PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 01/2017

Explorando o Conceito de Software Consciente: Um Estudo Ancorado em Revisão Colaborativa

Marcio Ricardo Rosemberg Francisco José Plácido da Cunha
Roxana Portugal Joanna Pivatelli Larissa Torres Santos
Ana Maria Moura Ricardo Venieris Erica Riello
Bruno Olivieri Paulo Henrique Cardoso Alves Marília Guterres Ferreira
Julio Cesar Sampaio do Prado Leite

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 01/17 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Janeiro, 2017

Explorando o Conceito de Software Consciente: Um Estudo Ancorado em Revisão Colaborativa

Marcio Ricardo Rosemberg[1] Francisco José Plácido da Cunha[1] Roxana Portugal[1] Joanna Pivatelli[1] Larissa Torres Santos[2] Ana Maria Moura[1] Ricardo Venieris[1] Erica Riello[1] Bruno Olivieri[1] Paulo Henrique Cardoso Alves[1] Marília Guterres Ferreira[1] Julio Cesar Sampaio do Prado Leite[1]

[1]Departamento de Informática – Pontifícia Universidade Católica do Rio de Janeiro (PUC-RIO); [2]Universidade Federal Rural do Rio de Janeiro (UFRRJ)

{mrosemberg, fcunha, rportugal, jpivatelli, rvenieris, julio} @inf.puc-rio.br; {llarissa, anammmoura, ericaflriello, mariliagf} @gmail.com; bruno@olivieri.com.br, ph.alves@live.com

Resumo: O conceito de consciência de software procura tratar a possibilidade de o software passar a ter comportamento que possibilite sua auto adaptação a eventuais mudanças em sua interação com o contexto em que está inserido. Esta adaptação, portanto, seria resultante de um estado de consciência ante as condições que enfrenta e os objetivos desejados. Podemos entender a ideia como uma evolução do próprio conceito de evolução de software na medida em que a evolução passa pela consciência de evoluir. A exploração desse conceito como um requisito não funcional é uma estratégia que possibilita um detalhamento do conceito, bem como da possibilidade de classificação de eventuais operacionalizações a serem realizadas para satisfazer a contento o requisito não funcional de consciência aplicado a artefatos de software. Esse trabalho parte de um estudo fundacional que propõe um mapeamento inicial de uma rede de metas flexíveis para representar o conceito de consciência. Nosso objetivo é confirmar essa rede como base fundacional, bem como instancia-lo através de um conjunto de propostas.

Palavras Chave: Consciência de Software, Auto adaptação, Metas Flexíveis

Abstract. The concept of software awareness addresses the possibility of the software to modify its behavior in order to allow its adaptation to changes occurring in the environment in which the software is running. Such adaptation would therefore be the result of a state of awareness in contrast to the conditions it meets and the desired goals. We can understand the idea as an evolution of the concept of software evolution in that evolution involves the awareness to evolve. The exploration of this concept as a non-functional requirement is a strategy that enables a concept detailing, as well as the possibility of operationalization classifications to be carried out to achieve the non-functional requirement of awareness applied to software artifacts. This work is part of a foundational study that proposes an initial mapping of a network of soft goals to represent the concept of awareness. Our primary objective is to confirm that network as foundational basis as well as to instantiate it through a set of published proposals that contribute to the awareness soft goals. The approach we adopted was based on a collaborative strategy.

Keywords: Software Awareness, Self-adaptation, Soft Goals

This work has been sponsored by the Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Sumário

1 Introdução 1

2 Marco Teórico 2

3 Processo de Revisão Colaborativa 3

4 Trabalhos Discutidos 7

4.1 Designing an adaptive computer-aided ambulance dispatch system with Zanshin: an experience report 7

4.2 Environmental awareness intrusion detection and prevention system toward reducing false positives and false negatives 11

4.3 Profiling. Memory in Lua. 16

4.4 Automated support for diagnosis and repair 21

4.5 MORPH: A Reference Architecture for Configuration and Behaviour Self-Adaptation 25

4.6 UCCA: A Unified Cooperative Control Architecture for UAV missions 30

4.7 Predictive Monitoring of Business Processes 37

4.8 Design of the Outer-tuning framework: self-tuning and ontology for relational 40

4.9 From Unknown to Known Impacts of Organizational Changes on Socio-technical Systems 44

4.10 JAT A Test Automation Framework for Multi-Agent Systems 47

5 Comparativo entre os Estudos de Caso 51

6 Conclusões 52

7 Trabalhos Futuros 53

Referências Bibliográficas 54

1 Introdução

O conceito de evolução é intrinsecamente ligado a Software, como bem fundamentou Lehman (Lehman e Ram 2001), e que Brooks acredita ser um dos graves bloqueios ao pleno entendimento de sua engenharia (de Souza Cunha 2014). Brooks diz: " But the linear, step-by-step Rational Model is misled in goal and approach.", and "Its Rational Model leads to the too-early binding of requirements, leading in turn to bloated products and schedule/budget/performance disasters". Essa citação no contexto da ciência do desenho reflete seu entendimento que software evolui e, portanto, o ciclo de fases é inadequado. Por outro lado, Lehman na sua sétima lei diz: "Processos de evolução de software são sistemas de retroalimentação em múltiplos níveis, em múltiplos laços (loops) e envolvendo múltiplos agentes".

Além da engenharia de software, diferentes áreas da ciência da computação têm investido em pesquisas que procuram tratar da questão de evolução, desde de estudos em inteligência artificial que focam no aprendizado, bem como estudos em biologia computacional que usam estratégias computacionais no estudo de estruturas da biologia. Em particular, em engenharia de software e em software de maneira geral, grande ênfase é dada a questões de adaptação, ou seja a capacidade de o software evoluir, adaptando-se a novas condições, quer internas ou externas ao software.

Nessa visão sobre evolução, entendemos o conceito de consciência de maneira mais restrita e relacionada a "estar a par do que acontece", deixando de lado uma abordagem mais ampla de consciência com a de Goguen (Goguen 2003). Embora restrita, o uso do termo consciência, traz para a engenharia de software a preocupação com o processo de percepção traduzido pelo prefixo auto (de Souza Cunha 2014), englobando aspectos de autogerenciamento e autogerenciamento base para estudos de computação autônoma (Sterritt e Hinchey 2006). Portanto, avanços no entendimento do que seja software consciente ajudará possíveis alternativas para tratar do problema de evolução de software, que na visão de Leite (Leite 2012) está intrinsecamente ligado a três pontos: 1) o software é mutável qualquer que seja o hardware, 2) existe um consenso social de que software é fácil de mudar, e 3) software está em todo lugar.

Nossa questão de pesquisa é como identificar características de consciência de software em contribuições científicas de forma a incrementar o conhecimento sobre esse conceito. Nesse sentido, nossa pesquisa procura confirmar uma base fundacional elaborada por Souza Cunha (de Souza Cunha 2014) bem como instancia-la com operacionalizações oriundas de um conjunto de artigos científicos que propõem estratégias que contribuem positivamente para a meta flexível, requisitos não funcionais de consciência. Procuramos operacionalizações para as metas flexíveis de consciência e as correlações entre elas.

Nossa pesquisa baseou-se na tese de Souza Cunha (de Souza Cunha 2014) e na revisão colaborativa de uma série de artigos com indícios de propostas com viés de autonomia. Fundamentalmente procuramos fazer um emparelhamento entre a base fundacional, expressa em uma rede de relacionamentos e dez artigos selecionados e revisados colaborativamente. E como resultado discussões é apresentada uma lista os resumos dos artigos debatidos, assim como seu posicionamento ante a base proposta por Souza Cunha.

Além da Introdução, essa monografia está organizada em sete outras Seções, que tratam: do marco teórico (Seção 2), do processo colaborativo utilizado (Seção 3), da operacionalização do SIG de consciência, através dos resumos (Seção 4), da comparação entre

as operacionalizações encontradas, com vistas a identificação de padrões (Seção 5), e da conclusão, onde resume-se o alcançado, apontando para futuros estudos (Seções 6 e 7)

2 Marco Teórico

A pesquisa aqui relatada fundamenta-se no "NFR framework" (Chung 2000), trabalho oriundo da Universidade de Toronto, na qual o SIG (Softgoal Interdependence Graph) lida com requisitos não-funcionais. O SIG usa a premissa de satisfação a contento descrita por Herbet Simon (Simon 1996), para lidar com metas flexíveis. Essa representação é pioneira porque trata justamente da capacidade de lidar com gradações entre relacionamentos de uma forma subjetiva, sem necessária metrificação. Sua semântica é definida por um catálogo de avaliação qualitativa, que mapeia ocorrências da origem para o destino usando uma função crescente.

Originating Label		**Contribution Link Type**						
Label	Name	Make	Break	Help	Hurt	Some+	Some-	Unknown
✓	Satisficed	✓	✗	✓.	✗.	✓.	✗.	?
✓.	Partially Satisficed	✓.	✗.	✓.	✗.	✓.	✗.	?
↯	Conflict	↯	↯	↯	↯	↯	↯	?
?	Unknown	?	?	?	?	?	?	?
✗.	Partially Denied	✗.	✓.	✗.	✓.	✗.	✓.	?
✗	Denied	✗	✓.	✗.	✓.	✗.	✓.	?

Figura 1 - Catálogo de Avaliação Qualitativa (Horkoff et al. 2006)

O uso do "NFR Framework" foi base para a aquisição de conhecimento sobre transparência de software (Leite 2010). Este conhecimento foi mapeado em um SIG e depois operacionalizado através da estratégia GQO (Goal, Question, Operationalization) (Serrano e Leite 2011) que permite com que: dado um SIG seja possível através de padrões de perguntas identificar possíveis alternativas para a operacionalização de uma meta-fléxivel. O uso do GQO com o SIG de transparência resultou no catálogo de transparência de software, detalhado em um Wiki (Catálogo) (Grupo de Engenharia de Requisitos da PUC-Rio, 2012).

Usando o " NFR Framework" e o conceito do GQO, Souza Cunha (de Souza Cunha, 2014) mapeou o conhecimento sobre consciência de software, bem como instanciou operacionalizações para demonstração do conceito. Segundo Souza Cunha, a Consciência de Software tornou-se um requisito importante na construção de sistemas com capacidade de auto adaptação.

Para Souza Cunha: "Consciência de Software (Software Awareness) tornou-se um requisito importante na construção de sistemas com capacidade de autoadaptação. Para que aplicações de software possam melhor se adaptar a mudanças nos diversos ambientes em que operam, ter consciência (no sentido de perceber e entender esses ambientes e a seu próprio funcionamento nestes ambientes) é fundamental. Entretanto, mesmo em um nível básico aplicado a software, consciência é um requisito difícil de definir. Nosso trabalho propõe a organização de um catálogo para o requisito de consciência de software, com mecanismos para instanciação e uso do conhecimento armazenado neste catálogo na modelagem e implementação de software para problemas onde a auto adaptação, e por consequência consciência, sejam requisitos chave."

Nosso trabalho, como já dito, se fundamenta no SIG proposto por Souza Cunha e que pode ser visto abaixo numa versão parcial, limitada as metas flexíveis mais abstratas.

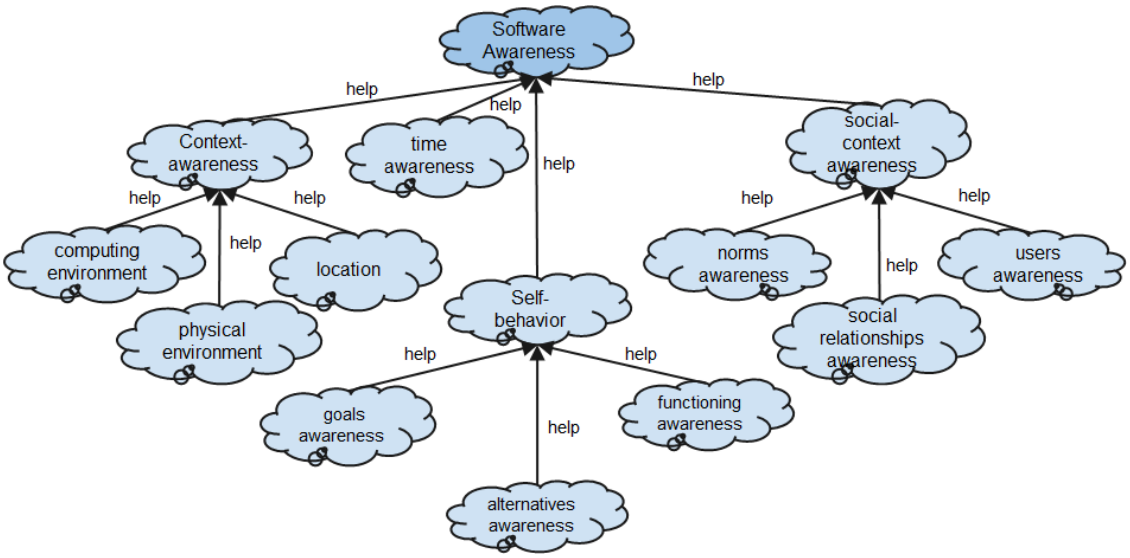

Figura 2 – Os níveis mais abstratos do SIG de Consciência de Software (de Souza Cunha 2014)

3 Processo de Revisão Colaborativa

Para responder a nossa questão de pesquisa, isto é saber sobre a adequação do SIG de consciência, bem como identificar possíveis operacionalizações, utilizamos um processo colaborativo de revisão bibliográfica. Esse processo pode ser descrito pelo modelo 3C de colaboração (Ellis et. al 1991) (Fuks et al. 2005)

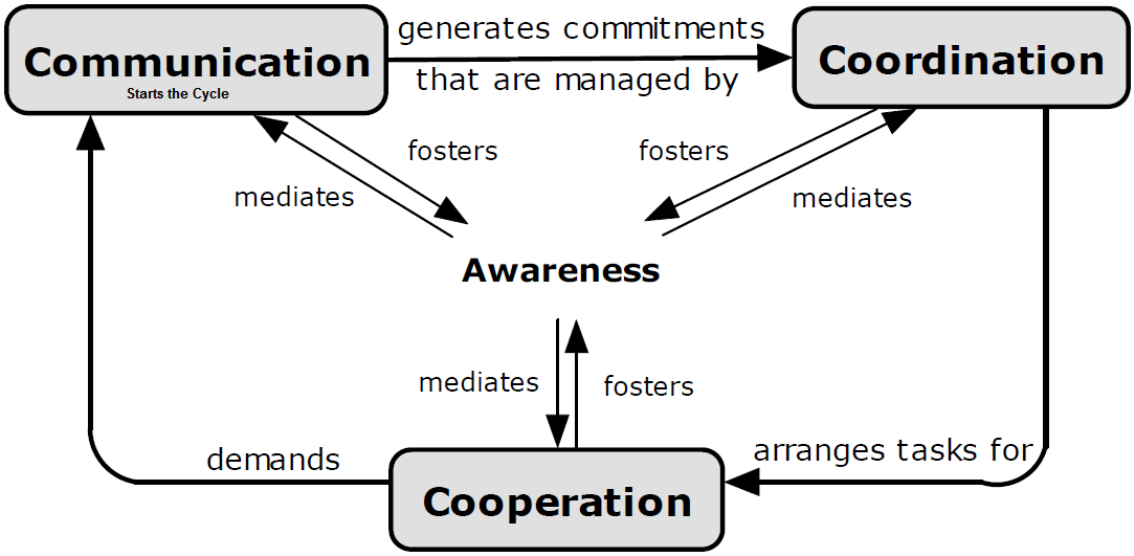

Figura 3 – Instanciação do Modelo 3C (Fuks et al. 2005)

Nosso processo de trabalho instanciou o modelo 3C como um ciclo, onde as tarefas podem ocorrer tanto de modo assíncrono ou síncrono, sem necessariamente existir uma ordem de precedência, entre elas, mas seguindo a tríade/fluxo como na Figura 3.

