, as *Nursing Units* podem tentar minimizar o tempo de permanência do paciente, enquanto a *Ancillary Unit* tenta maximizar o uso do equipamento e/ou minimizar os tempos depreendidos em configuração dos mesmos.

Neste problema de agendamento do hospital, alguns testes só podem ser feitos com o paciente fisicamente presente. Assim, surge um problema de restrição de recursos, ou ***shared resource-constrained relationship***, em que **o paciente se torna um recurso crucial não-compartilhável** para diferentes agentes.

Quando vários agentes tentam usar o mesmo recurso não compartilhável em horários sobrepostos, apenas um agente pode realmente obter o recurso e executar seu trabalho. Outros que não falharam em obter o recurso, desperdiçaram tempo e esforço da unidade.

A ideia por trás do mecanismo de coordenação com restrição de recursos é que quando um agente pretende executar uma tarefa com restrição de recurso, ele deve enviar um lance informando o intervalo de tempo que necessita do recurso e a sua prioridade local.

Após um pequeno *delay* de comunicação, ele passa a conhecer todos os lances que foram dados pelos outros agentes ao mesmo tempo que seu próprio lance. Como todos os agentes que fizeram lances terão as mesmas informações inicias, se todos usarem a mesma regra comumente aceita para decidir quem receberá o intervalo de tempo, eles poderão obter o mesmo resultado nessa rodada de lances.

O agente que ganhou manterá sua programação e executará essa tarefa no intervalo de tempo que ele solicitou no lance, e todos os outros marcarão esse intervalo de tempo com um compromisso de não fazer e ou sequer tentar executar a tarefa neste período que necessite do recurso. Todos os agentes que não obtiveram seus intervalos de tempo nessa rodada serão reprogramados e realizarão lances novamente. O processo detalhado é o seguinte:

Por exemplo, suponha que o atraso de comunicação seja uma unidade de tempo e que hajam três agentes e tarefas da seguinte maneira:

- O agente A tem as tarefas A11 e A12, cada uma de duração 3;
- O agente B tem a tarefa B11 de duração 2;
- O agente C tem a tarefa C11 de duração 4 e a tarefa C12 de duração 1;
- A11, B11 e C11 se caracterizam como ***shared resource-constrained relationship***.

O fluxo seria o seguinte:

Tempo 0: cada agente comunica estruturas de tarefas.

Tempo 1: cada agente faz sua própria programação local: Agente A: A11 (1-3), A12 (4-6); Agente B: Bll (1-2); Agente C: C12 (1-1). Momentaneamente a tarefa C11 não está no calendário local do Agente C. A razão podem ser causas externas, como C11 não estar habilitado.

O Agente A envia um lance para o intervalo de tempo 1-3 com prioridade 3.

O agente B envia um lance para o intervalo de tempo 1-2 com prioridade 4.

Tempo 2: cada agente reúne informações sobre o lance que aconteceu no tempo 1.

O Agente A descobre que outro agente ganhou o intervalo de tempo de 1 a 2, por isso marca esse intervalo de tempo como ocupado e, em seguida, tenta se reprogramar. Seus novos horários são: A12 (2-4), A11 (5-7).

O Agente A envia um lance de 5-7 com prioridade 3.

O Agente B ganhou, por isso mantém o seu horário e inicia a execução de B11.

O Agente C coloca C11 (3-6) na programação geral. O Agente C envia um lance para 3-6 com prioridade 2.

Tempo 3: o Agente A descobre que ganhou, por isso mantém sua programação e começa a executar A12.

O Agente C descobre que perdeu, por isso marca o intervalo de tempo 5-7 ocupado. A nova programação é C11 (7-10).

O Agente C envia um lance para o intervalo de tempo 7-10.

Tempo 4: o Agente C venceu, ele mantém sua programação e, portanto, aguardará até o tempo 7 para iniciar sua execução.

Implementação: a fim de demonstrar este padrão, o código apresentado no Apêndice B.5 simula o fluxo do contexto apresentado acima.

7 Conclusão

Esse capítulo pretende apresentar uma visão geral dos principais os resultados alcançados e possíveis trabalhos futuros que agregariam valor a esta área de pesquisa. Ele está dividido nas seguintes seções: Seção 7.1, responsável por retomar os objetivos, salientando as contribuições desse trabalho, bem como as ações para alcançar tais resultados, e a Seção 7.2, explorando uma série de possibilidades de continuação desse trabalho.

7.1 Considerações finais

O desenvolvimento de grandes SMAs é uma tarefa complexa que envolve os processos de requisitos, arquitetura, design e implementação desses sistemas. Em particular, o design arquitetural é crítico para lidar com o aumento de tamanho e de complexidade desses sistemas.

Como foi apresentado ao longo deste trabalho, estruturas organizacionais pra modelar SMA são amplamente utilizadas para apoiar atividades de análise e *design* destes sistemas. O presente trabalho apresentou uma abordagem para a elaboração de um catálogo de padrões arquiteturais baseado em estruturas organizacionais, bem como iniciou sua elaboração.

O catálogo é composto por elementos que facilitam a descrição de cada padrão, são eles: o nome do padrão, referências ao padrão, sua categoria, problema que se propõe a solucionar, a solução, os protocolos associados, sua modelagem e, por fim, a implementação.

Com o objetivo de fornecer o embasamento para proposta desse trabalho, foram apresentados alguns conceitos teóricos, referentes à agentes, SMA, arquitetura de software, padrão arquitetural, estruturas organizacionais, padrão de projeto e, por fim, catálogo de padrões.

A metodologia baseou-se, basicamente, em fluxos de atividades que foram realizadas para a elaboração de uma primeira versão do catálogo, utilizando-se técnicas de pesquisa científica e Engenharia de Software. Foi definido um protocolo de busca para a condução da Revisão Sistemática, onde foram identificados trabalhos que apresentam modelos arquiteturais candidatos a padrões arquiteturais.

Durante a primeira etapa da pesquisa, foi realizada a prova de conceito voltada à documentação do primeiro padrão arquitetural a compor o catálogo, o *Contract Net*. A catalogação deste padrão permitiu que se conduzisse atividades críticas, inerentes à pesquisa.

Percebeu-se que o desenvolvimento da prova de conceito foi de grande valia para a validação das ideias e da metodologia proposta. Analisar diferentes trabalhos sobre o tema também foi fundamental para que uma proposta coerente pudesse ser escrita.

Em seguida, partiu-se para a segunda etapa do trabalho, com práticas mais específicas da Engenharia de Software. O cronograma, apresentado na Seção 4.6, foi seguido. Os modelos arquiteturais, resultantes da Revisão Sistemática, foram analisados individualmente para que se identificasse aqueles que se caracterizavam como padrões arquiteturais. A ferramenta *Parsifal* foi de grande ajuda (Figura 28).

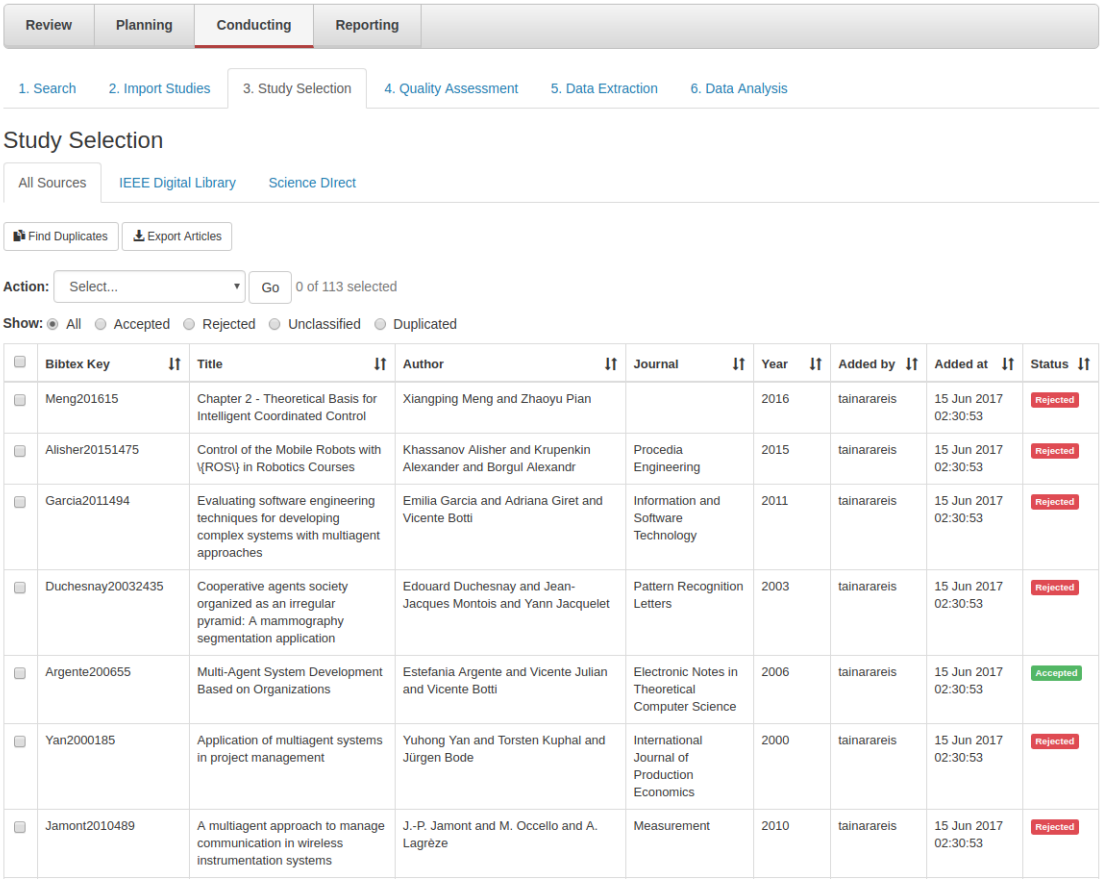

Figura 28 – Utilização da ferramenta *Parsifal*.

Os padrões identificados foram descritos e desenvolvidos em *sprints* com a duração de duas semanas. A cada *sprint* de desenvolvimento, um padrão arquitetural era catalogado, evidenciando uma evolução constante do catálogo bem como o desenvolvimento iterativo incremental, previsto no *Scrum* adaptado (Seção 4.5.2), e um refinamento em ciclos, previsto com pesquisa-ação (Seção 4.1). Utilizou-se a ferramenta *Trello* para apoiar a organização e planejamento das *sprints* (Figura 29). Cada padrão de projeto foi descrito no formato de *user stories* (Figura 30).