O processo que estamos tratando contou com a diferentes atores, agentes, papéis e posições. Foram 11 (onze) agentes reais, uns instanciando o ator pesquisador e outro

instanciando o ator pesquisador sênior. Os agentes desempenham os papéis de coordenador, revisor, e revisor mestre, ocupando as posições doutor, doutorandos (5) e mestrandos (6). A Figura 4 descreve o diagrama de atores (do Prado Leite, et al., 2007) utilizado. Estes atores se reuniram em 16 sessões semanais de três horas ao longo dos meses de Agosto (3), Setembro(4), Outubro(4), Novembro(4) e Dezembro(1) de 2015.

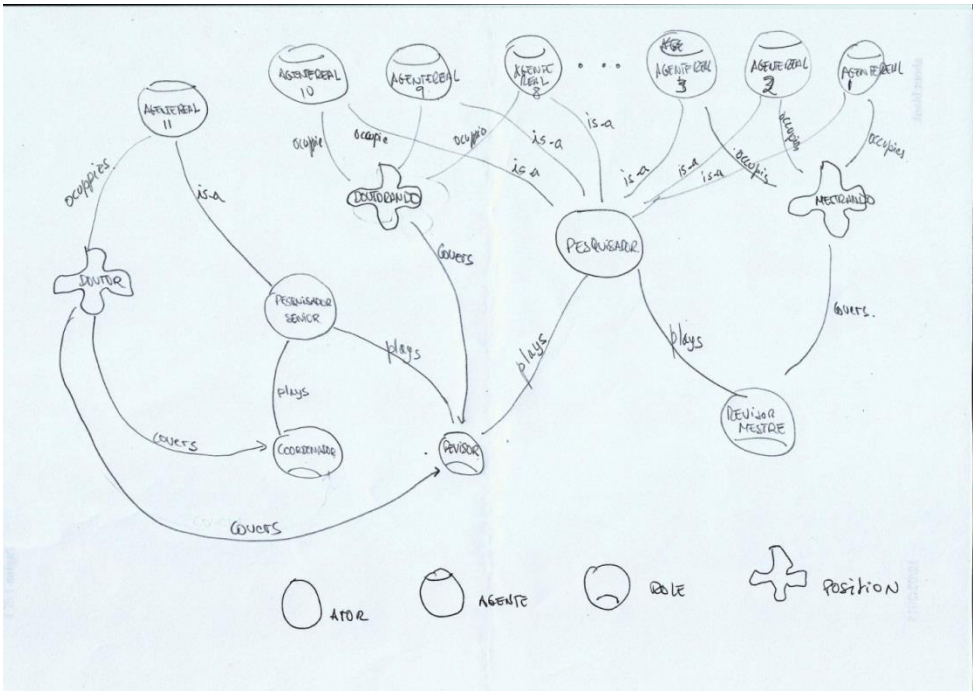

Figura 4 – Diagrama de Atores

O processo 3C inicia-se com o pesquisador sênior explanando (comunicação) os conceitos do "NFR Framework", da tese de Souza Cunha e do padrão GQO descritos na Seção 2. Os pesquisadores, dividiam-se em: agentes com alguma noção do marco teórico e agentes que viam os conceitos pela primeira vez. A primeira sessão serviu para propor o embasamento teórico necessário e nivelar o conhecimento dos pesquisadores.

Nessa primeira sessão, o pesquisador sênior aponta (coordenação) dois artigos: Awareness Requirements (Silva Souza, et al., 2012) e o estudo de caso da seção 4.1 (Silva Souza & Mylopoulos, 2015), como sementes para que todos façam a leitura (papel de revisor). Nessa seção também foi discutido pelos pesquisadores (coordenação) a forma final do relatório a ser apresentado.

Na segunda sessão, o pesquisador sênior projetou a tese de doutorado de Souza Cunha na sala de reunião, centrando-se no SIG da Figura 2 e questionou (comunicação) aos pesquisadores como enquadrar os artigos lidos. Seguiu-se um debate (cooperação) para ligar o material lido com o SIG, onde o pesquisador sênior utilizou o quadro branco para escrever o mapeamento (comunicação/cooperação). Essa operação resultou em refazer o mapeamento em função da discussão (cooperação). Neste processo, algumas vezes, o pesquisador sênior atua como revisor (cooperação) e algumas vezes como coordenador (coordenação). Para cada sugestão feita com base no SIG (Figura 2), seguiu-se o detalhamento da meta flexível na tese (projetada na sala), que detalhava uma série de perguntas (GQO). Estas perguntas ajudam (cooperação) na identificação de onde enquadrar a proposta do artigo como operacionalizações das metas flexíveis de consciência constantes

do SIG da tese. As operacionalizações foram mapeadas no quadro branco pelo pesquisador sênior utilizando a relação "help" (Figura 1) ou a relação "and" conforme o caso.

Também na segunda sessão, repetiu-se o processo supramencionado para o segundo artigo apontado pelo pesquisador sênior. Além disso, o pesquisador sênior estabeleceu (comunicação) que os outros artigos seriam escolhidos pelos demais pesquisadores e seriam confirmados (coordenação) pelo pesquisador sênior, posteriormente.

Nas sessões seguintes, repetiu-se o processo iniciado na primeira sessão, todavia com um aumento de produtividade, porque os pesquisadores passaram a discutir (cooperação) dois artigos por sessão.

Um exemplo do processo de cooperação: durante o processo de discussão das primeiras leituras, especificamente na consciência de contexto, havia uma dúvida recorrente se estávamos tratando de ambiente computacional ou ambiente físico. Esta dúvida somente ficou superada, quando discutimos o artigo do IPS/IDS (Sourour et al. 2009). Conseguimos concluir que para estar no ambiente físico, era necessário a monitoração de grandezas físicas (exemplos: temperatura, campo magnético, posição geográfica, potencial elétrico), enquanto que no ambiente computacional, a monitoração era relativa aos recursos computacionais (bytes Tx/Rx, % de uso de CPU, memória consumida ou liberada, cache hit/miss, espaço em disco, entre outros).

A escolha dos artigos pelos pesquisadores, de certa maneira, estabeleceu uma certa distribuição aleatória dos artigos a serem lidos, tendo em vista que cada um elegia um artigo segundo seus próprios interesses de pesquisa. Isso foi bastante relevante, porque os artigos vieram de diferentes áreas da ciência da computação e, portanto, não estavam focados somente no tema de adaptação ou computação autonômica. Acreditamos que isso ajudou na questão de cobertura (completude) do SIG.

Apesar disso, é relevante atentar que os artigos selecionados enquadraram-se no modelo MAPE-K (Monitor, Analize, Plan, Excute and Knowledge) (Kephart e Chess 2003) total ou em parte, mesmo levando-se em conta que os pesquisadores, em sua maioria, não eram profundos conhecedores do modelo e muitos dos artigos selecionados, sequer fazem referência ao referido modelo.

Pudemos observar que a consciência de software inicia-se na monitoração do contexto, do tempo ou do contexto social, descrito por Souza Cunha (Figura 2).

A partir da monitoração, o software podia simplesmente reportar a monitoração ou adaptar-se de alguma forma. Nos artigos selecionados, quando a adaptação ocorria, ela passava pelas fases de planejamento e análise, consultando e atualizando de uma base de conhecimento para enfim decidir o que fazer, num ciclo contínuo até que as metas fossem atingidas ou, em certos casos, executar a programação preparada para quando nenhuma das metas poderia ser alcançada.

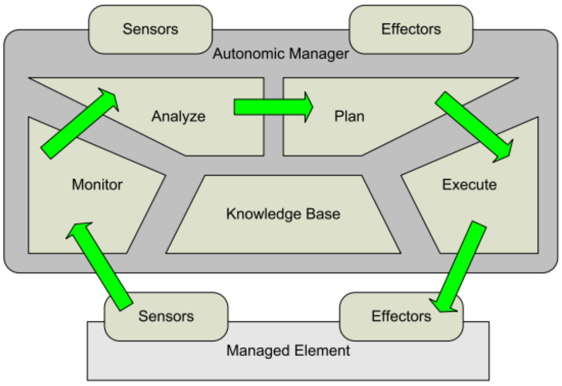

Figura 5 – Modelo MAPE-K

A Figura 6 abaixo ilustra um instantâneo do quadro branco elaborado durante uma sessão. As formas de nuvem são metas flexíveis, as retangulares são operacionalizações e os relacionamentos são setas. A cada sessão o quadro branco era fotografado para futuro redesenho, a acontecer durante o trabalho do revisor mestre.

Cada artigo, discutido segundo o processo descrito acima, foi distribuído (**coordenação**) pelos pesquisadores entre si (**cooperação**) de forma que cada pesquisador assumiu o papel de revisor mestre para um dos artigos revistos colaborativamente.

Ao revisor mestre cabia a escrita do resumo do artigo e da justificativa da operacionalização adicionada ao SIG durante o processo de **cooperação** descrito acima. Esses resumos são a memória de trabalho retrabalhada e apresentada (comunicação), para cada artigo, sendo revistos (**cooperação**) pelo pesquisador sênior e pesquisadores. A forma final de cada artigo foi baseada numa estrutura discutida numa das últimas sessões e envolveu também a padronização da linguagem gráfica utilizada para mostrar os componentes do grafo oriundos da operacionalização identificada para cada artigo. Essa forma final é apresentada na Seção 4, a seguir.

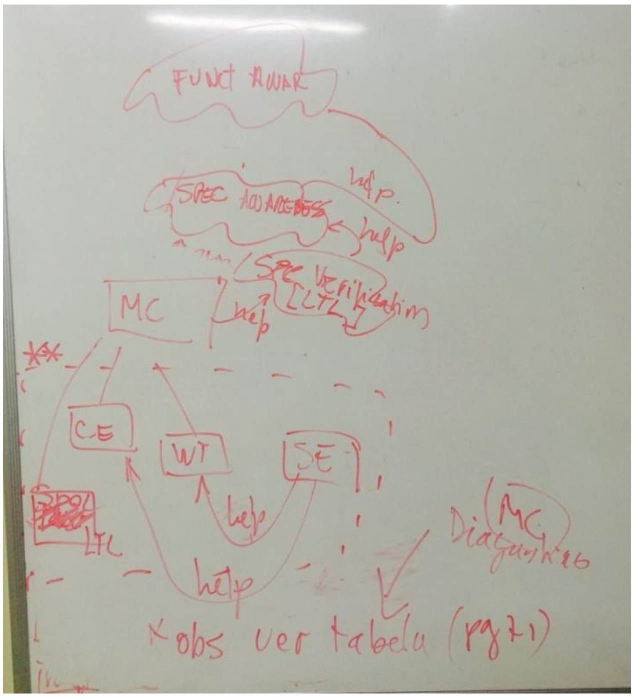

Figura 6 – Exemplo de um quadro branco

4 Trabalhos Discutidos

Nesta seção selecionamos 10 (dez) estudos de caso, que foram discutidos nas reuniões presenciais, onde nos concentramos em modelar as operacionalizações realizadas por cada trabalho em cada aspecto de consciência de software, conforme descrito por Souza Cunha.

É importante mencionar que cada estudo de caso apresenta as aspectos de consciência diferentes entre si, ou seja, na sua grande maioria, os estudos de caso não apresentam consciência em todos os aspectos.

4.1 Designing an adaptive computer-aided ambulance dispatch system with Zanshin: an experience report

4.1.1 Introdução do Estudo de Caso

Os autores do artigo (Silva Souza e Mylopoulos 2015), estudando sistemas adaptativos, verificou que tais sistemas precisam de alguma forma de auto monitoração em forma de um looping de retroalimentação. Por sua importância, os autores entenderam que a adaptabilidade deveria ser considerada logo nos estágios iniciais do desenvolvimento de software.

Os autores definiram a monitoração como uma "prótese arquitetural", visto que um software não pode determinar se seu desempenho está deficiente sem que seus requisitos estejam modelados, mesmo de forma implícita, a fim de que o software possua métricas para comparação. Para responder à pergunta de pesquisa: "Quais os requisitos que tal 'prótese' pretende alcançar?"

Dando ênfase à abordagem de Zanshin, que combina conceitos de GORE (Goal Oriented Requiremets Engeneering) (Alexander e Beus-Dukic 2009), com a Teoria de Controle por Feedback (Filieri, et al. 2015) e Raciocínio Qualitativo, os autores fizeram um experimento para tentar responder a questão de pesquisa.

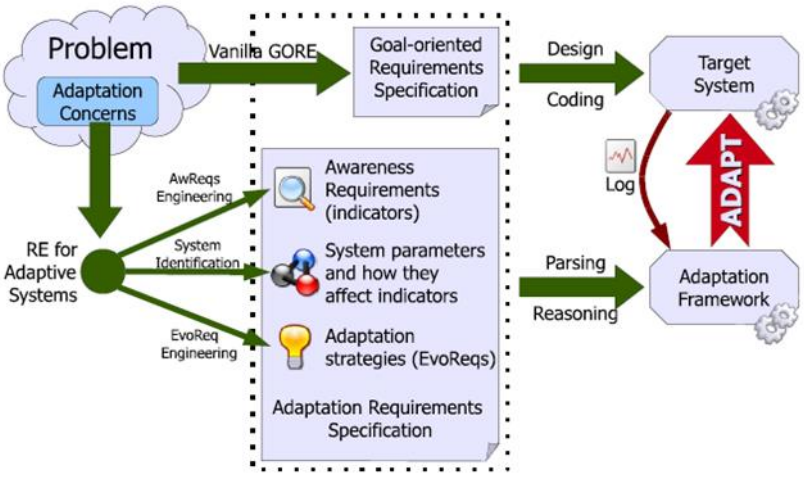

Figura 5 – Visão geral da Abordagem de Zanshin

Os autores aplicaram a aboradagem de Zanshin em um sistema adaptativo de ambulâncias assistido por computador, apresentando os resultados de simulações de cenários em tempo de execução.

A abordagem de Zanshin enxerga a consciência de software como requisitos sobre os estados do sistema, por exemplo: o sucesso/falha de suas tarefas em tempo de execução. A abordagem produz um modelo aumentado, a partir de um modelo orientado a metas.

4.1.2 Motivação para a Inclusão do Estudo de Caso

Por se tratar de um sistema que tem consciência do que ocorre à sua volta e que se adapta conforme as mudanças no ambiente ou nos requisitos de maneira que continue a cumprir com o seu propósito, o pesquisador sênior escolheu este artigo para analisarmos. A análise cumpriu dois objetivos: (1) orientar os alunos a identificar os requisitos de consciência, segundo o SIG de Souza Cunha e (2) esclarecer como as operacionalizações dos requisitos de consciência foram realizadas. Foram identificados 12 requisitos de consciência, em relação ao SIG de Souza Cunha, e, além disso, foram feitas 3 simulações: Adaptação através da evolução, Adaptação através da reconfiguração e testes de escalabilidade.

Podemos observar, por exemplo, o AR8 (Awareness Requirement) "DA MDTs communicate position should not be false more than once per minute" onde é exibida claramente a consciência de contexto.

4.1.3 Operacionalizações

A consciência de software como apresentada por Souza Cunha encontra-se em dois ramos: Auto Comportamento e Contexto.

4.1.3.1 Consciência do Auto Comportamento

Referente ao ramo de Consciência do Auto comportamento, o sistema realiza operacionalizações de quatro naturezas: Consciência de Funcionalidade; Consciência de Alternativas, Consciência de Objetivos e Monitor de Cumprimento de Metas.

Na consciência do auto comportamento, a operacionalização da consciência de alternativas, refinada pela avaliação das alternativas sob o impacto do contexto é alcançada pelos componentes OR e VP (Pontos de Variação), pelos Requisitos de Consciência de Zanshin (Areq – Zanshin, Ex.: As ambulâncias chegam em 8 minutos, os incidentes são resolvidos em 15 minutos, o despacho de uma ambulância ocorrem em 3 minutos, etc.) e pela Abordagem de Zanshin, respondendo à pergunta: Quais são os impactos da escolha de cada alternativa?

A operacionalização das consciência de metas, dá-se pela monitoração do status das ambulâncias, pela abordagem de Zanshin e pelo Vanilla Gore de Zanshin, respondendo às perguntas: Quais metas o software deve satisfazer, Como alocar as metas (e as metas flexíveis) no software, Como a influência do contexto pode ser modelada e Como a satisfação das metas é influenciada pelo contexto.

Os Requisitos de Consciência de Zanshin ajudam na operacionalização de Consciência Funcional – Requisitos em Tempo de Execução, enquanto que o Controle de Qualidade (Monitoração do status das ambulâncias, Logs e Mensagens geradas pelo próprio

sistema: Mensagens de monitoração e de exceção) proveem o feedback de qualidade do sistema, ajudando na operacionalização do refinamento Monitoria do Alcance das Meta. O conjunto ajuda a responder as perguntas: O software está atingindo as suas metas e Quão eficiente o software está sendo.

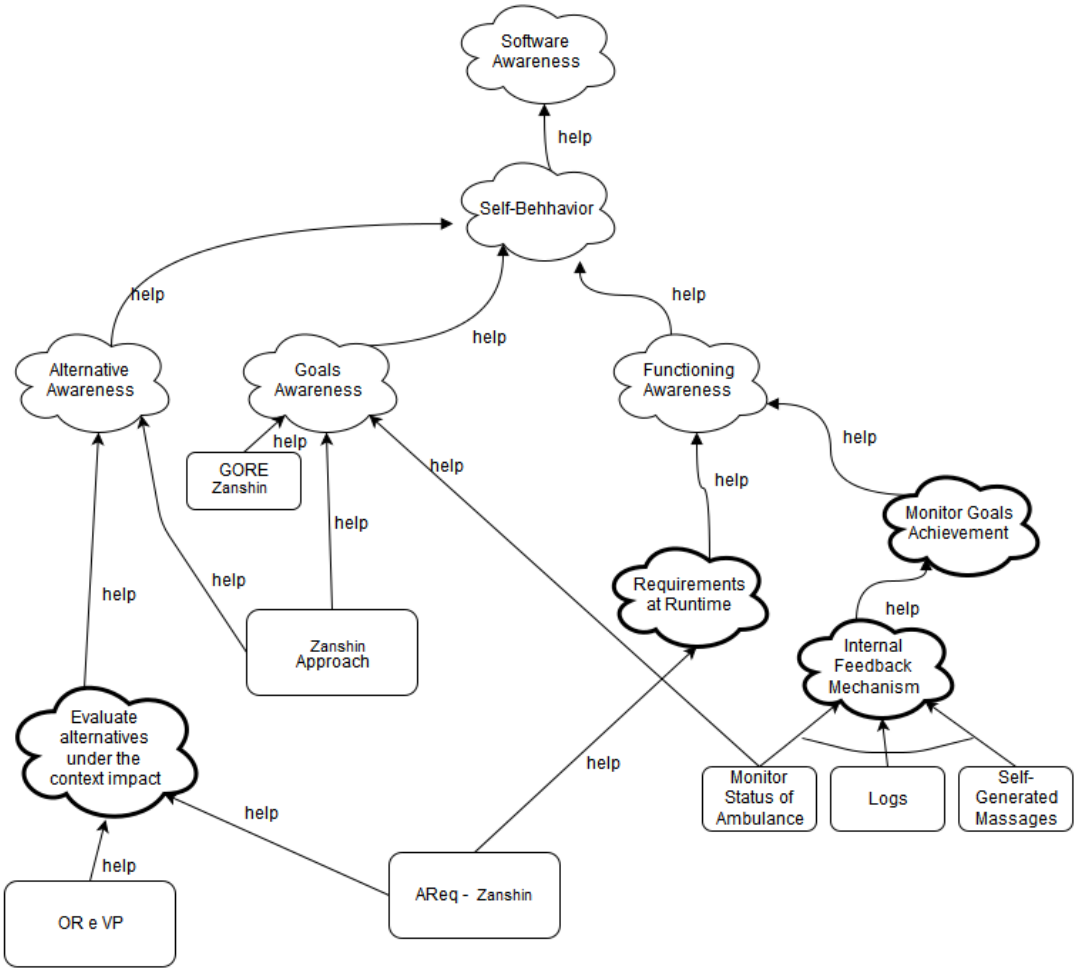

Figura 6 – Consciência do Auto Comportamento [A-CAD]

4.1.3.2 Consciência de Contexto

Referente ao ramo de Consciência de Contexto, o sistema realiza operacionalizações no ambiente físico, identificando ou monitorando a posição geográfica do incidente (via identificação da chamada), das ambulâncias, da configuração das ambulâncias, se as ambulâncias estão com algum problema mecânico ou aptas e a posição geográfica da ambulância para que a decisão de qual ambulância deve ser enviada para cada incidente seja a melhor possível no momento despacho da ambulância.

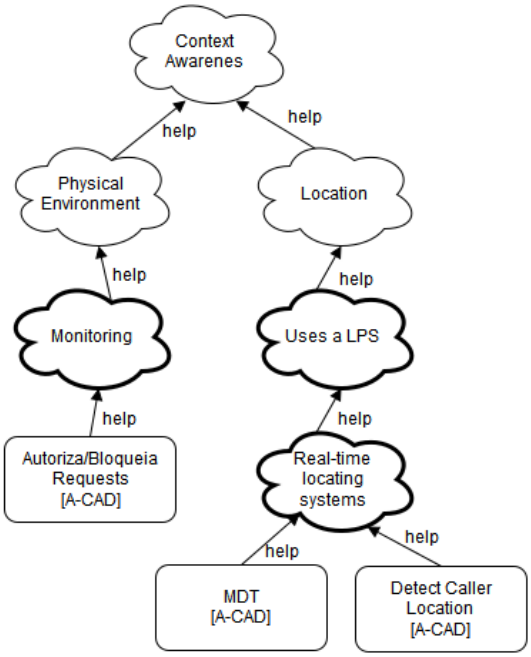

Figura 7 – Consciência de Contexto [A-CAD]

4.1.3.3 Consciência de Tempo

Estão modeladas as metas flexíveis REQ-12 (Ambulâncias chegam ao local do incidente em 8 minutos em 75% dos casos), REQ-16 (Ambulâncias são despachadas em até 3 minutos) e REQ-17 (incidentes resolvidos em no máximo 15 minutos), refinadas pelas metas de consciência de Zanshin AR-3, AR-4, AR-7 e AR-14, onde AR-3 e AR-4 refinam REQ-12, AR-7 refina REQ-17 e AR-14 refina REQ-16.

Esse conjunto de metas flexíveis e metas de consciência, em conjunto com as mensagens de exceção geradas quando tais metas falham, claramente exprimem o aspecto de Consciência de Tempo. O "timeout" de 3, 8 e 15 minutos diz respeito a um intervalo de tempo. Entretanto AR-4 especifica que o percentual de sucesso dos 8 minutos não pode cair por dois meses seguidos, que para ser medido necessita da Consciência de Tempo Linear.

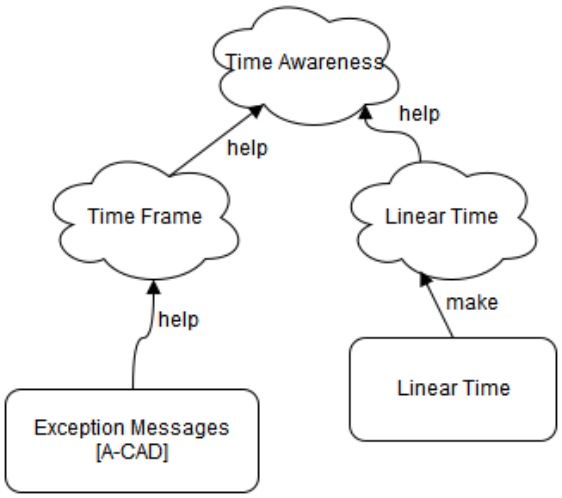

Figura 8 – Consciência de Tempo [A-CAD]

4.1.4 Conclusões

O monitoramento de requisitos é determinante para responder a questão "Quantas faltas podem ser toleradas respeitando cada requisito do sistema?" Além disso, o desenho de mecanismos de adaptação requer a identificação de um espaço de controles alternativos que podem alterar o comportamento do sistema. Sem uma definição clara de tal espaço, o desenho do componente de adaptação do sistema futuro seria incompleta.

4.2 Environmental awareness intrusion detection and prevention system toward reducing false positives and false negatives

4.2.1 Introdução do Estudo de Caso

O artigo publicado por (Sourour, et al., 2009) propõe uma nova forma de implementar softwares IDS/IPS (Intrusion Detection System/Intrusion Prevention System) com o objetivo de reduzir falsos positivos (o sistema detectou uma ameaça equivocadamente) ou falsos negativos (o sistema deixou de detectar uma ameaça real).

A proposta contempla uma abordagem híbrida, onde os Host Agents monitoram informações do host, tais como: sistema operacional, usuário autenticado, ações (leitura, escrita ou execução de arquivos), Endereços IP e MAC, processos com portas abertas e processos que utilizam recursos de rede, transmitindo estas informações ao Network Agent (Dispositivo Firewall ou Roteador de Borda). Desta forma, o Network Agent passa a ter a consciência da topologia da rede e os Host Agents passam a ter a consciência da política de usuários, uma vez que a recebem do Network Agent.

Vários padrões são utilizados para o monitoramento e comunicação entre os agentes. O protocolo SNMP (Simple Network Management Protocol) é utilizado para a monitoria dos hosts e a comunicação entre cada Host Agent com o Network Agent utiliza a KQML (Knowledge Query and Manipulation Language) como ACL (Agent Communication Language).

A política de segurança é armazenada em forma de grafos (Security Policy Graph).

Os eventos de segurança são descritos em IDMEF (Intrusion Detection Message Exchange Format), uma forma de XML aplicada à detecção de intrusões.

A arquitetura é dividida em três componentes principais: O Detector, o Analyser e o Controller.

O Detector coleta dados (de hosts ou da rede), identificam o tipo de tráfego e decidem se o tráfego será direcionado para o Analyser ou se o tráfego pode seguir diretamente ao seu destino.

O Analyser analisa o tráfego encaminhado pelo Detector criando os Analysers Events (ξ) composto dos atributos (State, Action, Actor, Target, Time e Info). O atributo de estado contém 0 para tráfego normal ou 1 para tráfego malicioso ou intrusivo. Tal análise visa a redução de falsos negativos. O atributo de ação tenta identificar se a ação que o tráfego deseja realizar é permitida ou não pela política de segurança. Ator e Alvo são, respectivamente, o objeto origem e o objeto destino. O atributo tempo contém o momento em que o evento ξ foi gerado. Já o atributo informação contém a assinatura de detecção a classe do ataque ou outra informação que ajude na identificação da ameaça.

O Controller toma a decisão final se o tráfego será liberado ou bloqueado. O estado do evento ξ determina como o Controller fará a sua avaliação final.

Para todo estado de ξ normal, o Controller confronta o tráfego com o conjunto de regras da política de segurança P. Caso o tráfego não esteja de acordo com a política de segurança, o tráfego é bloqueado. Esta medida também reduz a possibilidade de falsos negativos.