Os resultados e conclusões a cerca dos objetivos definidos para este trabalho são resumidos a seguir:

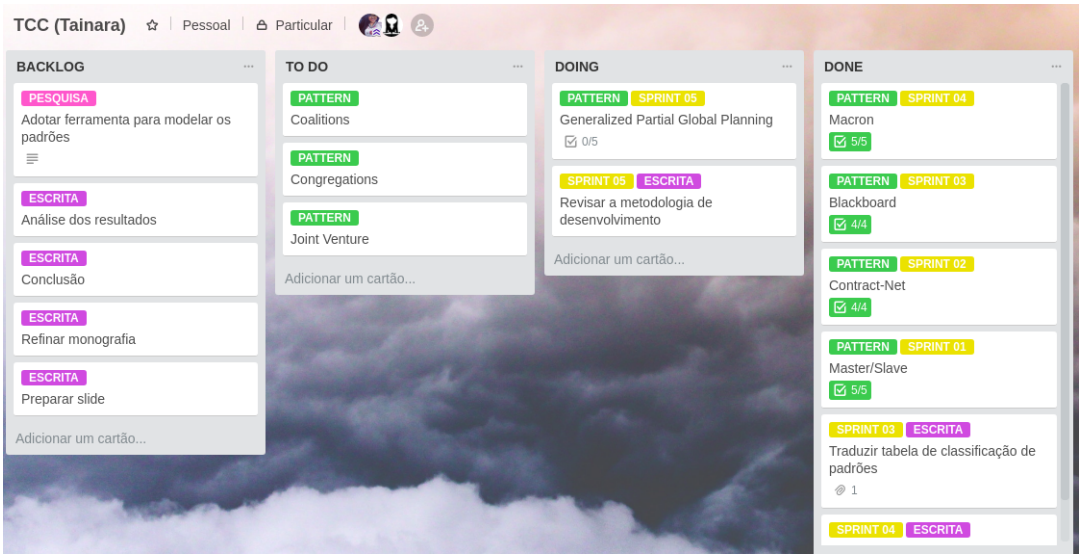

Figura 29 – Kanban. Fonte: Autora.

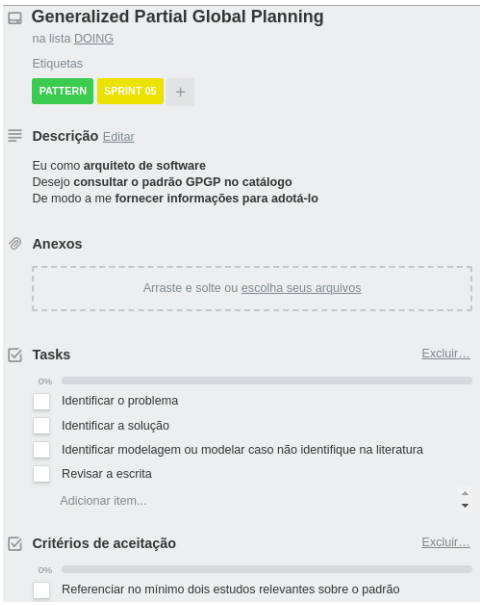

Figura 30 – *User storie - Generalized Partial Global Planning.* Fonte: Trello.

1. **Identificar modelos arquiteturais de SMA candidatos a padrões arquiteturais:** na Seção 6.1.2 são listados os artigos selecionados no segundo ciclo de busca e que possuem exemplares destes modelos.

2. **Selecionar modelos arquiteturais que se caracterizam como padrões arquiteturais:** foram caracterizados como padrões arquiteturais os modelos identificados, bem como nos estudos realizados e conhecimentos adquiridos pela autora ao longo da Revisão Sistemática.

3. **Catalogar os padrões arquiteturais:** cinco dos oito padrões identificados foram detalhados na Seção 5.1.

Dentre as contribuições dessa versão inicial do catálogo, tem-se:

- Possibilitar aos interessados maior conveniência na consulta, no conhecimento e no uso dos padrões arquiteturais associados ao contexto do Paradigma de Sistemas Multiagentes, e
- Propor uma forma mais padronizada para evolução dessa iniciativa por futuros colaboradores, uma vez que a proposta desse catálogo é ser um artefato vivo, evolutivo. A intenção é permitir, por exemplo, o acompanhamento de tendências tecnológicas (novas linguagens ou modelagens), ou mesmo o uso de notações mais genéricas possíveis visando a reutilização do conteúdo desse catálogo em diferentes domínios.

7.2 Trabalhos futuros

Este trabalho fornece a abertura para variados trabalhos futuros. Dentre eles, podem ser mencionados:

- Relacionar padrões de agentes com padrões de SMA: realizar estudos voltados para propostas de padrões arquiteturais que contemplem ambos os níveis organizacional e nível interno de um agente;
- Relacionar os padrões de SMA entre si: assim como os padrões de projetos detalhados por Gamma (1995) são relacionáveis entre si e podem se complementar, poderia ser investigado como os padrões identificados neste trabalho podem se relacionar entre si e serem integrados em pares ou em maior número;
- Estudar vantagens e desvantagens: nos elementos escolhidos para descrever os padrões arquiteturais identificados (Seção 5.1.3) foram diluídos alguns comentários da literatura sobre vantagens, desvantagens, limitações, indicações da adoção dos mesmos. Entretanto, não foi realizada uma pesquisa voltada para este conteúdo em específico, cabendo maiores investigações de modo a complementar o catálogo;
- Seria relevante abordar como tais padrões conferem melhor manutenibilidade ou não de código, e
- Implementar os padrões usando como base estudos de caso, aplicando-os em casos reais.

Referências

AART, C. van. *Organizational Principles for Multi-Agent Architectures*. [S.l.]: Springer Science & Business Media, 2004. Citado 2 vezes nas páginas 24 e 31.

APPLETON, B. *Patterns and software: Essential concepts and terminology*. 1997. Citado na página 34.

ARGENTE, E.; JULIAN, V.; BOTTI, V. Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science*, v. 150, n. 3, p. 55 – 71, 2006. ISSN 1571-0661. Proceedings of the First International Workshop on Coordination and Organisation (CoOrg 2005)Proceedings of the First International Workshop on Coordination and Organisation (CoOrg 2005). Disponível em: <http://www.sciencedirect.com/science/article/pii/S157106610600329X>. Citado 7 vezes nas páginas 13, 54, 55, 56, 57, 65 e 67.

ARIDOR, Y.; LANGE, D. B. Agent design patterns: elements of agent application design. In: ACM. *Proceedings of the second international conference on Autonomous agents*. [S.l.], 1998. p. 108–115. Citado 3 vezes nas páginas 13, 71 e 73.

AVGERIOU, P.; ZDUN, U. Architectural patterns revisited–a pattern. 2005. Citado 3 vezes nas páginas 24, 30 e 31.

BAJO, J. et al. A low-level resource allocation in an agent-based cloud computing platform. *Applied Soft Computing*, v. 48, p. 716 – 728, 2016. ISSN 1568-4946. Disponível em: <http://www.sciencedirect.com/science/article/pii/S156849461630268X>. Citado na página 65.

BEEDLE, M. et al. Scrum: An extension pattern language for hyperproductive software development. *Pattern Languages of Program Design*, v. 4, p. 637–651, 1999. Citado 2 vezes nas páginas 49 e 50.

BELLIFEMINE, F. et al. Jade programmer's guide. *Jade version*, v. 3, 2002. Disponível em: <http://jade.tilab.com/doc/programmersguide.pdf>. Citado 5 vezes nas páginas 13, 39, 68, 69 e 71.

BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. *Developing multi-agent systems with JADE*. [S.l.]: John Wiley & Sons, 2007. v. 7. Citado 7 vezes nas páginas 13, 15, 37, 68, 70, 71 e 72.

BIOLCHINI, J. et al. Systematic review in software engineering. *System Engineering and Computer Science Department COPPE/UFRJ, Technical Report ES*, v. 679, n. 05, p. 45, 2005. Citado 2 vezes nas páginas 13 e 48.

BONITASOFT. *About Bonita BPM*. [s.n.], 2016. Disponível em: <http://www.bonitasoft.com/products#about-bonita-bpm>. Citado na página 39.

BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. C. *Pattern-oriented software architecture, on patterns and pattern languages*. [S.l.]: John wiley & sons, 2007. v. 5. Citado na página 30.

CAIRE, G. Jade programming tutorial for beginners. *TILAB, CSELT*, 2009. Citado 4 vezes nas páginas 13, 37, 38 e 39.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Apress, 2014. Citado na página 40.

CHEN, X. Research on grid-service based virtual products experience environment supporting architecture in e-commerce applications. In: *2009 International Conference on E-Business and Information System Security*. [S.l.: s.n.], 2009. p. 1–5. Citado na página 64.

CHEN, Y.; XIAO, X.; SONG, Y. Research on the bidding information integrated system of generation group. In: *2008 International Seminar on Business and Information Management*. [S.l.: s.n.], 2008. v. 2, p. 225–228. Citado na página 64.

DAM, K. H. van; LUKSZO, Z. Modelling energy and transport infrastructures as a multi-agent system using a generic ontology. In: *2006 IEEE International Conference on Systems, Man and Cybernetics*. [S.l.: s.n.], 2006. v. 1, p. 890–895. Citado na página 64.

DEBIAN. *About Debian*. 2016. Disponível em: <https://www.debian.org/intro/about>. Citado na página 40.

DECKER, K. et al. Macron: an architecture for multi-agent cooperative information gathering. In: *Proccedings of the CIKM-95 Workshop on Intelligent Information Agents*. [S.l.: s.n.], 1995. Citado 4 vezes nas páginas 13, 80, 81 e 83.

DECKER, K.; LI, J. Coordinated hospital patient scheduling. In: IEEE. *Multi Agent Systems, 1998. Proceedings. International Conference on*. [S.l.], 1998. p. 104–111. Citado 2 vezes nas páginas 13 e 87.

DECKER, K. S.; LESSER, V. R. Generalizing the partial global planning algorithm. *International Journal of Intelligent and Cooperative Information Systems*, World Scientific, v. 1, n. 02, p. 319–346, 1992. Citado 4 vezes nas páginas 13, 85, 86 e 87.

DEUGO, D.; WEISS, M.; KENDALL, E. Reusable patterns for agent coordination. In: *Coordination of Internet agents*. [S.l.]: Springer, 2001. p. 347–368. Citado na página 60.

DONG, J.; CHEN, S.; JENG, J.-J. Event-based blackboard architecture for multi-agent systems. In: IEEE. *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*. [S.l.], 2005. v. 2, p. 379–384. Citado 2 vezes nas páginas 75 e 76.

ECLIPSE FOUNDATION. *Help - Eclipse Plataform: What is Eclipse?* 2016. Disponível em: <http://help.eclipse.org/neon/index.jsp>. Citado na página 40.

ENGEL, G. I. Pesquisa-ação. *Educar em Revista*, SciELO Brasil, n. 16, p. 181–191, 2000. Citado na página 44.

FILIPE, J. B. L.; FRED, A. L. N. Collective agents and collective intentionality using the eda model. In: *2007 Sixth Mexican International Conference on Artificial Intelligence, Special Session (MICAI)*. [S.l.: s.n.], 2007. p. 211–220. Citado na página 64.

FONSECA, J. J. S. d. Metodologia da pesquisa científica. *Fortaleza: UEC*, p. 65–75, 2002. Citado na página 45.

GABRIEL, A. et al. Multi-agent system to support creative workshop. In: *2015 11th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*. [S.l.: s.n.], 2015. p. 693–697. Citado na página 65.