Para todo estado de ξ intrusivo, o Controller confronta o tráfego com a política de segurança P e com a Cartografia da Rede C. A Cartografia da rede C é composta pela Topologia da rede T e do conjunto de aplicações e softwares S que estão em funcionamento no Host destino (Target). O Controller pode chegar a duas conclusões:

i) o tráfego é nocivo ao destino e uma notificação é necessária.

ii) o destino é imune ao tráfego, ex: um atacante está tentando valer-se de uma vulnerabilidade particular do sistema operacional Linux em um host que o Controller sabe que o sistema operacional é Windows). Neste caso, trata-se de um falso positivo que foi evitado. O ataque pode ser adicionado ao log, mas uma notificação não é necessária.

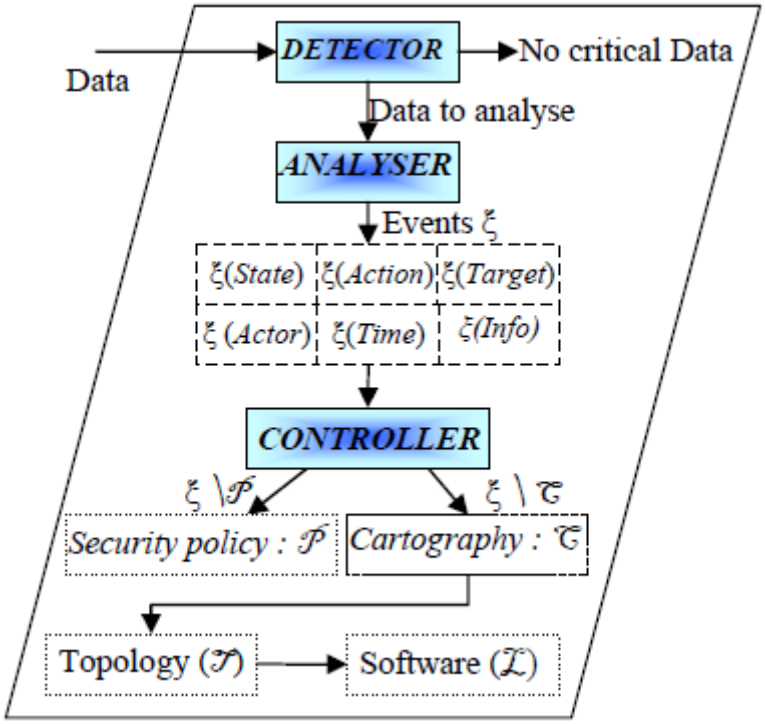

Figura 9 – IDS/IPS baseado em Políticas e Cartografia

4.2.2 Motivação para Inclusão do Estudo de Caso

Este artigo postula que - pelo fato de o sistema obter consciência do ambiente (política de segurança, topologia de rede e softwares em execução nos hosts) - ele será capaz de melhor analisar os pacotes trafegados na rede e, desta forma, será capaz de realizar uma melhor classificação do pacote (benigno ou maligno) e se uma notificação ao administrador da rede é necessária. As estatísticas mostradas na *seção 7 (tabela 1)* do artigo demonstram uma melhora significativa na qualidade da detecção dos pacotes, reduzindo tanto os falsos positivos quanto os falsos negativos.

4.2.3 Operacionalizações

A proposta contida no artigo realiza as seguintes operacionalizações:

- Consciência de Contexto
- Consciência de Tempo
- Consciência do Contexto Social

4.2.3.1 Consciência de Contexto

A operacionalização da consciência de contexto dá-se no ambiente de computacional. As operacionalizações respondem às perguntas:

- Quais recursos estão disponíveis?
- Como as informações são modeladas?
- Como as informações são trocadas entre diferentes atores?

KQML, por ser uma forma de XML, ajuda na operacionalização dos Padrões de Troca de Informações e também, por ser uma ontologia, ajuda na operacionalização de Modelos de Informações e Recursos.

O protocolo SNMP (Simple Network Management Protocol), da mesma forma, é um padrão de troca de informações, ajudando na operacionalização de Padrões de Troca de Informações.

Os eventos ξ do Analyser constituem uma ontologia para análise de tráfego de rede e, desta forma, ajudam na operacionalização de Modelos de Informações e Recursos.

A operacionalização de Modelos de Informações e Recursos é alcançada pelo softgoal Ontologias, via KQML, e através da arquitetura do IPS/IDS de Sourour et al., por intermédio de um ou mais dos seus componentes: Detector, Analyser e Controller.

O Detector usa as Regras de Identificação para avaliar o tráfego, sendo que estas regras conhecidas como Access Control Lists (ACL) constituem um padrão de modelagem de informações para Firewalls e, portanto, também ajuda na operacionalização de Padrões de Troca de Informações.

O Analyser produz os eventos ξ no formato IDMEF, que por ser uma forma de XML, também ajuda na operacionalização de Padrões de Troca de Informações.

O Controller, por sua vez, vale-se do Security Policy Graph para tomar a decisão final sobre a liberação ou bloqueio do tráfego suspeito. O Security Policy Graph contém as informações da rede e dos hosts, portanto, dos recursos disponíveis. Destarte, ele também ajuda na operacionalização de Padrões de Troca de Informações.

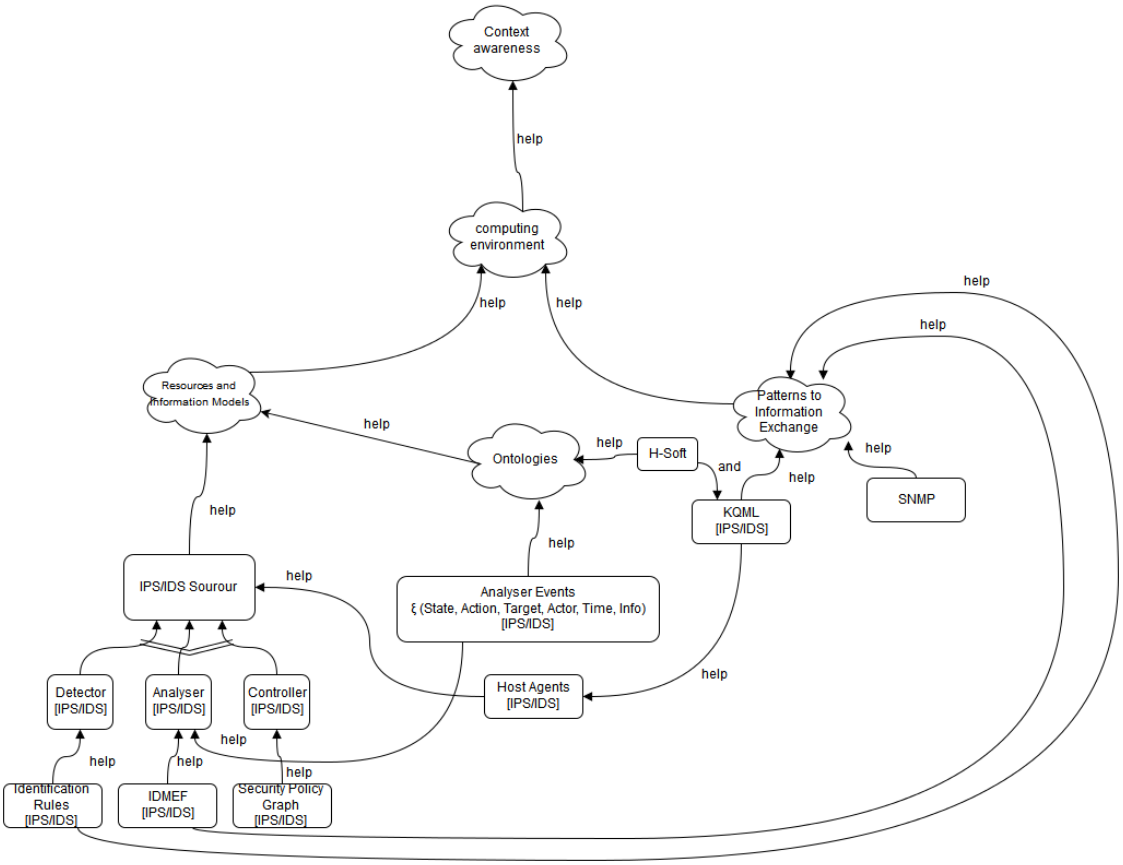

Figura 10 - Consciência de Contexto [IPS/IDS]

4.2.3.2 Consciência de Tempo

De acordo com Cunha, se o software precisa ter consciência do instante em que ele está operando ou se é relevante saber o instante com alto grau de acurácia, então existe a consciência de tempo. Apesar de o artigo de Sourour, et al. não mencionar se o tempo medido nos Host Agents ou Network Agent estão sincronizados com servidores NTP, podemos assumir que os relógios dos referidos agentes têm precisão suficiente para gerar eventos ξ fidedignos em relação ao tempo. Os eventos ξ do Analyser ajudam na operacionalização Intervalo de Tempo da consciência de tempo, assim como os relógios internos dos agentes e da arquitetura do IPS/IDS de Sourour, et al. ajudam na realização da operacionalização de Tempo Linear da consciência de tempo.

É importante ressaltar que Time Frame e Linear Time são extensões da consciência de tempo, inicialmente definida por Cunha em sua tese.

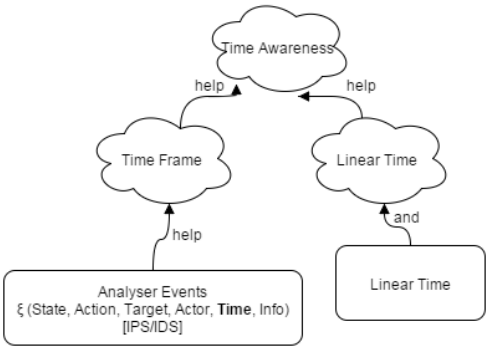

Figura 11 – Consciência de Tempo [IPS/IDS]

4.2.3.3 Consciência do Contexto Social

A operacionalização da consciência do contexto social ocorre na consciência de usuários. As operacionalizações respondem às perguntas:

- Quem são os usuários?
- Como os usuários podem ser identificados?

Os eventos ξ do Analyser ajudam na operacionalização Identificação dos Usuários.

O artigo não deixa claro como o usuário foi autenticado, entretanto, os Host Agents têm condições de determinar quem é o usuário autenticado no host, não importando o método que ele tenha utilizado para se autenticar no host.

Quando o usuário a ser identificado no evento ξ (atributo Actor) é um usuário externo, não importa qual o método usado para a autenticação (HTTP, HTTPS, Kerberos, OAUTH ou outro), um Token de sessão normalmente é criado (Vipin, et al., 2008). Desta forma, através de um Token de sessão, podemos identificar o usuário associado a ele.

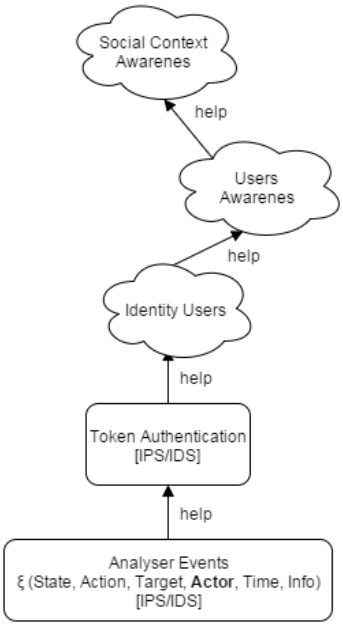

Figura 12 – Consciência do Contexto Social [IPS/IDS]

4.2.4 Conclusões

Ao adquirir conhecimento em tempo de execução, no caso a consciência do ambiente no qual o software está inserido, o mesmo melhorou seu desempenho em relação às implementações "inconscientes" do mesmo tipo de software. A consciência permitiu que o software tomasse melhores decisões na classificação dos pacotes trafegados e diminuiu significativamente o percentual de falsos positivos e falsos negativos

4.3 Profiling. Memory in Lua.

4.3.1 Introdução do Estudo de Caso

A dissertação (Musa e Ierusalimschy, 2015) apresenta duas bibliotecas para monitorar o uso de memória em um programa Lua, chamadas de luamemprofiler e lmprof. A ferramenta luamemprofiler é automática, usa a técnica de allocation-hook e apresenta relatório em tempo real. Lmprof por sua vez é uma ferramenta também automática, porém além de allocation-hook possui também call-hook e só é capaz de gerar um relatório após o término do programa. Luamemprofiler tem como objetivo quantificar as alocações de memória para cada um dos tipos de dados da linguagem, enquanto lmprof busca analisar como o comportamento das funções em termos de consumo de memória.

4.3.2 Motivação para a Inclusão do Estudo de Caso

A princípio a motivação para inclusão dessa dissertação era mostrar como as bibliotecas possuiriam consciência de software. No entanto, após a leitura da dissertação e debate entre os participantes, ficou claro que as bibliotecas não se enquadravam nos critérios que definem se um artefato possui consciência de software.

Surgiu então uma nova motivação para a inclusão dessa dissertação: embora as ferramentas propostas não apresentem características de consciência, consideramos a possibilidade dessas ferramentas serem utilizadas como uma forma de obter consciência de contexto, mais especificamente consciência do ambiente computacional. Considerando as fases de desenvolvimento e teste de aplicações, o desenvolvedor de uma aplicação Lua pode fazer uso delas para entender o comportamento de seu programa e, a partir das informações coletadas, buscar formas de otimizá-lo.

4.3.3 Operacionalizações

Como as ferramentas fornecem dados sobre o uso de memória, adicionamos ao SIG de Cunha um refinamento de Consciência do Ambiente Computacional: Consciência de Uso de Memória. Hierarquicamente abaixo deste refinamento, inserimos dois outros refinamentos denominados de Consciência de Uso de Memória Principal e Consciência de Uso de Memória por uma Função.

4.3.3.1 Consciência de Contexto – Luamemprofiler

Luamemprofiler: A biblioteca luamemprofiler operacionaliza a Consciência de Uso de Memória Principal, uma vez que esta monitora o uso de memória de um trecho do programa, mostra os dados coletados em um visualizador ao longo da execução e apresenta um relatório ao fim de sua execução.

Essa biblioteca pode ser carregada por meio da função require, uma função da biblioteca padrão de Lua que carrega e executa uma biblioteca, e, portanto ajuda na operacionalização dessa biblioteca.

Uma vez carregada a biblioteca luamemprofiler, as funções start e stop definidas por essa biblioteca podem ser utilizadas pela delimitar os trechos do programa cujo consumo de memória desejamos monitorar. Desse modo, essas funções também auxiliam a operacionalização da biblioteca. A função start recebe um parâmetro inteiro, chamado no artigo de heap-size-display, que define a porção de memória cujo uso de deseja visualizar. Se esse parâmetro não for passado, não haverá visualização do consumo de memória, somente será produzido o relatório ao final da execução. Logo, esse parâmetro ajuda a operacionalizar tanto a função start como o componente de display gráfico.

As informações sobre os espaços alocados e quais tipos de dados correspondem a cada região de memória estão armazenadas em um heap, que é inicializado na função start, fazendo com que o heap seja uma alternativa de operacionalização não somente da biblioteca, mas também da função start.

Os dados desse heap podem ser visualizados através do componente de display gráfico (Graphic Displayer), que, portanto, também contribui positivamente para a operacionalização da biblioteca. Esse display gráfico só está disponível quando um heap-size-display é passado, pois este não é capaz de auto adaptar o tamanho do display em função do consumo pontual de memória percebido pela biblioteca. Para que isso fosse feito, seria necessário que a biblioteca tivesse consciência de auto comportamento, conforme consta em suas considerações finais.

Outra forma de visualização de informações sobre uso de memória coletada com a biblioteca luamemprofiler é através do relatório final, em que são impressas a quantidade de memória alocada (proveniente de uma chamada de alloc), de memória realocada (realloc) e de memória liberada (free), assim como o número de vezes que essas operações foram realizadas. Também são impressos a quantidade de alocações de memória para cada tipo de dado de Lua (string, função, userdata, thread, tabela e outros) e o tamanho máximo de uso de memória atingido durante a execução do programa. Além dessas informações todas, consta também no relatório uma sugestão de valor para o heap-size-display, que pode ser passado para a função start da biblioteca em uma próxima execução para melhorar a visualização do Heap. Por esse motivo o relatório (Report) não só ajuda na operacionalização da biblioteca, mas também na operacionalização do heap-size-display.

Entrando, em um nível menos abstrato, podemos citar operacionalizações referentes à implementação da biblioteca: a função lua_setallocf, da API C de Lua, ajuda a operacionalizar a biblioteca luamemprofiler e a função customizada de alocação (allocation function), uma vez que permite que da nova função de alocação seja registrada durante a execução do programa Lua.

Em Lua, há somente uma função para fazer alocação e liberação de memória, cuja implementação padrão utiliza as funções alloc, realloc e free da biblioteca padrão de C [Ier13]. Conforme a descrição apresentada na dissertação, a implementação da nova função de alocação é um wrapper em torno da função de alocação default para coleta de metadados. Logo, essas funções ajudam a operacionalizar a função default de alocação de memória de Lua (l_alloc), que, por sua vez, auxilia a operacionalização da função de alocação customizada, junto com os metadados coletados sobre a alocação.

Dessa forma, toda vez que um espaço de memória é alocado ou liberado por módulos Lua é interceptada de forma a armazenar os metadados referentes àquele espaço. Entretanto é importante ressaltar que módulos C chamados a partir de programas Lua podem alocar e liberar memória de forma independente, de modo que não é possível monitorar a alocação e liberação de memória desses módulos.

A função de alocação (allocation function) contribui positivamente para a operacionalização da biblioteca luamemprofiler.

Por fim, podemos dizer que as operacionalizações respondem às seguintes perguntas propostas por Cunha para identificar consciência do ambiente computacional:

- Como a informação é modelada?
- Como a informação é armazenada?

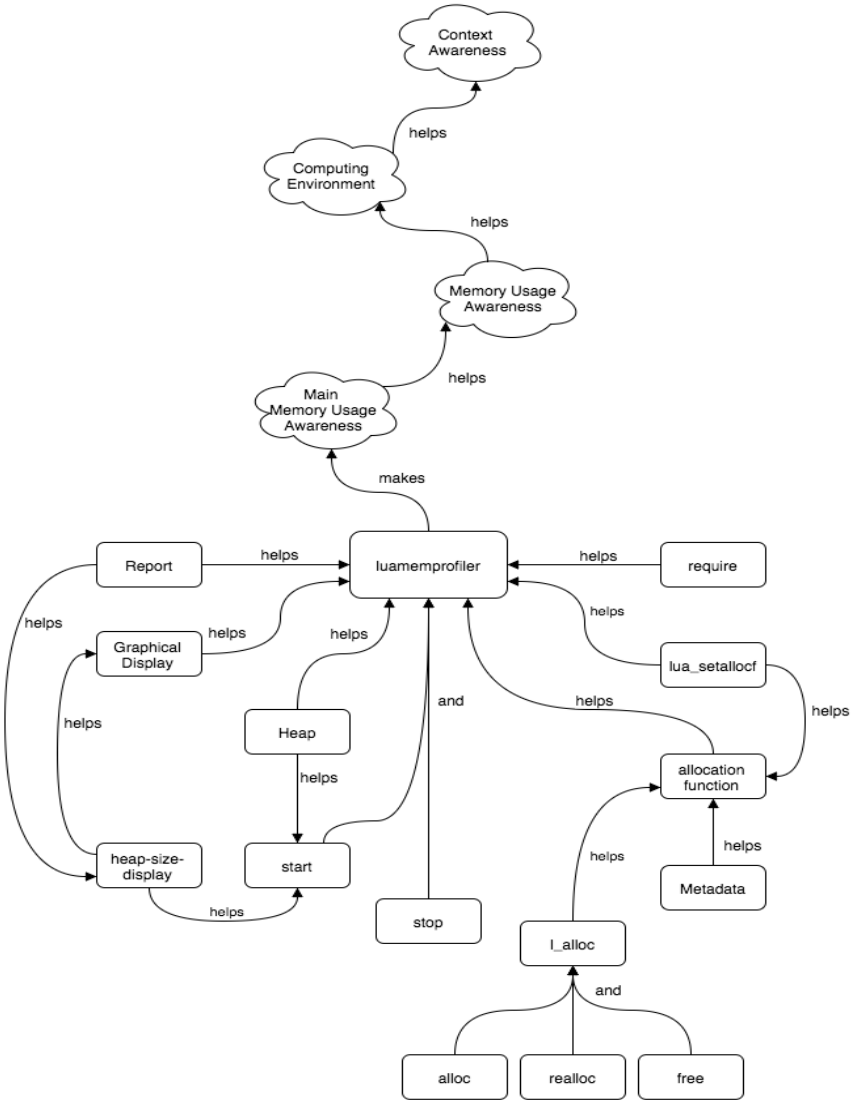

Figura 13 – Consciência de Contexto [luamemprofiler]

4.3.3.2 Consciência de Contexto de lmprof

A biblioteca lmprof operacionaliza a Consciência de Uso de Memória por Funções, gerando relatórios que mostram o comportamento de funções em relação a alocação e liberação de memória.

Assim como a biblioteca anterior, deve ser carregada dinamicamente através da função require para disponibilizar para o programa Lua as funções start e stop dessa biblioteca. De novo, assim como luamemprofiler, require ajuda a operacionalizar a biblioteca lmprof.

Carregada a biblioteca lmprof, as funções start e stop podem ser utilizadas pela delimitar os trechos do programa cujo consumo de memória por função desejamos monitorar. Dessa forma, essas funções também auxiliam a operacionalização da biblioteca.

A função start utiliza a função lua_sethook para definir um hook que intercepta as chamadas de função. Logo lua_sethook ajuda a operacionalizar não só a biblioteca mas também a função start e o próprio hook. Além disso, o contador de chamadas da função corrente (call counter), o tamanho da memória retida (retained memory size) e o tamanho da memória ocupada (shallow memory size), que são atualizados a cada chamada de função, auxiliam a operacionalização do hook.

Lmprof, assim como luamemprofiler, utiliza a função lua_setallocf, junto com uma função de alocação (allocation function) para implementar o allocation-hook necessário para monitorar a alocação de memória.

Já a função stop salva os metadados (Metadata) em um arquivo no formato de uma tabela Lua (Lua table). Logo tanto os metadados quanto o tipo de dados tabela de Lua ajudam a operacionalizar a função stop. A tabela lua, por sua vez auxilia a operacionalização dos metadados.

Existem dois perfis de geração de relatório: Flat Profile e Call Graph Profile. Esses dois perfis ajudam a operacionalizar a biblioteca lmprof. Ambos perfis são operacionalizados pelos metadados (Metadata), utilizados na geração dos relatórios desses perfis. O Call Graph Profile também é operacionalizado pelo grafo de chamadas (Call Graph), que modela o comportamento do programa em relação as chamadas de funções e recursões.

Por fim, podemos dizer que as operacionalizações respondem às seguintes perguntas propostas por Cunha para identificar consciência do ambiente computacional:

- Como a informação é modelada?
- Como a informação é armazenada?
- Como a informação pode ser recuperada?

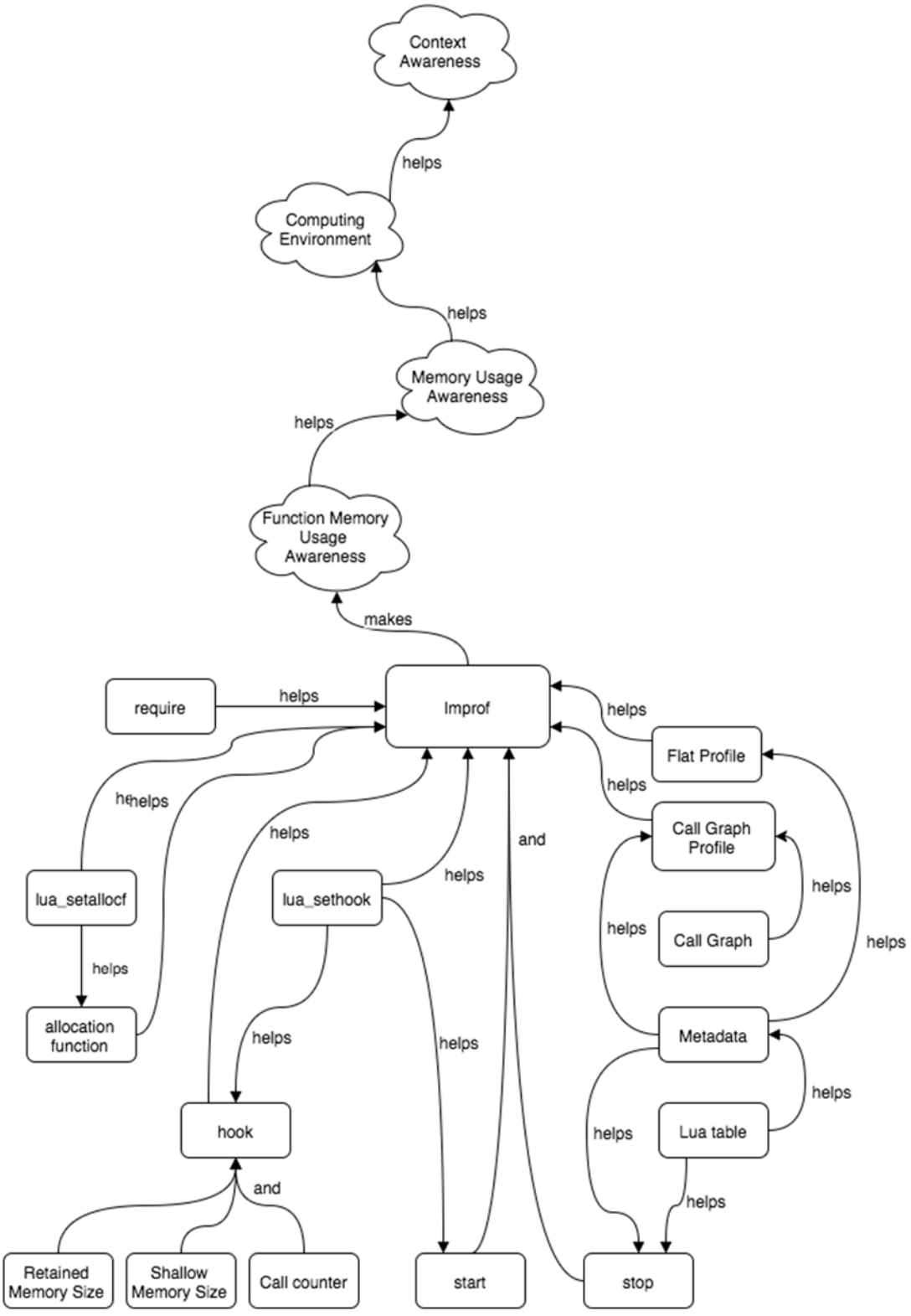

Figura 14 – Consciência de Contexto [lmprof]

4.3.4 Conclusões

É evidente que as bibliotecas luamemprofiler e lmprof não possuem por si só consciência de software, porém podemos inferir que elas podem ser ferramentas para operacionalizar consciência de uso de memória no desenvolvimento de software

4.4 Automated support for diagnosis and repair

4.4.1 Introdução do Estudo de Caso

O artigo (Alrajeh, et al., 2015) disserta acerca da um framework que tem como objetivo a sinergia entre a verificação de modelo (Model Checking) e o aprendizado de máquina (Machine Leraning) para automatizar o ciclo de verificação, diagnóstico e reparo, que os engenheiros de software comumente utilizam para o desenvolvimento de sistemas complexos, reduzindo o esforço humano e potencialmente produzindo um produto mais robusto.

Os autores argumentam que a verificação de modelo realiza uma busca exaustiva de violações de propriedades nas descrições formais, tais como: código, requisitos, especificações de projeto, assim como as configurações de rede e de infraestrutura, produzindo contraexemplos quando tais propriedades são violadas. Todavia, mesmo sendo eficaz para a detecção de falhas, a verificação de modelos provê apenas suporte limitado para que se compreenda as causas do problema, quanto mais sua resolução.

Sobre aprendizado de máquina, os autores argumentam que os reparos sugeridos reparam as falhas encontradas sem a introdução de novas falhas, visto que os algoritmos de aprendizagem baseados em lógica usam os exemplos corretos e os contraexemplos que geraram violações para estender e modificar uma descrição formal. A nova descrição formal permanece em conformidade com os exemplos ao mesmo tempo que evita os contraexemplos.

O framework foi exemplificado tomando como exemplo o sistema de despacho de ambulâncias de Londres, os seguintes passos são analisados:

- espera-se um incidente a ser resolvido;
- após análise da gravidade do incidente é solicitada a intervenção de uma ambulância;
- a ambulância, para resolver o incidente, encaminha o paciente para admissão no hospital mais próximo do local;
- o hospital deve ter todos os recursos para tratar o paciente;
- o objetivo é o desempenho sem falhas da tarefa de admissão do paciente dentro dos parâmetros de tempo, previamente destacados em (4.1.3.3

Não foi considerado o caso em que o hospital mais próximo carece de recursos suficientes (como um leito vago), um problema não identificados na análise original.