GAMMA, E. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Education India, 1995. Citado 6 vezes nas páginas 13, 15, 32, 33, 34 e 94.

GIL, A. C. Como elaborar projetos de pesquisa. *São Paulo*, v. 5, p. 61, 2002. Citado 2 vezes nas páginas 43 e 44.

GIL, A. C. *Métodos e técnicas de pesquisa social*. [S.l.]: Atlas, 2008. v. 6. Citado na página 43.

GITHUB. *GitHub*. 2016. Disponível em: <https://github.com/>. Citado na página 40.

GREEN, S. et al. Software agents: A review. *Department of Computer Science, Trinity College Dublin, Tech. Rep. TCS-CS-1997-06*, 1997. Citado na página 23.

GRISS, M. L. Software agents as next generation software components. *Component-based software engineering*, Addison-Wesley, p. 641–657, 2001. Citado 3 vezes nas páginas 13, 27 e 28.

HOANG, T. T. H.; OCCELLO, M.; JAMONT, J. P. A generic recursive multiagent model to simplify large scale multi-level systems observation. In: *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*. [S.l.: s.n.], 2011. v. 2, p. 155–158. Citado na página 65.

ITO, T.; SALLEH, M. R. A blackboard-based negotiation for collaborative supply chain system. *Journal of Materials Processing Technology*, Elsevier, v. 107, n. 1, p. 398–403, 2000. Citado 5 vezes nas páginas 13, 75, 77, 78 e 79.

JADE. Java agent development framework. 2017. Disponível em: <http://jade.tilab.com/>. Citado na página 37.

JENNINGS, N. R. On agent-based software engineering. *Artificial intelligence*, Elsevier, v. 117, n. 2, p. 277–296, 2000. Citado 2 vezes nas páginas 13 e 29.

JENNINGS, N. R.; WOOLDRIDGE, M. J. Software agents. *IEE review*, p. 17–20, 1996. Citado 2 vezes nas páginas 23 e 27.

JOHNSON, R. et al. Design patterns: Elements of reusable object-oriented software. *Boston, Massachusetts: Addison-Wesley*, 1995. Citado na página 32.

JUNIOR, G. B. d. S. et al. Padrões arquiteturais para o desenvolvimento de aplicações multiagente. Universidade Federal do Maranhão, 2003. Citado 5 vezes nas páginas 13, 23, 25, 60 e 61.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Citado na página 47.

KOLP, M.; GIORGINI, P.; MYLOPOULOS, J. Organizational multi-agent architectures: a mobile robot example. In: ACM. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*. [S.l.], 2002. p. 94–95. Citado na página 104.

KOLP, M.; GIORGINI, P.; MYLOPOULOS, J. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 13, n. 1, p. 3–25, 2006. Citado 3 vezes nas páginas 25, 31 e 103.

KOSKIMIES, K. *Pattern Mining and Pattern Life-Cycle*. 2002. Disponível em: <http://www.cs.tut.fi/~ohar/Slides2000/Luento5/sld032.htm>. Citado 2 vezes nas páginas 13 e 60.

LARMAN, C. *Utilizando UML e padrões*. [S.l.]: Bookman Editora, 2002. Citado na página 59.

LATEX. *A document preparation system*. 2016. Disponível em: <https://latex-project.org/>. Citado na página 40.

LESSER, V. R. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on knowledge and data engineering*, IEEE, v. 11, n. 1, p. 133–142, 1999. Citado na página 28.

LIU, Q.; CUI, X.; HU, X. Conflict resolution within multi-agent system in collaborative design. In: *2008 International Conference on Computer Science and Software Engineering*. [S.l.: s.n.], 2008. v. 1, p. 520–523. Citado na página 64.

MALONE, T. W. Organizing information processing systems: Parallels between human organizations and computer systems. *Cognition, Computation and Cooperation*, Ablex, Norwood, NJ, p. 56–83, 1990. Citado 2 vezes nas páginas 54 e 57.

MATSON, E. T. Embedding intelligent agents to enable physical robotic and sensor organizations. In: *2009 IEEE International Symposium on Computational Intelligence in Robotics and Automation - (CIRA)*. [S.l.: s.n.], 2009. p. 309–315. Citado na página 64.

MCARTHUR, S. D. et al. Multi-agent systems for power engineering applications—part i: Concepts, approaches, and technical challenges. *IEEE Transactions on Power systems*, IEEE, v. 22, n. 4, p. 1743–1752, 2007. Citado 2 vezes nas páginas 27 e 28.

MERCIER, A.; OCCELLO, M.; JAMONT, J. P. Using multiagent self-organization techniques to improve dynamic skill searching in virtual social communities. In: *2010 22nd IEEE International Conference on Tools with Artificial Intelligence*. [S.l.: s.n.], 2010. v. 1, p. 481–484. Citado na página 65.

NICKLES, M.; ROVATSOS, M.; WEISS, G. Expectation-oriented modeling. *Engineering Applications of Artificial Intelligence*, v. 18, n. 8, p. 891 – 918, 2005. ISSN 0952-1976. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0952197605000618>. Citado na página 65.

PAPEERIA. *Papeeria: online LaTeX editor*. 2016. Disponível em: <https://www.papeeria.com/>. Citado na página 41.

PARSIFAL. *Perform Systematic Literature Reviews*. 2016. Disponível em: <https://parsif.al/>. Citado na página 41.

PENG, X. h. et al. Study on cooperation mechanism in agent organization. In: *2006 International Conference on Machine Learning and Cybernetics*. [S.l.: s.n.], 2006. p. 906–910. Citado na página 64.

PRODANOV, C. C.; FREITAS, E. C. de. *Metodologia do Trabalho Científico: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico-2ª Edição*. [S.l.]: Editora Feevale, 2013. Citado 2 vezes nas páginas 43 e 44.

RICHARDS, M. Software architecture patterns. *O'Reilly Media*, 2015. Citado 2 vezes nas páginas 23 e 24.

RUBIN, K. S. *Essential Scrum: a practical guide to the most popular agile process*. [S.l.]: Addison-Wesley, 2012. Citado 2 vezes nas páginas 49 e 50.

SCHWABER, K.; SUTHERLAND, J. *The Scrum Guide – The definitive guide to Scrum: The rules of the game (2016)*. 2016. Disponível em: <http://www.scrumguides.org/docs/scrumguide/v2016/2016-Scrum-Guide-US.pdf>. Citado na página 49.

SELKER, T. Coach: a teaching agent that learns. *Communications of the ACM*, ACM, v. 37, n. 7, p. 92–99, 1994. Citado na página 27.

SERRANO, M. *Desenvolvimento Intencional de Software Transparente Baseado em Argumentação*. Tese (Doutorado) — PUC-Rio, 2011. Citado 2 vezes nas páginas 13 e 29.

SHAW, M. Some patterns for software architectures. *Pattern languages of program design*, v. 2, p. 255–269, 1996. Citado 3 vezes nas páginas 24, 30 e 31.

SHEHORY, O. M. *Architectural properties of multi-agent systems*. [S.l.]: Carnegie Mellon University, The Robotics Institute, 1998. Citado 4 vezes nas páginas 13, 80, 81 e 82.

SILVA, R. M. d. Qualidade na modelagem de processos de software. 2014. Citado na página 48.

SYCARA, K. P. Multiagent systems. *AI magazine*, v. 19, n. 2, p. 79, 1998. Citado 3 vezes nas páginas 24, 54 e 58.

TRELLO. *Sobre o Trello*. [s.n.], 2017. Disponível em: <https://trello.com/tour>. Citado na página 39.

TUNG DO, T.; FAULKNER, S.; KOLP, M. Organizational multi-agent architectures for information systems. In: *ICEIS (3)*. [S.l.: s.n.], 2003. p. 89–96. Citado na página 104.

UNLAND, R. et al. Aegis: Agent oriented organisations. *Accounting, Management and Information Technologies*, v. 5, n. 2, p. 139 – 162, 1995. ISSN 0959-8022. Disponível em: <http://www.sciencedirect.com/science/article/pii/095980229500009X>. Citado na página 65.

VLIET, H. V.; VLIET, J. V. *Software engineering: principles and practice*. [S.l.]: Wiley New York, 2007. v. 3. Citado 2 vezes nas páginas 29 e 30.

WEERDT, M. D.; CLEMENT, B. Introduction to planning in multiagent systems. *Multiagent and Grid Systems*, IOS Press, v. 5, n. 4, p. 345–355, 2009. Citado na página 85.

WEISS, G. *Multiagent systems: a modern approach to distributed artificial intelligence*. [S.l.]: MIT press, 1999. Citado 3 vezes nas páginas 13, 75 e 77.

WEISS, G. Computational organization theory. In: _____. *Multiagent Systems:A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 2000. p. 299–330. ISBN 9780262257183. Disponível em: <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6284635>. Citado na página 64.

WOOLDRIDGE, M. *An introduction to multiagent systems*. [S.l.]: John Wiley & Sons, 2009. Citado 4 vezes nas páginas 23, 25, 28 e 29.

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: Theory and practice. *The knowledge engineering review*, Cambridge Univ Press, v. 10, n. 02, p. 115–152, 1995. Citado na página 28.

WU, K.; DUAN, C.; SHI, Q. Multi-agent based power grid data integration and sharing platform. In: *2009 International Conference on Artificial Intelligence and Computational Intelligence*. [S.l.: s.n.], 2009. v. 4, p. 184–188. Citado na página 64.

XIANG, Y.; HANSHAR, F. Comparison of tightly and loosely coupled decision paradigms in multiagent expedition. *International Journal of Approximate Reasoning*, v. 51, n. 5, p. 600 – 613, 2010. ISSN 0888-613X. PGM-2008. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0888613X1000023X>. Citado na página 65.

ZAMBONELLI, F.; JENNINGS, N. R.; WOOLDRIDGE, M. Organisational abstractions for the analysis and design of multi-agent systems. In: SPRINGER. *Agent-oriented software engineering*. [S.l.], 2001. p. 235–251. Citado 3 vezes nas páginas 23, 31 e 32.

Apêndices

APÊNDICE A – Protocolo de Revisão Sistemática

A.1 Tema

O objetivo dessa revisão sistemática é identificar modelos arquiteturais organizacionais para sistemas multiagentes. Tais modelos são vistos como candidatos a padrões arquiteturais para SMA.

A.2 Problema

Não foram encontrados em pesquisas preliminares pela autora estudos que reúnam modelos arquiteturais para SMA e que tratem o nível organizacional. E, consequentemente não são muito bem conhecidas as características destes modelos.

A.2.1 Questões de pesquisa

Serão investigadas as respostas para as seguintes questões:

1. Quais são os modelos arquiteturais de nível organizacional para SMA?
2. Quais são as características dos modelos arquiteturais identificados?

A.2.2 Intervenção

Serão observados modelos de arquitetura de SMA que se enquadram como estruturas organizacionais.

A.2.3 População

Áreas de aplicação e pesquisa de SMA serão observadas durante a intervenção.

A.2.4 Controle

O conjunto de dados iniciais utilizados são artigos que abordam diretamente o objeto de estudo, são eles:

- *Multi-agent architectures as organizational structures* (KOLP; GIORGINI; MYLOPOULOS, 2006);

- *Organizational Multi-Agent Architectures for Information Systems* (TUNG DO; FAULKNER; KOLP, 2003);
- *Organizational multi-agent architectures: a mobile robot example* (KOLP; GIORGINI; MYLOPOULOS, 2002).

A.2.5 Resultados

Espera-se a obtenção de modelos arquiteturais organizacionais para SMA que sejam potencialmente classificáveis como padrões arquiteturais.

A.2.6 Aplicação

Serão beneficiadas áreas de aplicação de SMA; especificamente em processos de desenvolvimento de software utilizando como arquitetura estruturas organizacionais.

A.2.7 Palavras-chave e sinônimos

Foram consideradas como sinônimos as seguintes palavras-chave: *"Multi-agent* System"* e *"Multiagent* System"*; *"Organisation* Structur*"* e *"Organization* Structur*"*. As demais palavras chave são: *"Architectur* Pattern"*.

A.3 Seleção de fontes de pesquisa

A.3.1 Critérios para seleção de fonte

Os seguintes critérios foram adotados para a seleção das fontes de pesquisa:

1. Dentre as áreas de domínio das fontes de pesquisa deverão se enquadrar à área da Computação ou Engenharia de Software;
2. Os artigos deverão estar disponíveis gratuitamente ou as fontes deverão ser conveniadas com a UnB;
3. Deve disponibilizar os artigos eletronicamente;
4. Deve conter artigos em inglês e português.

A.3.2 Lista de fontes de pesquisa

As fontes de pesquisa adotadas são:

1. *IEEE Computer Science Digital Library*[1];

[1] http://ieeexplore.ieee.org

2. *Science Direct*[2].

A.4 Seleção de trabalhos

A.4.1 *Strings* de pesquisa

A *string* de pesquisa foi a seguinte:

"Multiagent System"AND ("Organization Structure"OR "Organisation* Structure")*

A.4.2 Critérios de inclusão

Serão adotados os seguintes critérios para incluir artigos aos resultados:

1. O artigo deve ser escrito em inglês ou português;
2. O artigo deve estar disponível em meio eletrônico;
3. O artigo deve apresentar no mínimo um modelo arquitetural organizacional para SMA descrevendo suas características de forma explícita; e/ou apresentar relato de experiências com tais modelos; e/ou comparar modelos.

A.4.3 Critérios de exclusão

Serão adotados os seguintes critérios para excluir artigos dos resultados:

1. O artigo não aborda SMA;
2. O artigo não aborda arquiteturas de SMA;
3. O artigo não aborda o nível organizacional da arquitetura de SMA;
4. O artigo não se propõe a descrever um padrão arquitetural organizacional para SMA;
5. O artigo não pertence ao domínio da computação ou engenharia de software;
6. O artigo não está disponível para acesso gratuito por meio do convênio com a UnB.

[2] http://www.sciencedirect.com/

A.4.4 Procedimento para seleção dos trabalhos

Os critérios de inclusão e exclusão serão aplicados em três etapas de leitura dos artigos. Na primeira e segunda etapas, o artigo pode ser selecionado ou descartado. Na terceira etapa, o artigo pode ser de descartado ou, por fim, aceito. A seguir, são detalhados os conteúdos a serem lidos em cada etapa:

1ª etapa: título, resumo e palavras-chave;

2ª etapa: introdução e conclusão;

3ª etapa: todo o conteúdo.

A.5 Coleta de dados

Os artigos retornados deverão ser catalogados na ferramenta *Parsifal*, descrita na seção 3.2.9, a fim de facilitar a seleção dos mesmos, a análise do conteúdo e extração de informações.

APÊNDICE B – Implementação dos padrões arquiteturais organizacionais para SMA

B.1 *Contract Net*

B.1.1 Contexto

O exemplo a seguir baseia-se na venda e compra de livros entre agentes vendedores (instâncias da classe *BookSellerAgent*) e agentes compradores (instâncias da classe *BookBuyerAgent*).

Sob a perspectiva do agente comprador, este recebe o título do livro a ser comprado como um argumento em linha de comando e solicita periodicamente a todos os agentes vendedores que ofereçam uma oferta. Assim que uma oferta é recebida, o agente comprador aceita e emite uma ordem de compra.

Se mais de um agente vendedor oferecer uma oferta, o comprador compara os preços e realiza a compra do livro mais barato. Tendo comprado o livro, o agente do comprador termina.

Sob a perspectiva do agente vendedor, este possui uma interface, através da classe *BookSellerGui*, na qual o usuário pode inserir novos títulos e seus respectivos preços. Ao receber solicitações para fornecer uma proposta para um livro, eles verificam se possui o título solicitado, podendo responder com o preço caso o possua, ou recusando-se a fazer uma proposta caso não o possua. Em caso de enviar uma proposta com o preço e de receber uma ordem de compra, o livro é vendido e removido do catálogo.

B.1.2 Preparação do ambiente

O exemplo foi rodado em sistema operacional Ubuntu 16.04, utilizando a IDE Eclipse Neon versão 4.6.3. Os seguintes passos devem ser seguidos para rodar este exemplo:

1. Realizar o *download* do arquivo *jadeAll.zip* em *http://jade.tilab.com/download/jade/*[1] e descompactá-lo, bem como descompactar as pastas que encontram-se dentro do mesmo;

2. Editar o arquivo *.bashrc*. Para isso, executar o comando *nano .bashrc* e adicionar as linhas abaixo ao final do arquivo. Substituir */caminho/para-o/jade* com a localização

[1] Último acesso: Junho 2017

da pasta jade no computador. Por fim, salvar a edição.

```
#jade
export JADE_LIB=/caminho/para-o/jade
export JADE_CP=$JADE_LIB/http.
jar:$JADE_LIB/iiop.
jar:$JADE_LIB/jade.
jar:$JADE_LIB/jadeTools.
jar:
$JADE_LIB/commons-codec/commons-codec-1.3.
jar
alias rJade='
java -cp $JADE_CP jade.Boot'
alias cJade='
javac -cp $JADE_CP'
```

3. Importar o código *bookTrading*, presente na pasta *jade/jade-examples/src/examples/bookTrading*, para o Eclipse.
4. Selecionar *Java Build Path > Libraries > Add External JARs.*
5. Adicionar as seguintes bibliotecas do JADE ao projeto:
   - *jade/jade-src/lib/commons-codec/commons-codec- 1.3.jar*;
   - *jade/jade-bin/lib/jade.jar*;
   - jade/jade-examples/lib/jadeExamples.jar.
6. Selecionar *Run > Run Configurations > Java Application.*
7. Certificar que os seguintes parâmetros estão correspondentes aos valores a seguir:
   - *Project: BookAgents*
   - *Main class: jade.Boot*
8. Na aba *Arguments* inserir, em *Program Arguments*, o seguinte texto: *-nomtp -gui bookSeller1:sell.BookSellerAgent; bookSeller2:sell.BookSellerAgent; bookBuyer1:sell.BookBuyerAgent("colecionador");*
9. Selecionar *Aplly > Run.*

B.1.3 Classe *BookSellerAgent*

Para criar um agente JADE, define-se, basicamente, uma subclasse da classe *jade.core.Agent*[2] que implementa-se o método *setup()* e os comportamentos do agente. Por meio da classe jade.core.Agent, os agentes podem realizar operações básicas da plataforma como registro, configuração, e troca de mensagens. Através do método *setup()*, um agente registra seus serviços nas páginas amarelas e adiciona seus comportamentos.

[2] http://jade.tilab.com/doc/api/jade/core/Agent.html (último acesso: Junho 2017)

Inicialmente, cria-se um catálogo de livros e a interface gráfica para o registro dos mesmos através da classe *BookSellerGui*.

```
37 public class BookSellerAgent extends Agent {
38     // The catalogue of books for sale (maps the title of a book to its price)
39     private Hashtable catalogue;
40     // The GUI by means of which the user can add books in the catalogue
41     private BookSellerGui myGui;
42
43     // Put agent initializations here
44     protected void setup() {
45         // Create the catalogue
46         catalogue = new Hashtable();
47
48         // Create and show the GUI
49         myGui = new BookSellerGui(this);
50         myGui.showGui();
```

Em seguida, registra-se o serviço de venda de livros (um serviço do tipo *Book-selling*) nas páginas amarelas para que esteja disponível para outros agentes. Para publicar seus serviços, um agente deve inscrever-se no agente DF (*Directory Facilitator agent*). O DF reune e associa descrições de serviço aos seus identificadores (AID, *Agent Identifier*); de modo que os agentes visitantes possam contactá-lo à procura de agentes que prestam os serviços de que necessitam.

O padrão FIPA estabelece que cada instância de agente é identificada pelo AID (*Agent Identifier*). Na plataforma JADE, um AID é uma instância da classe *jade.core.AID*. Esta provê o método *getAID()* que retorna o nome global do agente na plataforma no seguinte formato: *<nome_local>@<nome-plataforma>*.

O agente deve, portanto, fornecer seu AID e uma lista de seus serviços fornecidos através de uma descrição adequada, como uma instância da classe DFAgentDescription[3] e, ao final, invocar o método estático *register()* da classe DFService[4].

```
52         // Register the book-selling service in the yellow pages
53         DFAgentDescription dfd = new DFAgentDescription();
54         dfd.setName(getAID());
55         ServiceDescription sd = new ServiceDescription();
56         sd.setType("book-selling");
57         sd.setName("JADE-book-trading");
58         dfd.addServices(sd);
59         try {
60             DFService.register(this, dfd);
61         }
62         catch (FIPAException fe) {
63             fe.printStackTrace();
64         }
```

[3] http://jade.tilab.com/doc/api/jade/domain/FIPAAgentManagement/DFAgentDescription.html (último acesso: Junho 2017)

[4] http://jade.tilab.com/doc/api/jade/domain/DFService.html (último acesso: Junho 2017)

Por fim, ainda no método *setup()*, são adicionados os comportamentos que atendem às consultas dos agentes compradores (através da classe *OfferRequestsServer* e os comportamentos que atendem às ordens de compra dos agentes compradores (através da classe *PurchaseOrdersServer*).

```
66          // Add the behaviour serving queries from buyer agents
67          addBehaviour(new OfferRequestsServer());
68
69          // Add the behaviour serving purchase orders from buyer agents
70          addBehaviour(new PurchaseOrdersServer());
71      }
```

Além do método *setup()*, a classe possui o método *takeDown()* que é invocado imediatamente antes de um agente ter encerrado seus serviços com objetivo de executar várias operações de limpeza.

```
73      // Put agent clean-up operations here
74      protected void takeDown() {
75          // Deregister from the yellow pages
76          try {
77              DFService.deregister(this);
78          }
79          catch (FIPAException fe) {
80              fe.printStackTrace();
81          }
82          // Close the GUI
83          myGui.dispose();
84          // Printout a dismissal message
85          System.out.println("Seller-agent "+getAID().getName()+" terminating.");
86      }
```

A classe *OfferRequestsServer* implementa um dos comportamentos da classe *BookSellerAgent*. Ao extender a classe *jade.core.behaviours.CyclicBehaviour*[5], o método *action()* repete-se à medida que o *setup()* invoque-o. Através deste comportamento, o agente vendedor pode atender aos pedidos de oferta recebidos de agentes compradores.