Verificando-se a descrição formal original do domínio em relação ao objetivo declarado no modelo, gerando-se automaticamente cenários e exemplificando os casos baseados em lógica de autoaprendizagem, revisa-se a descrição formal de acordo com este cenário, substituindo o modelo original por um com o objetivo de admitir o paciente no hospital mais próximo que possua os recursos disponíveis. A união da verificação de modelo e aprendizagem baseada na lógica, portanto, fornece suporte automatizado para especificação, verificação, diagnóstico e reparação, reduzindo o esforço humano, desta forma produzindo um produto mais robusto. O restante deste artigo explora um quadro geral para a integração de verificação de modelo e aprendizagem baseada na lógica, conforme o framework apresentado na figura abaixo.

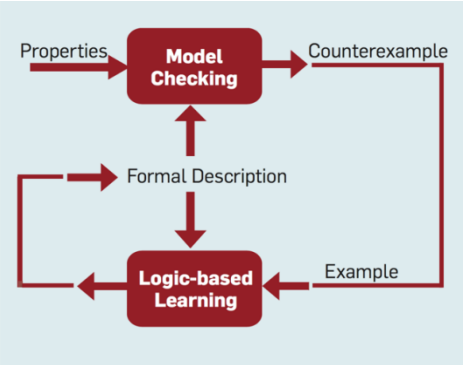

Figura 15 – Framework genérico para verificação, diagnóstico e reparação de modelo.

4.4.2 Motivação para Inclusão do Estudo de Caso

O trabalho foi selecionado principalmente porque usa uma outra abordagem para resolver o mesmo problema descrito no estudo de caso 4.1 Uma vez que este artigo não faz referências a softwares auto adaptativos ou conscientes, podemos comparar os resultados e verificar se ambas as soluções apresentam os mesmos aspectos de consciência de software.

A seção 4.4.3 detalha os aspectos de consciência (operacionalizações) que foram identificados neste trabalho, tendo como base o SIG de Software Consciente proposto por Souza Cunha.

4.4.3 Operacionalizações

Após a análise do trabalho identificamos que este apresenta operacionalizações de Consciência Funcional e Consciência de Alternativas, ambas no aspecto de Consciência do Auto Comportamento. Nesta seção, são apresentadas as operacionalizações identificadas e as características que contribuíram para tal resultado.

A Consciência Funcional é percebida a partir do modelo de ações e execuções o software pode perceber seu próprio comportamento e possíveis falhas em sua execução (conjunto de ações que não atingem o objetivo desejado), construindo contraexemplos ao modelo original. Nesta etapa é possível mensurar o quão efetivo é o software, possibilitando inclusive a execução de simulações e analisando os casos de sucesso e insucesso.

Sobre a Consciência de Alternativas, o framework proposto responde as perguntas: como implementar as alternativas, qual o impacto de escolher cada alternativa e como avaliar a existência de alternativas de acordo com o contexto corrente.

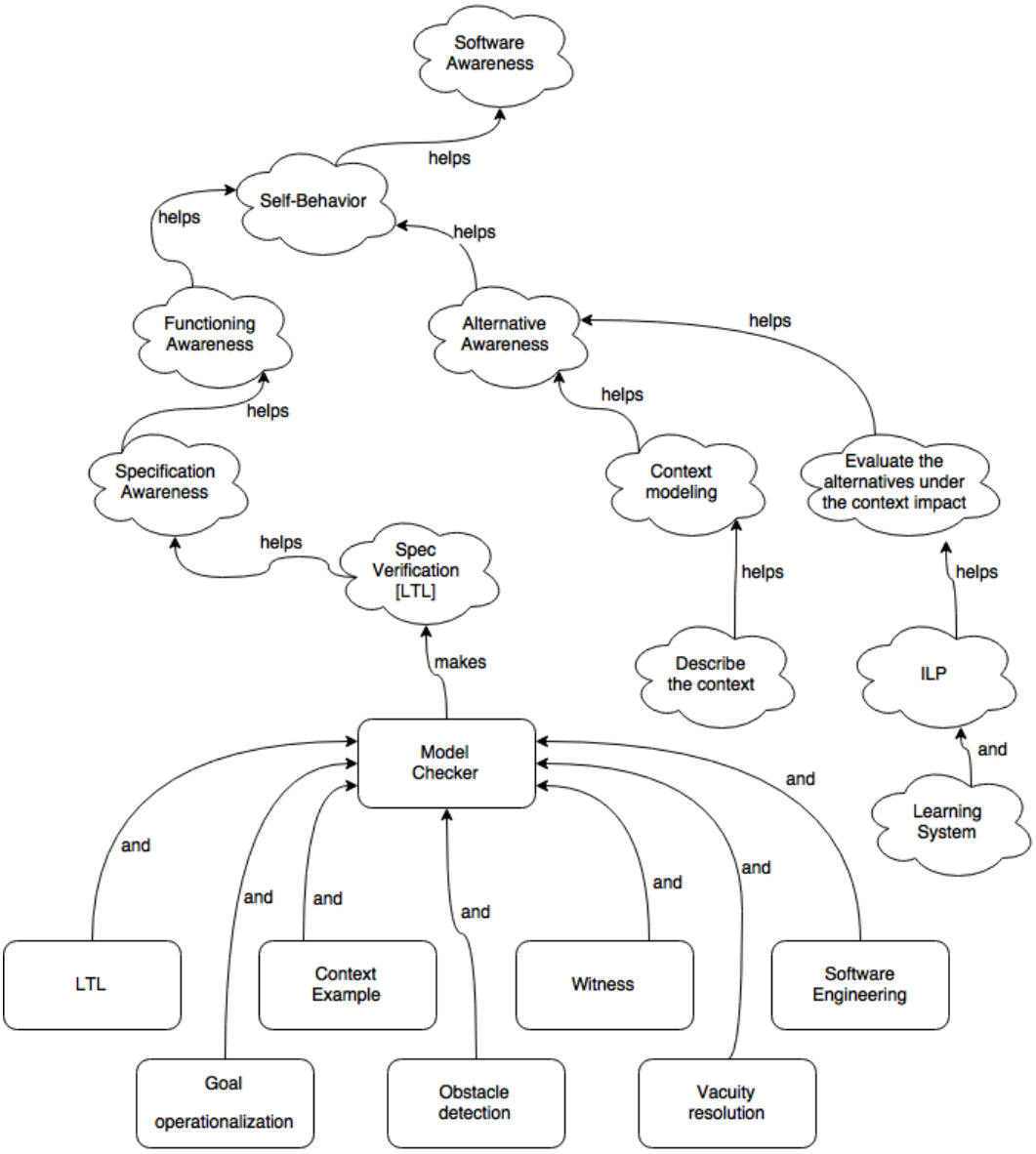

Figura 16 – Consciência do Auto Comportamento [Automate support for diagnosis and repair]

O framework proposto implementa a Consciência de Alternativas em 4 passos:

Model checking: O objetivo deste passo é verificar a descrição formal de violações de propriedades. Como resultado temos um exemplo de sucesso ou um contraexemplo. No caso de um contraexemplo (violação de regra), o processo de verificar-diagnosticar-reparar continua para o passo seguinte.

Elicitation: O contraexemplo produzido pela etapa de verificação de modelo não é uma expressão exaustiva de todas as formas que uma propriedade P1 pode ser violada; outras situações também poderia levar a uma violação de P1 e também de uma outra propriedade P2. Este passo dá aos engenheiros de software a oportunidade de apresentar contraexemplos adicionais possivelmente relacionados. Pode-se também encontrar descrições e propriedades incompatíveis; ou seja, descrições que sempre violam alguma propriedade. Os engenheiros de software podem, portanto, fornecer tracers (chamados de "testemunhas") que exemplificam como a propriedade deve ser satisfeita. Tais exemplos podem ser ativados manualmente pelo engenheiro de software ou gerados automaticamente através de uma análise automatizada.

Logic-based learning: . Tendo contraexemplos identificados e tracers testemunhas, o software de aprendizagem baseado na lógica executa automaticamente o processo de reparo. O objetivo na aprendizagem é calcular modificações apropriadas à descrição formal de tal forma que o contraexemplo detectado seja removido, assegurando aos tracers testemunhas a eliminação do contraexemplo sob a designação alterada.

Selection: No caso em que as alterações algoritmo de aprendizagem baseado em LBL (logic-based learning) encontra alternativas para a mesma tarefa de reparação, um mecanismo de seleção é necessária para decidir qual usar. A seleção é dependente do domínio e requer a entrada de um especialista de domínio humano. Quando uma seleção é feita, a descrição formal é atualizado automaticamente.

Na figura abaixo é apresentado um exemplo de um controle de portas de um trem, neste é encontrado um contraexemplo onde o trem está em movimento com as portas abertas, violando uma regra (passo 1). Um engenheiro de software pode confirmar se a especificação e propriedades são consistentes, recolhendo automaticamente um tracer-testemunha que mostra como P1 pode ser satisfeita, mantendo as portas fechadas enquanto o trem está se movendo e abri-las quando o trem está parado (passo 2). Tendo como exemplo negativo as portas que se abrem quando o trem está se movendo, e exemplo positivo portas que se abrem quando ele parou, o objetivo da tarefa de reparo é fortalecer o pré e pós-condições das operações do controlador de trem para impedir que as portas do trem se abram quando for indesejável. O algoritmo de aprendizagem verifica que a pré-condição da operação de portas abertas não é suficientemente restritiva e, consequentemente, calcula uma pré-condição reforçada exigindo o trem estar parado para que as portas se abram e que as portas estejam fechadas para que o trem comece a andar (passo 3). A alternativa reforçada de pré-condição "portas estão fechadas, o trem não está acelerando" é sugerido pelo software de aprendizagem, neste caso em que os especialistas do domínio poderia optar por substituir a definição original da operação abrir portas.

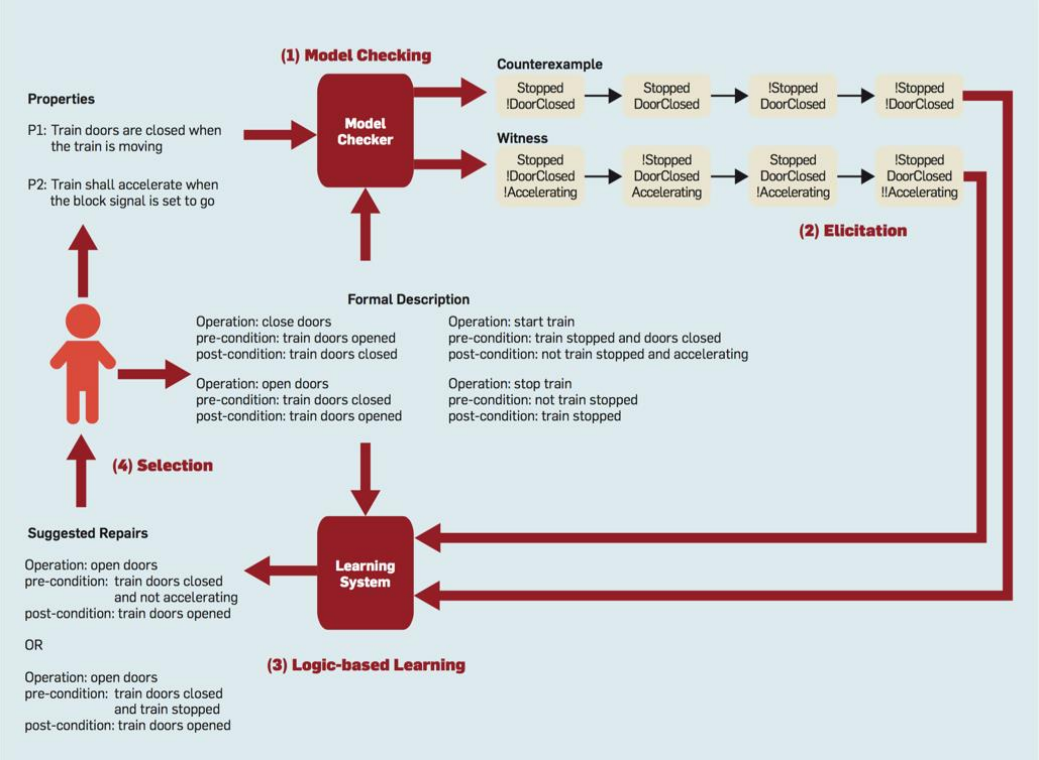

Figura 17 – Exemplo Controlador de Trem

4.4.4 Conclusões

Pelas operacionalizações da seção 4.4.3, pudemos verificar que o framework apresenta consciência do Auto Comportamento, como o artigo descrito em 4.1 sendo que o artigo desta seção não apresenta consciência de metas. O artigo não discorre acerca que como as descrições formais seriam monitoradas em tempo de execução, partindo do princípio que tal monitoração já faz parte da ferramenta de verificação de modelos.

4.5 MORPH: A Reference Architecture for Configuration and Behaviour Self-Adaptation

4.5.1 Introdução do Estudo de Caso

Este trabalho apresenta o framework arquitetural de referência MORPH (Braberman, et al., 2015) para auto adaptação de configuração e comportamento de sistemas. Como um framework arquitetural, o MORPH representa um conjunto de elementos que podem ser implementados em um sistema visando à sua auto adaptação em tempo de execução. A ênfase da auto adaptação, no contexto deste trabalho , é na mudança de configuração e atualização de comportamento.

As mudanças arquiteturais necessárias em tempo de execução podem ocorrer somente em nível de configuração (ex.: componentes, conexões, parâmetros, etc.), somente em nível de comportamento (ex.: orquestração dos componentes) ou até mesmo em ambos os níveis. Por compreender estas motivações, Braberman et. al. propuseram um framework arquitetural que visa permitir que ambos os tipos de modificações em tempo de execução.