Uma instância da classe *jade.lang.acl.MessageTemplate*[6] possibilita que as mensagens recebidas pelo agente vendedor sejam filtradas. Esta classe especifica templates para serem usados ao chamar o método *receive()*. Quando um modelo é especificado, o método *receive()* retorna a primeira correspondência da mensagem, se houver, e ignora todas as mensagens não correspondentes.

Cabe ressaltar que toda mensagem corresponde a instâncias da classe ACLMessage[7]. Esta classe implementa o padrão FIPA-ACL e, assim, disponibiliza um conjunto de atributos que estão de acordo com as especificações FIPA.

[5] http://jade.tilab.com/doc/api/jade/core/behaviours/CyclicBehaviour.html (último acesso: Junho 2017)
[6] http://jade.tilab.com/doc/api/jade/lang/acl/MessageTemplate.html (último acesso: Junho 2017)
[7] http://jade.cselt.it/doc/api/jade/lang/acl/ACLMessage.html (último acesso: Junho 2017)

Neste caso, espera-se receber a performativa CFP (*Call For Proposal*) onde o agentes compradores solicitam propostas aos agentes vendedores especificando o nome do livro que desejam comprar. Se o livro solicitado estiver disponível no catálogo local, o agente vendedor responde com uma mensagem *PROPOSE* especificando o preço. Caso contrário, uma mensagem *REFUSE* é enviada.

```
108     private class OfferRequestsServer extends CyclicBehaviour {
109         public void action() {
110             MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
111             ACLMessage msg = myAgent.receive(mt);
112             if (msg != null) {
113                 // CFP Message received. Process it
114                 String title = msg.getContent();
115                 ACLMessage reply = msg.createReply();
116 
117                 Integer price = (Integer) catalogue.get(title);
118                 if (price != null) {
119                     // The requested book is available for sale. Reply with the price
120                     reply.setPerformative(ACLMessage.PROPOSE);
121                     reply.setContent(String.valueOf(price.intValue()));
122                 }
123                 else {
124                     // The requested book is NOT available for sale.
125                     reply.setPerformative(ACLMessage.REFUSE);
126                     reply.setContent("not-available");
127                 }
128                 myAgent.send(reply);
129             }
130             else {
131                 block();
132             }
133         }
134     } // End of inner class OfferRequestsServer
```

De modo semelhante, a classe *PurchaseOrdersServer* também especifica um comportamento do tipo *CyclicBehaviour* e aguarda o recebimento de uma performativa específica: *ACCEPT_PROPOSAL*. Esta é recebida quando o agente comprador aceita a proposta e a venda do livro pode, portanto, ser processada. O agente vendedor remove o livro comprado de seu catálogo e responde com uma mensagem *INFORM* para notificar o comprador de que a compra foi concluída.

```
144     private class PurchaseOrdersServer extends CyclicBehaviour {
145         public void action() {
146             MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
147             ACLMessage msg = myAgent.receive(mt);
148             if (msg != null) {
149                 // ACCEPT_PROPOSAL Message received. Process it
150                 String title = msg.getContent();
151                 ACLMessage reply = msg.createReply();
152 
153                 Integer price = (Integer) catalogue.remove(title);
154                 if (price != null) {
155                     reply.setPerformative(ACLMessage.INFORM);
156                     System.out.println(title+" sold to agent "+msg.getSender().getName());
```

```
157                     }
158                     else {
159                         // The requested book has been sold to another buyer in the meanwhile .
160                         reply.setPerformative(ACLMessage.FAILURE);
161                         reply.setContent("not-available");
162                     }
163                     myAgent.send(reply);
164                 }
165                 else {
166                     block();
167                 }
168             }
169         } // End of inner class OfferRequestsServer
170 }
```

B.1.4 Classe *BookBuyerAgent*

A classe *BookBuyerAgent* implementa os comportamentos relacionados à compra do livro de interesse do agente comprador. No método *setup()*, adiciona-se um comportamento do tipo *TickerBehaviour*[8]. Esta classe possibilita que execute-se um trecho de código por um período de tempo pré-definido por meio do método *onTick()*. Para este caso, foi definida a duração de um minuto.

Uma instância da classe *ServiceDescription*[9] é criada para definir um serviço do tipo *book-selling*, para que, através de uma instância da classe *DFAgentDescription()*, sejam buscados agentes vendedores que possuam em seu catálogo o *targetBookTitle*. Por fim, os AIDs dos agentes que prestam o serviço solicitado são salvos em uma lista chamada *sellerAgents*.

```
36 public class BookBuyerAgent extends Agent {
37     // The title of the book to buy
38     private String targetBookTitle;
39     // The list of known seller agents
40     private AID[] sellerAgents;
41
42     // Put agent initializations here
43     protected void setup() {
44         // Printout a welcome message
45         System.out.println("Hallo! Buyer-agent "+getAID().getName()+" is ready.");
46
47         // Get the title of the book to buy as a start-up argument
48         Object[] args = getArguments();
49         if (args != null && args.length > 0) {
50             targetBookTitle = (String) args[0];
51             System.out.println("Target book is "+targetBookTitle);
52
53             // Add a TickerBehaviour that schedules a request to seller agents every minute
54             addBehaviour(new TickerBehaviour(this, 60000) {
```

[8] http://jade.tilab.com/doc/api/jade/core/behaviours/TickerBehaviour.html (último acesso: Junho 2017)

[9] http://jade.tilab.com/doc/api/jade/domain/FIPAAgentManagement/ServiceDescription.html (último acesso: Junho 2017)

```
55                  protected void onTick() {
56                      System.out.println("Trying to buy "+targetBookTitle);
57                      // Update the list of seller agents
58                      DFAgentDescription template = new DFAgentDescription();
59                      ServiceDescription sd = new ServiceDescription();
60                      sd.setType("book-selling");
61                      template.addServices(sd);
62                      try {
63                          DFAgentDescription[] result = DFService.search(myAgent, template);
64                          System.out.println("Found the following seller agents:");
65                          sellerAgents = new AID[result.length];
66                          for (int i = 0; i < result.length; ++i) {
67                              sellerAgents[i] = result[i].getName();
68                              System.out.println(sellerAgents[i].getName());
69                          }
70                      }
71                      catch (FIPAException fe) {
72                          fe.printStackTrace();
73                      }
```

Em seguida, o comportamento *RequestPerformer* é adicionado ao agente comprador.

```
75                      // Perform the request
76                      myAgent.addBehaviour(new RequestPerformer());
```

Caso nenhum livro tenha sido especificado, o agente é finalizado.

```
80          else {
81              // Make the agent terminate
82              System.out.println("No target book title specified");
83              doDelete();
84          }
85      }
```

Logo após, é definida a classe *RequestPerformer*. Primeiramente, é enviada uma mensagem CFP para todos os agentes vendedores da lista *sellerAgents*.

```
98      private class RequestPerformer extends Behaviour {
99          private AID bestSeller; // The agent who provides the best offer
100         private int bestPrice;  // The best offered price
101         private int repliesCnt = 0; // The counter of replies from seller agents
102         private MessageTemplate mt; // The template to receive replies
103         private int step = 0;
104
105         public void action() {
106             switch (step) {
107             case 0:
108                 // Send the cfp to all sellers
109                 ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
110                 for (int i = 0; i < sellerAgents.length; ++i) {
111                     cfp.addReceiver(sellerAgents[i]);
112                 }
113                 cfp.setContent(targetBookTitle);
114                 cfp.setConversationId("book-trade");
115                 cfp.setReplyWith("cfp"+System.currentTimeMillis()); // Unique value
```

```
116                 myAgent.send(cfp);
117                 // Prepare the template to get proposals
118                 mt = MessageTemplate.and(MessageTemplate.MatchConversationId("book-trade"),
119                         MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
120                 step = 1;
121                 break;
```

Em seguida, todas as mensagens recebidas são lidas. Aquelas cujas performativas correspondem a *PROPOSE*, têm seu conteúdo lido, o que corresponde ao preço do livro. Logo após, os preços são comparados até que se encontre o mais barato (*bestPrice*).

```
122             case 1:
123                 // Receive all proposals/refusals from seller agents
124                 ACLMessage reply = myAgent.receive(mt);
125                 if (reply != null) {
126                     // Reply received
127                     if (reply.getPerformative() == ACLMessage.PROPOSE) {
128                         // This is an offer
129                         int price = Integer.parseInt(reply.getContent());
130                         if (bestSeller == null || price < bestPrice) {
131                             // This is the best offer at present
132                             bestPrice = price;
133                             bestSeller = reply.getSender();
134                         }
135                     }
136                     repliesCnt++;
137                     if (repliesCnt >= sellerAgents.length) {
138                         // We received all replies
139                         step = 2;
140                     }
141                 }
142                 else {
143                     block();
144                 }
145                 break;
```

Em seguida, é enviado o pedido ao vendedor que forneceu a melhor oferta (*bestSeller*) através da performativa *ACCEPT_PROPOSAL*.

```
146             case 2:
147                 // Send the purchase order to the seller that provided the best offer
148                 ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
149                 order.addReceiver(bestSeller);
150                 order.setContent(targetBookTitle);
151                 order.setConversationId("book-trade");
152                 order.setReplyWith("order"+System.currentTimeMillis());
153                 myAgent.send(order);
154                 // Prepare the template to get the purchase order reply
155                 mt = MessageTemplate.and(MessageTemplate.MatchConversationId("book-trade"),
156                         MessageTemplate.MatchInReplyTo(order.getReplyWith()));
157                 step = 3;
158                 break;
```

A performativa *INFORM* significa, neste caso, que a compra foi executada com êxito. Caso receba qualquer outra mensagem, significa que a tentativa de compra falhou.

```
159             case 3:
160                 // Receive the purchase order reply
161                 reply = myAgent.receive(mt);
162                 if (reply != null) {
163                     // Purchase order reply received
164                     if (reply.getPerformative() == ACLMessage.INFORM) {
165                         // Purchase successful. We can terminate
166                         System.out.println(targetBookTitle+" successfully purchased from agent
                                "+reply.getSender().getName());
167                         System.out.println("Price = "+bestPrice);
168                         myAgent.doDelete();
169                     }
170                     else {
171                         System.out.println("Attempt failed: requested book already sold.");
172                     }
173
174                     step = 4;
175                 }
176                 else {
177                     block();
178                 }
179                 break;
180             }
181         }
```

B.1.5 Resultados da execução

A Figura 31 apresenta a inicialização do contêiner e dos serviços da plataforma JADE. A Figura 32 apresenta a plataforma JADE e a estrutura do contêiner.

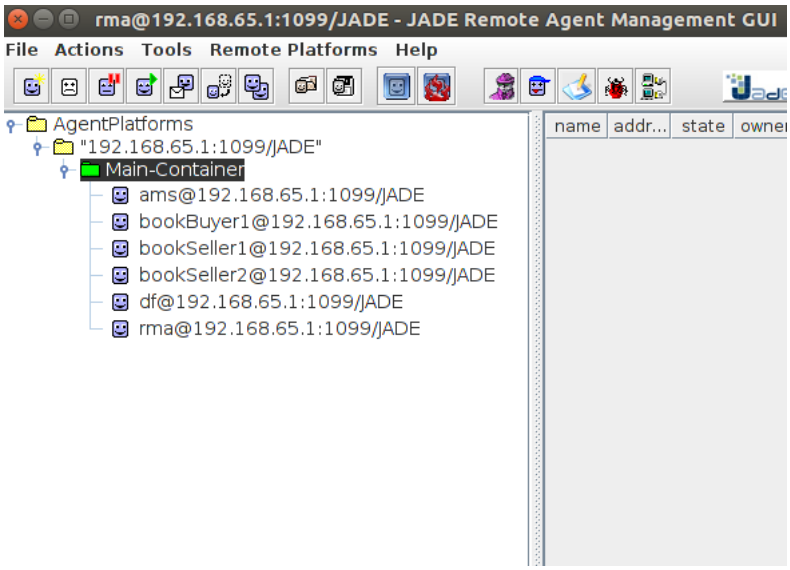

Figura 32 – Plataforma JADE

A Figura 33 apresenta o comportamento cíclico do agente comprador, que realiza várias tentativas de compra do livro a cada minuto.

```
jun 18, 2017 10:56:58 PM jade.core.Runtime beginContainer
INFORMAÇÕES: ----------------------------------
    This is JADE 4.4.0 - revision 6778 of 21-12-2015 12:24:43
    downloaded in Open Source, under LGPL restrictions,
    at http://jade.tilab.com/
----------------------------------------
jun 18, 2017 10:56:59 PM jade.imtp.leap.LEAPIMTPManager initialize
INFORMAÇÕES: Listening for intra-platform commands on address:
- jicp://192.168.65.1:1099

jun 18, 2017 10:57:01 PM jade.core.BaseService init
INFORMAÇÕES: Service jade.core.management.AgentManagement initialized
jun 18, 2017 10:57:01 PM jade.core.BaseService init
INFORMAÇÕES: Service jade.core.messaging.Messaging initialized
jun 18, 2017 10:57:01 PM jade.core.BaseService init
INFORMAÇÕES: Service jade.core.resource.ResourceManagement initialized
jun 18, 2017 10:57:01 PM jade.core.BaseService init
INFORMAÇÕES: Service jade.core.mobility.AgentMobility initialized
jun 18, 2017 10:57:01 PM jade.core.BaseService init
INFORMAÇÕES: Service jade.core.event.Notification initialized
jun 18, 2017 10:57:01 PM jade.core.AgentContainerImpl joinPlatform
INFORMAÇÕES: ----------------------------------------
Agent container Main-Container@192.168.65.1 is ready.
--------------------------------------------
Hallo! Buyer-agent bookBuyer1@192.168.65.1:1099/JADE is ready.
Target book is O colecionador
Trying to buy O colecionador
Found the following seller agents:
bookSeller2@192.168.65.1:1099/JADE
bookSeller1@192.168.65.1:1099/JADE
Attempt failed: O colecionador not available for sale
```

Figura 31 – Saídas do console: inicialização da plataforma.

```
Trying to buy O colecionador
Found the following seller agents:
bookSeller2@192.168.65.1:1099/JADE
bookSeller1@192.168.65.1:1099/JADE
Attempt failed: O colecionador not available for sale
Trying to buy O colecionador
Found the following seller agents:
bookSeller2@192.168.65.1:1099/JADE
bookSeller1@192.168.65.1:1099/JADE
Attempt failed: O colecionador not available for sale
Trying to buy O colecionador
Found the following seller agents:
bookSeller2@192.168.65.1:1099/JADE
bookSeller1@192.168.65.1:1099/JADE
Attempt failed: O colecionador not available for sale
Trying to buy O colecionador
Found the following seller agents:
bookSeller2@192.168.65.1:1099/JADE
bookSeller1@192.168.65.1:1099/JADE
Attempt failed: O colecionador not available for sale
Trying to buy O colecionador
Found the following seller agents:
bookSeller2@192.168.65.1:1099/JADE
bookSeller1@192.168.65.1:1099/JADE
Attempt failed: O colecionador not available for sale
```

Figura 33 – Saídas do console: tentativas de compra do agente comprador

Ao inserir no catálogo o livro procurado pelo agente comprador (Figura 34), o agente comprador encontra o livro desejado e realiza sua compra (Figura 35).

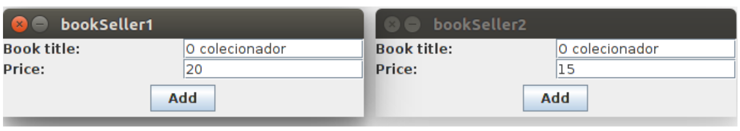

Figura 34 – Interface para catalogação dos livros.

```
Trying to buy 0 colecionador
Found the following seller agents:
bookSeller2@192.168.65.1:1099/JADE
bookSeller1@192.168.65.1:1099/JADE
0 colecionador sold to agent bookBuyer1@192.168.65.1:1099/JADE
0 colecionador successfully purchased from agent bookSeller2@192.168.65.1:1099/JADE
Price = 15
Buyer-agent bookBuyer1@192.168.65.1:1099/JADE terminating.
```

Figura 35 – Saídas do console: finalização da compra.

B.2 *Master-Slave*

B.2.1 Contexto

O exemplo a seguir baseia-se na Figura 17 apresentada na Seção 6.3. Em resumo, um agente *ConcreteCashFlowAgent* deseja obter o balanço total de pagamentos da sua caixa registradora.

B.2.2 Preparação do ambiente

O exemplo foi rodado em sistema operacional Ubuntu 16.04, utilizando a IDE Eclipse Neon versão 4.6.3. Os seguintes passos devem ser seguidos para rodar este exemplo e os exemplos a seguir:

1. Faço o download do código através do GitHub[10] e descomprima a pasta;
2. Abra a IDE Eclipse Neon;
3. Clique em *File* > *Import* > *General* > *Existing Projects into Workspace*;
4. Selecione a pasta que contém o código;
5. Clique em *Finish*;
6. No menu *Package Explorer* navegue até a classe *Main* do *package* ***master_s****laveecliqueem****Run***.

B.2.3 Classe *Main*

Basicamente, o *cashFlowAgent*, instância de *ConcreteCashFlowAgent*, precisa ler um arquivo CSV para que, em seguida, obtenha o balanço todos dos pagamentos realizados naquele dia.

```
1  package master_slave;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          //Read the csv file with the Cash Register
```

[10] https://github.com/tainarareis/MultiagentsSystemsCatalog

```
7           ConcreteCashFlowAgent cashFlowAgent = new ConcreteCashFlowAgent();
8           cashFlowAgent.readCSV();
9
10          //Get total balance of payments
11          cashFlowAgent.getResult();
12      }
13  }
```

B.2.4 Classes *MasterAgent* *ConcreteCashFlowAgent*

A classe *MasterAgent* é interface para que a classe concreta *ConcreteCashFlowAgent* e outras concretas que virem a existir possam se basear.

```
1   package master_slave;
2
3   public interface MasterAgent {
4
5       abstract void readCSV();
6
7       abstract void getResult();
8   }
```

O *ConcreteCashFlowAgent* possui uma instância *slave* do *ConcreteSlaveAgent* para realizar tarefas de baixo nível para ele. Os métodos *readCSV()* e *getResult()* têm como objetivo controlar como e quando o *slave* executará tais tarefas.

```
1   package master_slave;
2
3   import java.io.BufferedReader;
4   import java.io.FileReader;
5   import java.io.IOException;
6
7   public class ConcreteCashFlowAgent {
8
9       ConcreteSlaveAgent slave = new ConcreteSlaveAgent();
10
11      protected void readCSV() {
12      //Inform file Path
13          String csvFile =
                "/home/tainara/eclipse-workspace/MultiagentSystemsCatalog/src/master_slave/cash_register.csv";
14
15          //Each new line is recognized by empty string
16          String line = "";
17
18          //Comma is used as separator
19          String cvsSplitByComma = ",";
20
21          try (BufferedReader br = new BufferedReader(new FileReader(csvFile))) {
22
23              while ((line = br.readLine()) != null) {
24
25
26                  String[] field = line.split(cvsSplitByComma);
27                  slave.addPayment(Float.parseFloat(field[1]),
28                                   Float.parseFloat(field[2]));
```

```
29
30                  System.out.println("Payment [Item= " + field[0] +
31                          " , Quantity= " + field[1] +
32                          " , Amount=" + field[2] + "]");
33
34              }
35
36          } catch (IOException e) {
37              e.printStackTrace();
38          }
39      }
40
41      protected void getResult() {
42          float total = slave.getTotalBalance();
43          System.out.println("The total balance of payments was $" + total);
44      }
45
46  }
```

B.2.5 Classes *SlaveAgent* e *ConcreteSlaveAgent*

A classe *SlaveAgent* possui os métodos comuns para futuros *slaves* deste contexto. A classe *ConcreteSlaveAgent* implementa tais métodos. Basicamente, o *slave* pode adicionar um pagamento (*addPayment(float quantity, float amount)*) e também obter o balanço final (método *getTotalBalance()*).

```
1  package master_slave;
2
3  public abstract class SlaveAgent {
4
5      protected abstract void addPayment(float quantity, float amount);
6
7      protected abstract float getTotalBalance();
8
9  }
```

```
1  package master_slave;
2
3  public class ConcreteSlaveAgent extends SlaveAgent {
4
5      private float totalBalance;
6
7      protected void addPayment(float quantity, float amount) {
8          float result = quantity * amount;
9          this.totalBalance =+result;
10     }
11
12     protected float getTotalBalance() {
13         return this.totalBalance;
14     }
15
16 }
```

B.2.6 Resultados da execução

Apenas para exemplificar este contexto e o fluxo de eventos que ocorreria quando executada a implementação apresentada, a Figura 36 apresenta a saída no console da IDE Eclipse Neon.

```
<terminated> Main (2) [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Jun 23, 2018, 8:39:25 PM)
The requested service can be executed by the Functional Manager 1.
Functional Manager 1: Contacting functional unit to execute service...
         Access < Tidbits > < Use Wais | Use FTP >
         Access < InfoMac >
         Get From < Seller >
         Access < News >
Service finished.
```

Figura 36 – Saídas do console: *Master-Slave* executa o serviço solicitado.

B.3 *Blackboard*

B.3.1 Contexto

O exemplo a seguir baseia-se no contexto apresentado na Seção 6.4. Em resumo, o *Blackboard* recebe uma ordem de uma *Company*, e deve se encarregar de fazer com que esta se cumpra dentro do tempo delimitado, a partir da cooperação dos agentes *SupplierAgent1* e *SupplierAgent2*.