A Figura 18 apresenta a arquitetura de referência MORPH. Nesta arquitetura são propostas três camadas principais: Goal Management, Strategy Management e Strategy Enactment. Para apoiar estas três camadas, é proposto também um repositório comum de conhecimento. Além destas camadas, a figura mostra o sistema alvo e a infraestrutura de log. De um modo geral, cada elemento da arquitetura tem suas responsabilidades a saber:

- Goal Management é responsável por reagir a mudanças no modelo de metas ou à necessidade de reconfiguração, propondo novas estratégias que sejam possíveis de serem executadas pelas camadas subsequentes. Para atender a este objetivo, esta camada possui três elementos: Goal Model Manager, Reconfiguration Problem Solver e o Behaviour Problem Solver. O Goal Manager define se a necessidade de mudança é relativa à reconfiguração, comportamento ou ambos e solicita estratégias aos solucionadores de problema específicos. Estas estratégias são passadas à camada Strategy Management. Os fluxos de comunicação entre estas camadas e elementos internos também podem ser visualizados na Figura 18.

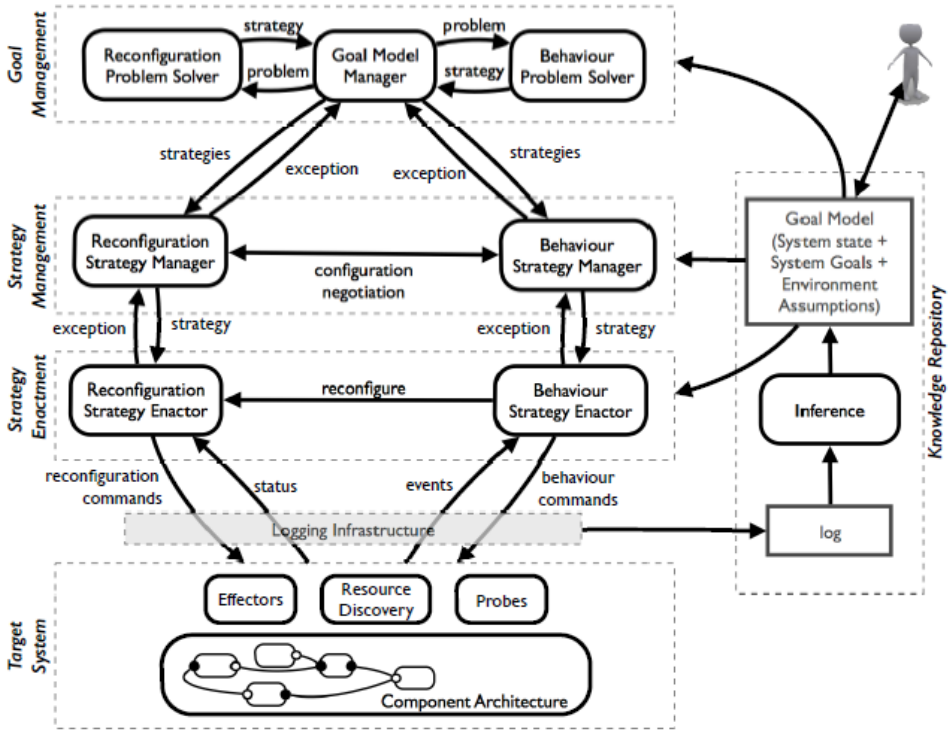

Figura 18 – A arquitetura de referência MORPH

Strategy Management é responsável por fazer adaptações para mudanças de acordo com estratégias pré-processadas e no repositório de conhecimento comum. Quando uma mudança necessária não possui uma estratégia adequada disponível, esta camada gera uma exceção para a camada acima, Goal Management, para que sejam geradas novas estratégias de acordo com a nova necessidade de mudança. Para atingir seus objetivos, esta camada é composta por dois elementos: reconfiguration Strategy Management e o Behaviour Strategy Manager. Cada um dos elementos desta camada repassam estratégias para serem executadas camada subseqüente e negociam configurações quando as mudanças são dos dois tipos (i.e.: configuração e comportamento) e devem ocorrer de modo orquestrado, como apresentado na Figura 18.

Strategy Enactment é responsável por executar a estratégia selecionada pela camada Strategy Managment. Para realizar esta execução esta camada utiliza dois elementos: Reconfiguration Strategy Enactor e Behaviour Strategy Enactor. Como mostra a Figura 18, entre os elementos desta camada, só há comunicação do Behaviour Strategy Enactor para o Reconfiguration Strategy Enactor, pois Braberman et al. entendem que algumas mudanças de comportamento podem exigir alguma reconfiguração em um determinado momento. Cada elemento desta camada depende dos elementos do sistema alvo para concretizar as ações das estratégias.

- Knowledge Repository é responsável por armazenar o modelo de metas (estado do sistema, metas e conhecimentos acerca do ambiente) e um log com os dados de execução do sistema alvo. Os usuários, stakeholders e administradores também podem contribuir para este repositório comum de conhecimento. É importante ressaltar que este repositório é utilizado pelas três camadas apresentadas anteriormente e que, por estar sendo proposto em uma arquitetura de referência, não é determinado o formato para armazenamento dos dados no repositório.

- Sistema Alvo (Target System) é responsável pela implementação dos componentes do sistema propriamente dito e pela implementação dos elementos que realmente executam as ações solicitadas pela camada Strategy Enactor (i.e.: Effector, Resource Discovery e Probes).

4.5.2 Motivação para Inclusão do Estudo de Caso

Diante da introdução ao artigo, é possível identificar que o trabalho de Braberman et. al. foi selecionado por propor uma arquitetura de referência que visa tornar um sistema alvo um software auto adaptativo. Devido ao fato de não propor uma implementação específica, o trabalho descreve os elementos arquiteturais e o raciocínio de cada um destes elementos a fim de que possa ser implementada de acordo com a tecnologia do sistema alvo.

Neste contexto, a arquitetura de referência é uma solução de operacionalização de consciência reutilizável em diferentes sistemas. A seção 4.5.3 detalha os tipos de consciência que foram identificados neste trabalho a partir da leitura do trabalho de Braberman et. al e tendo como base o SIG de Consciência proposto por Souza Cunha

4.5.3 Operacionalizações

4.5.3.1 Consciência de Contexto

A operacionalização da consciência de contexto foi identificada devido ao fato de que o sistema alvo implementa elementos como: Effectors (responsáveis pela execução de comandos de comportamento e reconfiguração) e Componentes de Arquitetura (executam as funcionalidades específicas do sistema). Estes elementos são acionados de acordo com a consciência de que eles existem no ambiente computacional.

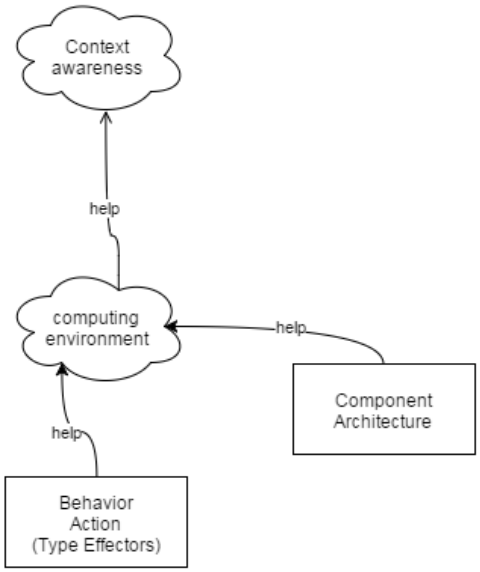

Figura 19 – Consciência do Contexto [MORPH]

4.5.3.2 Consciência do Auto Comportamento

Os componentes Goal Model Manager, suportado pelos componentes Reconfiguration Problem Solver e Behavior Problem Solver realizam a operacionalização da Consciência

de Metas, respondendo as perguntas: Qual metas o software deve alcançar e Como colocar as metas e metas flexíveis no software.

Os componentes Strategic Management e Strategic Enactor, suportados por seus subcomponentes e pelo Goal Model Manager realizam a operacionalização da Consciência de Alternativas, respondendo as perguntas: Quais são as alternativas possíveis para alcançar as metas ou as metas flexíveis e Onde colocar as alternativas no software. Com a ajuda do Knowledge Repository, o Strategic Management pode avaliar o impacto das escolhas das alternativas, baseando-se no seu aprendizado, o que ajuda a responder as perguntas: Qual o impacto de escolher uma determinada alternativa e Como avaliar as alternativas existentes de acordo com o contexto atual.

Quanto a Consciência de Funções, esta é realizada pelos componentes Logging Infrastructure e Target System, este suportado pelos seus subcomponentes e aquele ajudado pelo Knowledge Repository. O conjunto responde as perguntas: Como o software percebe seu próprio comportamento, O software está atingindo suas metas e Quão eficiente o software está sendo.

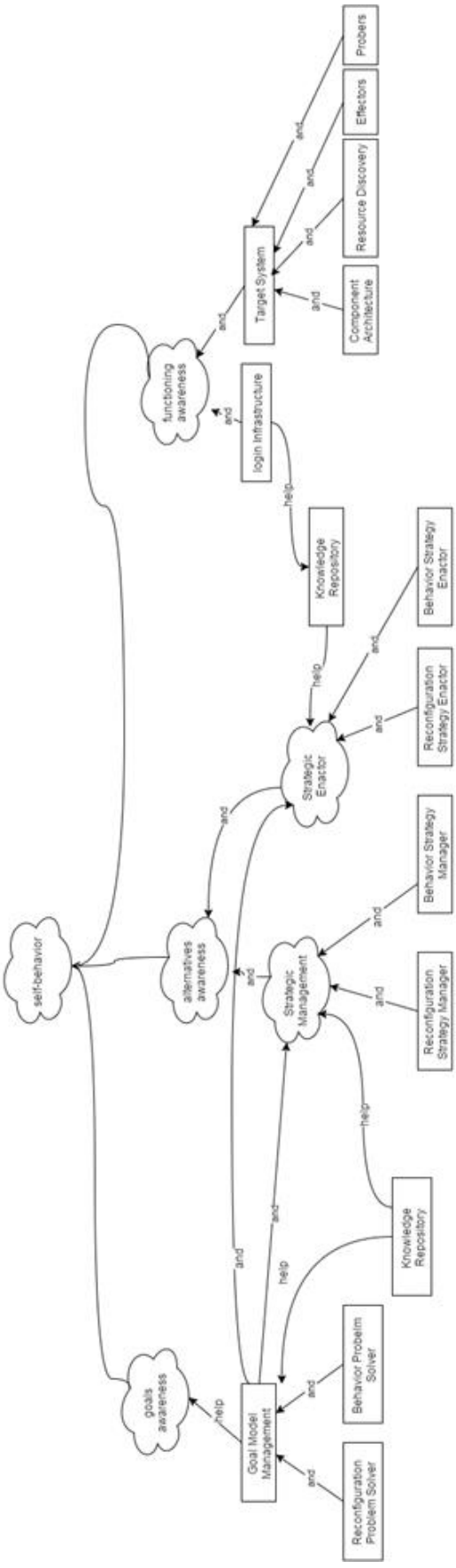

Figura 20 – Consciência do Auto Comportamento [MORPH]

4.5.3.3 Consciência do Contexto Social

A operacionalização da consciência de contexto social foi identificada devido ao fato de que os usuários, administradores e stakeholders podem contribuir com o repositório comum de conhecimento. Deste modo, é possível introduzir nas adaptações do sistema as preferências do usuário e também possível conhecer algumas ações realizadas pelos usuários através da infraestrutura de log que retroalimenta a repositório comum de conhecimento.

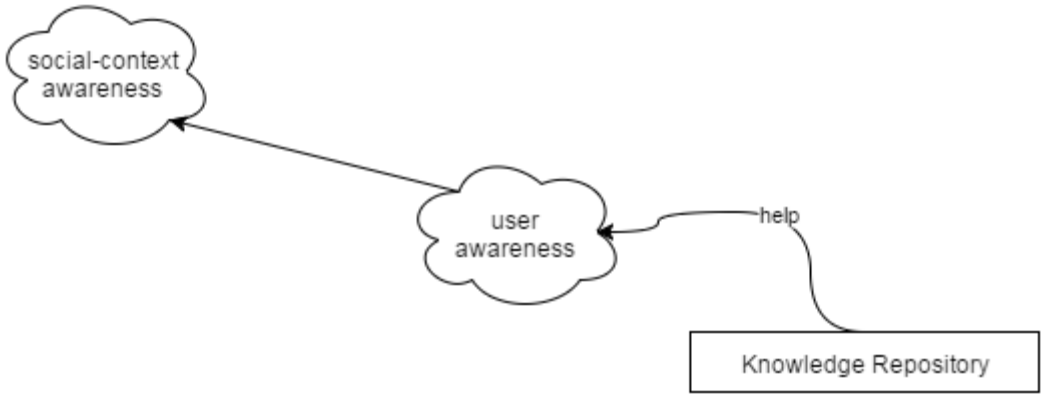

Figura 21 – Consciência de Contexto Social [MORPH]

4.5.4 Conclusões

Diante da proposta deste trabalho e após a leitura do trabalho do Braberman et. al, foi possível observar as operacionalizações de consciência apresentadas na seção 4.5.3 Além disso, o MORPH tem um diferencial de ser uma proposta de arquitetura de referência, o que torna as operacionalizações reutilizáveis em outros sistemas auto adaptativos baseados no MORPH.

4.6 UCCA: A Unified Cooperative Control Architecture for UAV missions

4.6.1 Introdução do Estudo de Caso

O trabalho em foco (Tian, et al., 2012) apresenta a arquitetura UCCA, uma arquitetura de controle cooperativa para UAV (Veículos Aéreos Não Tripulados, ou flying robots ou usualmente, drones). O UCCA é uma arquitetura que possibilita o suporte para operações com múltiplos UAV e possui diversas características tais quais: é modularizada; permite o controle na troca de informações internamente e externamente ao UAV; suporta a execução de missões com múltiplos objetivos; facilita a cooperação de módulos internos de diferentes funções; suporta o emprego de UAV em missões de ambiente e contextos incertos. A figura a seguir apresenta o UCCA de maneira resumida e intuitiva. Maiores detalhes podem ser identificados no trabalho na íntegra.