B.3.2 Preparação do ambiente

Devem ser seguidos os mesmos passos da Seção B.4.2, porém, deve-se optar pelo *package* ***blackboard***.

B.3.3 Classe *Blackboard*

O *Blackboard* conhece tanto a *Company* (instância *oliverEnterprise*), quanto o *ManufacturerAgent*, e os agentes *SupplierAgent1* e *SupplierAgent2*. Inicialmente, *oliverEnterprise* publica o leilão (método *publishAuction()*). O *Blackboard* passa a conhecer quais são as informações do leilão e notifica os *supplier agents*. Cada agente realiza um lance, informando quanto tempo levaria para realizar a entrega do produto solicitado pela *oliverEnterprise*. O *ManufacturerAgent*, coleta os lances e aplica seu critério de seleção para escolher qual agente deve realizar a entrega. Em seguida, o *Blackboard* prossegue controlando o fluxo de entrega até que o produto chegue ao destino final (método *controlDelivery(String selectedAgent)*.

```
1 package blackboard;
2
3 import java.util.HashMap;
```

```
4  import java.util.Map;
5
6  /**
7   * Blackboard is responsible for controlling the auction for supply distribution.
8   */
9  public class Blackboard {
10
11     private static Company oliverEnterprise = new Company();
12     private static ManufacturerAgent manufacturerAgent = new ManufacturerAgent();
13     private static Map auctionInformations = new HashMap<>();
14     private static Map bids = new HashMap<>();
15
16     private static SupplierAgent1 supplierAgent1 = new SupplierAgent1();
17     private static SupplierAgent2 supplierAgent2 = new SupplierAgent2();
18
19     public static void main(String[] args) {
20
21         auctionInformations = oliverEnterprise.publishAuction();
22         supplierAgent1.notifyAgent(auctionInformations);
23         supplierAgent2.notifyAgent(auctionInformations);
24
25         bids.put("Supplier Agent 1", supplierAgent1.bid());
26         bids.put("Supplier Agent 2", supplierAgent2.bid());
27
28         manufacturerAgent.applySelectionCriteria(bids);
29         controlDelivery(manufacturerAgent.getSelectedAgent());
30
31     }
32
33     private static void controlDelivery(String selectedAgent) {
34
35         Boolean deliveryProblems;
36
37         switch (selectedAgent) {
38             case "Supplier Agent 1":
39                 supplierAgent1.deliver();
40                 deliveryProblems = supplierAgent1.reportDeliveryProblems();
41                 if (deliveryProblems == false) {
42                     supplierAgent1.finishDelivery();
43                     break;
44                 } else {
45                     String alternativeAgent = manufacturerAgent.getAlternativeAgent();
46                     controlDelivery(alternativeAgent);
47                     break;
48                 }
49             case "Supplier Agent 2":
50                 supplierAgent2.deliver();
51                 deliveryProblems = supplierAgent2.reportDeliveryProblems();
52                 if (deliveryProblems == false) {
53                     supplierAgent2.finishDelivery();
54                     break;
55                 } else {
56                     String alternativeAgent = manufacturerAgent.getAlternativeAgent();
57                     controlDelivery(alternativeAgent);
58                 }
59         }
60     }
61 }
```

B.3.4 Classe *ManufacturerAgent*

O lance mais apropriado é selecionado pelo *ManufacturerAgent* com base em critérios de seleção (método *applySelectionCriteria(Map bids)*), que neste caso é o lance cujo tempo de entrega associado for o menor. Se, por ventura, um problema de entrega ocorre com um dos fornecedores, o *ManufacturerAgent* busca um fornecedor alternativo para substituí-lo.

```
1  package blackboard;
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  /**
7   * ManufacturerAgent applies the selection criteria to select the supplier dynamically.
8   */
9  public class ManufacturerAgent {
10
11     Map bids = new HashMap<>();
12     String selectedAgent;
13     String alternativeAgent;
14
15     protected void applySelectionCriteria(Map bids) {
16         int sa1DeliverTime = (int) bids.get("Supplier Agent 1");
17         int sa2DeliverTime = (int) bids.get("Supplier Agent 2");
18
19         if (sa1DeliverTime < sa2DeliverTime) {
20             this.setSelectedAgent("Supplier Agent 1");
21             this.setAlternativeAgent("Supplier Agent 2");
22             System.out.println("Manufacturer Agent selected: Supplier Agent 1.");
23             System.out.println("Obs: If any problem occurs, Supplier Agent 2 will be the alternative supplier.");
24         } else {
25             this.setSelectedAgent("Supplier Agent 2");
26             this.setAlternativeAgent("Supplier Agent 1");
27             System.out.println("Manufacturer Agent selected: Supplier Agent 2.");
28             System.out.println("Obs: If any problem occurs, Supplier Agent 1 will be the alternative supplier.");
29         }
30     }
31
32     protected void setSelectedAgent(String selectedAgent) {
33         this.selectedAgent = selectedAgent;
34     }
35
36     protected String getSelectedAgent() {
37         return selectedAgent;
38     }
39
40     protected String getAlternativeAgent() {
41         System.out.println("Searching for alternative agent to end the delivery.");
42         return alternativeAgent;
43     }
44
45     protected void setAlternativeAgent(String alternativeAgent) {
46         this.alternativeAgent = alternativeAgent;
47     }
48
49 }
```

B.3.5 Classe *Company*

A *Company* publica um leilão no quadro-negro (através do método *publishAuction()*), informando: o endereço da entrega do produto (*Deliver Adress*), o produto que deseja que seja entregue (*Product Name*), a quantidade do mesmo (*Quantity*), e o tempo limite que o agente deve levar para entregá-lo (*Time limit to deliver*).

```
1  package blackboard;
2
3  import java.util.Arrays;
4  import java.util.HashMap;
5  import java.util.Map;
6
7  /*
8   * A company is interested in having a certain product delivered.
9   * So, it can publish an auction at the Blackboard to ask an agent to do it.
10  */
11 public class Company {
12
13      Map auctionInformations = new HashMap<>();
14
15      public Company() {
16          this.auctionInformations.put("Company Name", "Oliver Enterprise");
17          this.auctionInformations.put("Delivery Adress", "324 Street - Salt Lake City - UT");
18          this.auctionInformations.put("Product Name", "Wall paint");
19          this.auctionInformations.put("Quantity", 20);
20          this.auctionInformations.put("Time limit to deliver", 800);
21      }
22
23      protected Map publishAuction() {
24          System.out.println("Company Oliver Enterprise publishing auction informations to Blackboard...");
25          return this.auctionInformations;
26      }
27 }
```

B.3.6 Classe *SupplierAgent*

A classe *SupplierAgent* fornece os atributos e métodos comuns aos agentes fornecedores: o inventário de produtos (*inventory*), o tempo de entrega (*timeToDeliver*), as informações do leilão quando vir a receber alguma notificação sobre o mesmo(*auctionInformation*), e o controle de inventário (por meio dos métodos *addProduct(String productName, int code)* e *removeProductQuantity(String productName, int quantity)*).

```
1  package blackboard;
2
3  import java.util.HashMap;
4  import java.util.Map;
5
6  /**
7   * Each Supplier Agent has an inventory with a collection of products available,
8   * and a certain time to deliver. This agent can participate of company's auctions.
9   * If an auction is published at the Blackboard and the Supplier Agent has the called
10  * product, it can offer its service to to Company.
11  */
```

```
12 public class SupplierAgent {
13
14     Map inventory = new HashMap<>();
15     int timeToDeliver;
16     Map auctionInformation = new HashMap<>();
17
18     protected void addProduct(String productName, int code) {
19         this.inventory.put(code, productName);
20     }
21
22     protected void removeProductQuantity(String productName, int quantity) {
23         int oldQuantity = (int) this.inventory.get(productName);
24         this.inventory.put(productName, oldQuantity−quantity);
25     }
26
27     protected int getTimeToDeliver() {
28         return this.timeToDeliver;
29     }
30
31 }
```

B.3.7 Classes *SupplierAgent1* e *SupplierAgent2*

Os agentes fornecedores, *SupplierAgent1* e *SupplierAgent2*, herdam os atributos e métodos da classe pai (*SupplierAgent*). Além de serem capazes de controlar seu inventário, implementam métodos para: serem notificados quando um leilão é publicado (*notifyAgent(Map auctionInformations)*), realizarem lances no leilão (*bid()*), realizarem a entrega (*deliver()*), reportarem quaisquer problemas que os impeçam de finalizar a entrega (*reportDeliveryProblems()*), e, por fim, finalizarem a entrega (*finishDelivery()*).

```
1  package blackboard;
2
3  import java.util.Map;
4
5  /**
6   * SupplierAgent1 has its own inventory and time to deliver.
7   */
8  public class SupplierAgent1 extends SupplierAgent{
9
10     public SupplierAgent1() {
11         super();
12         this.inventory.put("Roof tile", 60);
13         this.inventory.put("Brick", 58);
14         this.inventory.put("Wooden door", 85);
15         this.inventory.put("Wall paint", 74);
16         this.timeToDeliver = 90;
17         System.out.println("Supplier Agent 1 added to Blackboard.");
18     }
19
20     protected void notifyAgent (Map auctionInformations) {
21         this.auctionInformation = auctionInformations;
22         System.out.println("Supplier Agent 1: Notified.");
23     }
24
25     protected int bid () {
```

```
26            int auctiontimeLimit = (int) this.auctionInformation.get("Time limit to deliver");
27            if (this.getTimeToDeliver() <= auctiontimeLimit) {
28                System.out.println("Supplier Agent 1, Bid = " + this.getTimeToDeliver() +".");
29                return this.getTimeToDeliver();
30            } else {
31                System.out.println("Supplier Agent 1, Bid = None.");
32                return 0;
33            }
34        }
35
36        protected void deliver () {
37            System.out.println("Supplier Agent 1 delivering " + this.auctionInformation.get("Product Name") +".");
38        }
39
40        protected Boolean reportDeliveryProblems() {
41            System.out.println("Supplier Agent 1 facing problems to deliver.");
42            return true;
43        }
44
45        protected void finishDelivery() {
46            System.out.println("Supplier Agent 1 finished delivery on time.");
47            this.removeProductQuantity((String) this.auctionInformation.get("Product Name"), (int)
                  this.auctionInformation.get("Quantity"));
48        }
49
50    }
```

```
1   package blackboard;
2
3   import java.util.HashMap;
4   import java.util.Map;
5
6   /**
7    * SupplierAgent2 has its own inventory and time to deliver.
8    */
9   public class SupplierAgent2 extends SupplierAgent{
10
11      public SupplierAgent2() {
12          super();
13          this.inventory.put("Tile", 95);
14          this.inventory.put("Sand", 86);
15          this.inventory.put("Eletrical wiring", 21);
16          this.inventory.put("Wall paint", 74);
17          this.timeToDeliver = 100;
18          System.out.println("Supplier Agent 2 added to Blackboard.");
19      }
20
21      protected void notifyAgent (Map auctionInformations) {
22          this.auctionInformation = auctionInformations;
23          System.out.println("Supplier Agent 2: Notified.");
24      }
25
26      protected int bid() {
27          int auctiontimeLimit = (int) this.auctionInformation.get("Time limit to deliver");
28          if (this.getTimeToDeliver() <= auctiontimeLimit) {
29              System.out.println("Supplier Agent 2, Bid = " + this.getTimeToDeliver() +".");
30              return this.getTimeToDeliver();
31          } else {
32              System.out.println("Supplier Agent 2, Bid = None");
```

```
33                  return 0;
34              }
35          }
36
37          protected void deliver() {
38              System.out.println("Supplier Agent 2 delivering " + this.auctionInformation.get("Product Name") +".");
39          }
40
41          protected Boolean reportDeliveryProblems() {
42              System.out.println("Supplier Agent 2 facing no problems to deliver.");
43              return false;
44          }
45
46          protected void finishDelivery() {
47              System.out.println("Supplier Agent 2 finished delivery on time.");
48          }
49
50  }
```

B.3.8 Resultados da execução

De modo a exemplificar este contexto e o fluxo de eventos que ocorreria quando executada a implementação apresentada, é apresentada a saída no console da IDE Eclipse Neon (Figura 37).

```
<terminated> Blackboard (1) [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Jun 24, 2018, 11:30:52 PM)
Supplier Agent 1 added to Blackboard.
Supplier Agent 2 added to Blackboard.
Company Oliver Enterprise publishing auction informations to Blackboard...
Publication = [Company Name=Oliver Enterprise, Delivery Adress=324 Street - Salt Lake - UT
, Product Name=Wall paint, Quantity=20, Time limit to deliver=800]
Supplier Agent 1: Notified.
Supplier Agent 2: Notified.
Supplier Agent 1, Bid = 90.
Supplier Agent 2, Bid = 100.
Manufacturer Agent selected: Supplier Agent 1.
Obs: If any problem occurs, Supplier Agent 2 will be the alternative supplier.
Supplier Agent 1 delivering Wall paint.
Supplier Agent 1 facing problems to deliver.
Searching for alternative agent to end the delivery.
Supplier Agent 2 delivering Wall paint.
Supplier Agent 2 facing no problems to deliver.
Supplier Agent 2 finished delivery on time.
```

Figura 37 – Saídas do console: *Blackboard* controla o solicitado.

B.4 MACRON

B.4.1 Contexto

O exemplo a seguir baseia-se no contexto apresentado na Seção 6.5. Em resumo, um agente *QueryManagerAgent* deseja obter um serviço, especificamente, o serviço *Get Online Reviews*. O diagrama apresentado na Seção 24 é base da implementação deste contexto.

B.4.2 Preparação do ambiente

Devem ser seguidos os mesmos passos da Seção B.4.2, porém, deve-se optar pelo *package* ***macron***.

B.4.3 Classe *Main*

A *Main*, nesta implementação, apenas recebe as requisições dos *Query Manager Agents*, e, no caso, apenas do *QueryManagerAgent*. Quando o serviço que este busca está disponível, é delegado que seja executado.

```
1  package macron;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          QueryManagerAgent queryManager = new QueryManagerAgent();
7          Boolean isServiceAvailable;
8
9          //QueryManager wants to retrieve all online reviews about some product,
10         isServiceAvailable = queryManager.findFunctionalManager("Get Online Reviews");
11
12         //The query can only occur if there is an agent that can execute it.
13         if (isServiceAvailable == true) {
14             queryManager.executeQuery();
15         }
16     }
17 }
```

B.4.4 Classe *QueryManagerAgent*

O *QueryManagerAgent* consulta o *OrganizationChartManagerAgent* para verificar se há algum agente que ofereça o serviço que busca, através da função *findFunctionalManager(String requestedService)*. Também delega ao *OrganizationChartManagerAgent* que controle a execução do serviço quando o mesmo é encontrado e está disponível, através da função *executeQuery()*.

```
1  package macron;
2
3  public class QueryManagerAgent {
4
5      String requestedService;
6      OrganizationChartManagerAgent ocmAgent = new OrganizationChartManagerAgent();
7
8      //Verify if there is any functional manager that can offer the requested service
9      protected Boolean findFunctionalManager(String requestedService) {
10         return this.ocmAgent.findFunctionalManager(requestedService);
11     }
12
13     //Delegates ocmAgent to control the query execution
14     protected void executeQuery() {
15         this.ocmAgent.ordenateServiceExecution();
```

```
16      }
17  }
```

B.4.5 Classe *OrganizationalChartManagerAgent*

O *OrganizationalChartManagerAgent* procura *Functional Managers* através do método *findFunctionalManager(String requestedService)* cujas *Functional Units* possam desempenhar o serviço *Get Online Reviews*. Sua principal função é conhecer os gerentes funcionais, sua disponibilidade, suas capacidades, e assim pode ser consultado pelo *Query Manager* a qualquer instante.

```
1   package macron;
2
3   import java.util.HashMap;
4   import java.util.Map;
5
6   public class OrganizationChartManagerAgent {
7
8       //OCM known agents
9       private FunctionalManager1Agent fm1Agent = new FunctionalManager1Agent();
10      private FunctionalManager2Agent fm2Agent = new FunctionalManager2Agent();
11
12      //All services this agents offers
13      private Map allServices = new HashMap<>();
14
15      //The agent localized to execute the service to the query manager
16      private String functionalManagerLocalized;
17
18      public OrganizationChartManagerAgent() {
19          this.allServices.put(fm1Agent.getService(), fm1Agent.getName());
20          this.allServices.put(fm2Agent.getService(), fm2Agent.getName());
21      }
22
23      protected Boolean findFunctionalManager(String requestedService) {
24          this.functionalManagerLocalized = (String) this.allServices.get(requestedService);
25          if (this.functionalManagerLocalized != null) {
26              System.out.println("The requested service can be executed by the " + this.functionalManagerLocalized +
                    ".");
27              return true;
28          } else {
29              System.out.println("The requested service has no functional manager that can do it.");
30              return false;
31          }
32      }
33
34      protected void ordenateServiceExecution() {
35          if (this.functionalManagerLocalized == "Functional Manager 1") {
36              this.fm1Agent.executeService();
37          } else if (this.functionalManagerLocalized == "Functional Manager 2") {
38              this.fm2Agent.executeService();
39          }
40      }
41
42  }
```

B.4.6 Classe *FunctionalManager1Agent*

A pedido do *QueryManagerAgent*, o *FunctionalManager1Agent* atribui tarefas a agentes dentro da sua *Functional Unit*. Esta é responsável por alocar os recursos necessários para realizar o serviço. Neste caso, os recursos são: *Tidbits, Wais,* FTP, *InfoMac*, *Seller*, e *News*. Todos são consultados para recuperar os *online reviews* solicitados. Caso houvesse sido solicitado um outro serviço, como, por exemplo, *Get Published Reviews*, seriam necessários os recursos *Tidbits, Library, Online Sources, Fax,* e *Seller*. Ou seja, pra executar diferentes serviços, seriam necessários adotar os mesmos recursos. Cabe ao *Functional Manager*, na estrutura MACRON tomar as decisões a cerca de como fazer a melhor utilização de tais recursos de modo dinâmico.

```
1  package macron;
2
3  public class FunctionalManager1Agent extends FunctionalManagerAgent {
4
5      public FunctionalManager1Agent() {
6          this.name = "Functional Manager 1";
7          this.service = "Get Online Reviews";
8      }
9
10     protected void executeService() {
11         System.out.println("Functional Manager 1: Contacting functional unit to execute service...");
12         this.alocateResources();
13         this.getOnlineReviews();
14         System.out.println("Service finished");
15     }
16
17     protected void alocateResources() {
18         //Strategy of resources alocation here
19     }
20
21     protected void getOnlineReviews() {
22         System.out.println("Access Tidbits < Use Wais | Use FTP >");
23         System.out.println("Access InfoMac");
24         System.out.println("Get From Seller");
25         System.out.println("Access News");
26     }
27
28 }
```

B.4.7 Resultados da execução

Apenas para exemplificar este contexto e o fluxo de eventos que ocorreria quando executada a implementação apresentada, a Figura 38 apresenta a saída no console da IDE Eclipse Neon.

```
<terminated> Main (2) [Java Application] /usr/lib/jvm/java-8-oracle/bin/java (Jun 23, 2018, 8:39:25 PM)
The requested service can be executed by the Functional Manager 1.
Functional Manager 1: Contacting functional unit to execute service...
          Access < Tidbits > < Use Wais | Use FTP >
          Access < InfoMac >
          Get From < Seller >
          Access < News >
t Service finished.
```

Figura 38 – Saídas do console: *FunctionalManager1Agent* executa o serviço solicitado.

B.5 *Generalized Partial Global Planning*

A seguir é apresentado um pseudo-código que simula o fluxo entre os agentes simulando as ações descritas na Seção 6.6.

```
1  package gpgp;
2
3  public class GPGP {
4
5      public static void main(String[] args) {
6
7      //At time=0 each agent creates its task structure
8      Agent agentA = new Agent();
9      agentA.addTask(A11, 3); //addTask(taskName, duration)
10     agentA.addTask(A12,3);
11
12     Agent agentB = new Agent();
13     agentB.addTask(B11,2);
14
15     Agent agentC = new Agent();
16     agentC.addTask(C11, 4);
17     agentC.addTask(C12, 1);
18
19     //At time=1, each agent makes its own schedule
20     agentA.schedule(A11, 1, 3); //schedule(taskName, startTime, endTime)
21     agentA.schedule(A12, 4, 6);
22     agentB.schedule(B11, 1, 2);
23     agentC.schedule(C12, 1, 1);
24
25     //At time=2, gpgp plans the tasks considering the global needs
26     GeneralizedPartialGlobalPlanner gpgp = new GeneralizedPartialGlobalPlanner();
27     winner = gpgp.generatePlanning(1, 2) //generates the winner for the interval from time 1 to 2
28     gpgp.notifyAgents(agentA, agentB, agentC); //notifies all agents about the winner of the interval
29
30     //Consider the winner was agentB...
31     agentB.executeTask(B11);
32     agentA.busyInterval(1, 2); //busyInterval(startTime, endTime)
33     agentC.busyInterval(1, 2);
34
35     //agentA reschedules, and agentC schedules task C11
36     agentA.schedule(A11, 5, 7);
37     agentA.schedule(A12, 2, 4);
38     agentC.schedule(C11, 3, 6);
39
40     //At time=3
41     winner = gpgp.generateaPlanning(3, 4);
42     gpgp.notifyAgents(agentA, agentB, agentC);
```

```
43
44      //Consider the winner was agentA with task A11...
45      agentA.executeTask(A11);
46      agentB.busyInterval(5, 7);
47      agentC.busyInterval(5, 7);
48
49      //agentC reschedules
50      agentC.schedule(C11, 7, 10);
51
52      //At time=4
53      winner = gpgp.generatePlanning(7, 10);
54
55      //Consider the winner was agentA with task C11...
56      agentC.executeTask(C11);
57
58      }
59  }
```