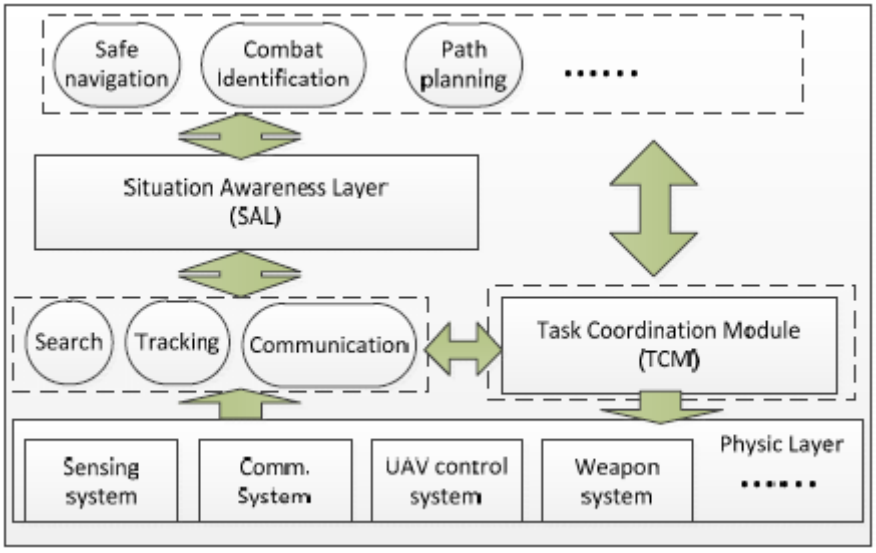

Figura 22 – Componentes Principais do UCCA

A Figura 22 apresentou os componentes principais do UCCA. As funções na camada inferior pertencem ao Physical Layer, no qual há acoplagem de equipamentos ao sistema (como sensores, atuadores e etc). Os dois conjuntos de funções que estão dentro de retângulos pontilhados que estão junto à margem esquerda são chamados de Functional Layers. O retângulo inferior é chamado de Functional Layer Low devido ao fato de que agrega as funções de análise e controle de baixo nível, ligadas ao Physical Layer.

O retângulo superior é chamado de Functional Layer High devido ao fato de lidar apenas com a base de conhecimento e o TCM e é responsável por planejamentos e os objetivos. O TCM é o componente responsável pelas execuções e priorizações dos comandos oriundos da Functional Layer High, os executando diretamente na Physical Layer mediante avaliações informações no SAL. O SAL é o componente responsável pela base de conhecimento do UCCA.

Realizando um breve de/para com o modelo MAPE-K (3 podemos observar que as funções de Mapeamento são realizadas pelo Functional Layer Low acessando a Physical Layer. A Análise e Planejamentos são atribuições do Functional Layer High. A Execução é organizada pelo TCM e o Conhecimento (K) é gerido pela SAL.

4.6.2 Motivação para Inclusão do Estudo de Caso

É razoável e intuitivo alegar que há esforços para que robôs ajam, no máximo possível, de maneira autônoma ou até mesmo inteligente. Inteligentes no sentido de, dentro do domínio de suas atividades autônomas, possam efetuar escolhas que tragam os maiores benefícios aos seus objetivos.

Nos esforços nessa linha de raciocínio, as operacionalizações utilizadas nas tarefas dos UAV implementam diversas funcionalidades que contribuem para o nível de consciência do software de comando e controle dos UAV. Por esses motivos esse trabalho foi escolhido para ser estudando quanto as suas operacionalizações que ajudam no SIG de consciência de software de Souza Cunha.

O artigo em estudo é muito breve sobre implementações, aplicações, ou mesmo possíveis casos de uso baseados em sua proposta. Para exercitarmos o uso de seus recursos foi necessário inferir comportamentos que consideramos básicos e esperados de UAV em missões autônomas.

4.6.3 Operacionalizações

O UCCA apresenta operacionalizações em todos os ramos de Consciência de Software proposto por Souza Cunha.

4.6.3.1 Consciência do Auto Comportamento

O UCCA apresenta operacionalizações de Consciência de Funcionalizardes, Consciência de Alternativas e Consciência de Metas

Consciência de Funcionalidades: A Figura 23 apresenta como três componentes do UCCA podem contribuir para a consciência de software de um UAV. Todos componentes do Functional Layer High do UCCA podem ajudar a consciência de um UAV, no entanto para que isso ocorra é mandatória a utilização do componente SAL. O SAL por sua vez, pode ter tido informações a partir do Objective Driven Learning (ODL) framework que é utilizado para implementar o SAL.

Um exemplo de tal operacionalização seria o SAL sendo utilizado por uma funcionalidade da camada Functional Layer High como, por exemplo, a Safe Navigation para verificar se uma possível nova rota passa por algum ponto de perigo anteriormente encontrado

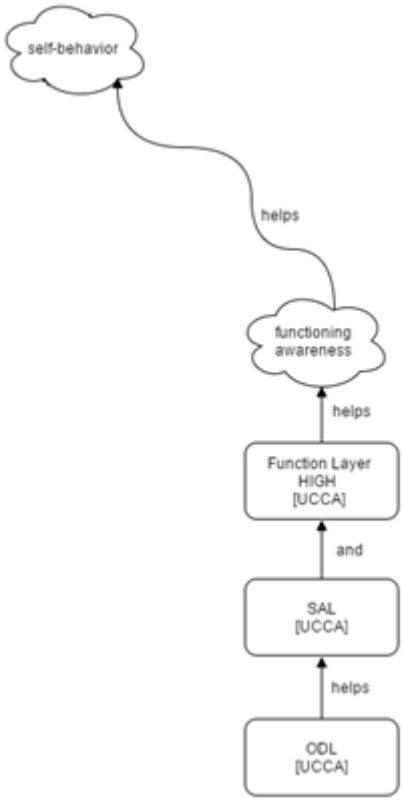

Figura 23 – Consciência Funcional [UCAA]

Consciência de Alternativas: A Figura 24 apresenta como três componentes do UCCA podem contribuir para a consciência de alternativas de um UAV. Funções da Functional Layer High como, por exemplo, a Path Planning poderiam gerar mais de uma rota entre dois pontos de interesse. A TCM por sua vez, em posse dessas duas alternativas poderia escolher a mais adequada. Isto poderia ocorrer com ajuda da ODL.

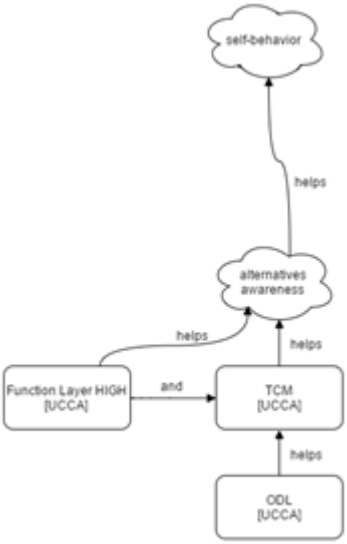

Figura 24 – Consciência de Alternativas [UCAA]

Consciência de Metas: A Figura 25 apresenta como dois componentes do UCCA podem contribuir para a consciência de metas de um UAV. Considerando que "UAV Goals" sejam os objetivos que um UAV possua, o Functional Layer High seria capaz de alterá-lo ou complementá-lo a partir de informações do SAL. Um exemplo de tal operacionalização seria manter o consumo de combustível/hora abaixo de um determinado threshold e isso poderia ser verificado através do SAL.

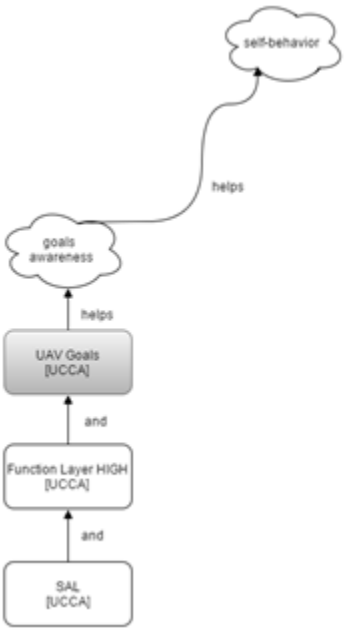

Figura 25 – Consciência de Metas [UCAA]

4.6.3.2 Consciência Social (Social-Context Awareness)

Outro ramo de divisão da consciência de software como apresentada por Souza Cunha é do ramo Social-Context Awareness. Ela pode receber operacionalizações de três naturezas: Consciência de Normas; Consciência de Usuário e Consciência de Social de Relacionamento. Neste caso o UCCA apresenta operacionalizações somente quando a

Consciência de Social e de Relacionamento, mais especificamente no módulo responsável pelo relacionamento com outros UAV.

A Consciência de Social e de Relacionamento pode ser auxiliada pelo Functional Layer Low do UCCA. Porém, para que isso aconteça, as funções do Functional Layer Low necessariamente farão uso do componente SAL diretamente. Nesta operacionalização o SAL pode, ou não, fazer uso da Functional Layer Low, que se utilizada, fatalmente recorre ao componente TCM.

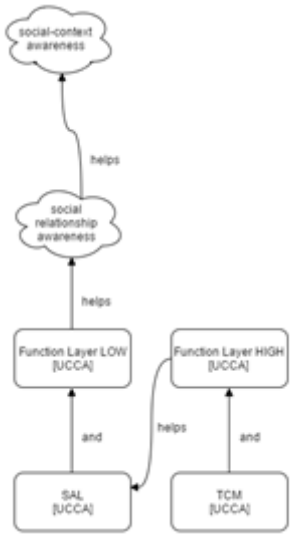

Figura 26 – 4.5.3.3 Consciência do Contexto Social [UCAA]

4.6.3.3 Consciência Temporal (Time Awareness)

A consciência temporal pode receber ajuda das operacionalizações do UCCA através componentes da Functional Layer Low (especificamente do tracking) que faria o papel intermediário com os sensores do UAV.

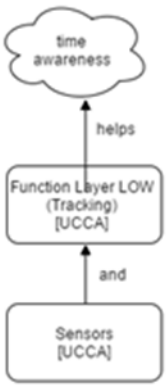

Figura 27 – Consciência Temporal [UCAA]

4.6.3.4 Consciência do Contexto (Context Awareness)

A consciência de localização de um UAV pode ser auxiliada de diversas maneiras através do componente SAL, que neste caso fatalmente faria o uso encadeado do Functional Layer Low para acesso aos sistemas de sensoriamento. Um exemplo prático seria a localização ser mantida atualizada no SAL pela leitura de GPS quando em ambientes externos ou através de sistemas de localização indoor quanto em ambientes encobertos.

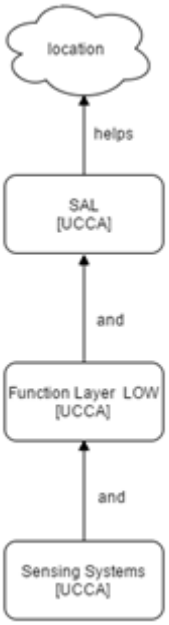

Figura 28 – Consciência de Contexto [UCAA]

4.6.3.5 Consciência de Ambiente Físico

A consciência de ambiente físico pode ser auxiliada por dois caminhos, o Monitoramento e a Comunicação com outros UAV. No caso de ser auxiliada pelo monitoramento, ela seria auxiliada pela camada de sensores prevista do SIG original de Herbert et al., que por sua vez, seriam lidos pelo Functional Layer Low da UCCA. No caso ser auxiliada pela Comunicação, ela seria fatalmente alimentada diretamente pela SAL. No caso do acesso direto a SAL, ela forneceria informações também informadas pelos outros dois componentes.

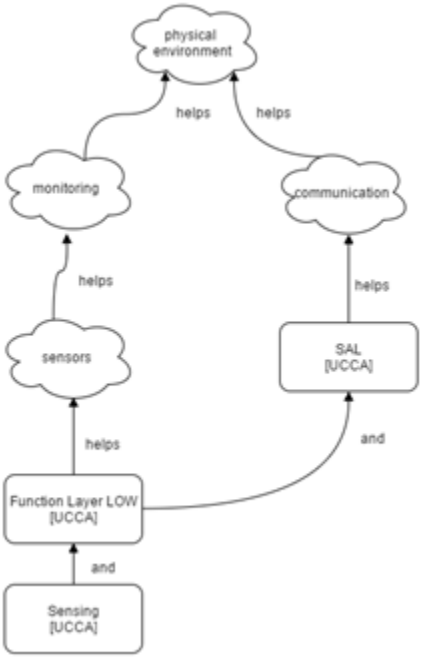

Figura 29 – Consciência de Ambiente Físico [UCAA]

4.6.3.6 Consciência do Ambiente Computacional

A consciência do ambiente computacional pode ser auxiliada de três formas distintas pelo UCCA, juntas, separadas ou mesmo duas a duas. A primeira é diretamente sendo auxiliada pelo SAL, que quando utilizada necessita do Function Layer Low fazendo uso do Communication System (pertencente da Physical Layer).

A segunda maneira é ser auxiliada pelo TCM que cascateia os requisitos anteriores e também precisa do Physical Layer diretamente e do "UAV Control System". A Figura 30 apresenta uma operacionalização nomeada de "UAV Control System", esse termo embora não seja original do artigo em estudo, foi utilizado no estudo como sendo o apanhado de todos os componentes necessários para que a TCM efetivamente envie uma tarefa já priorizada para ser executada.

A terceira maneira seria o auxílio pelo Function Layer High que pode ou não contar com o TCM e suas dependências.

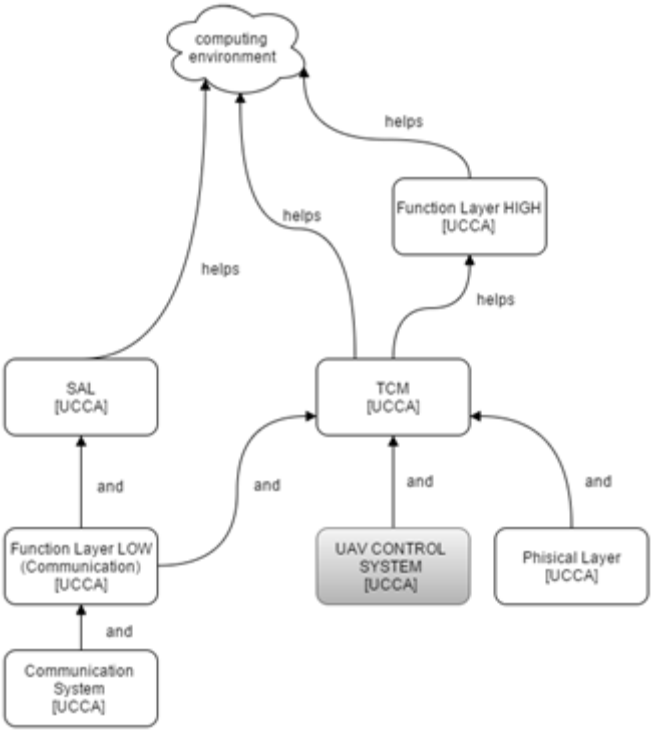

Figura 30 – Consciência do Ambiente Computacional [UCCA]

4.6.4 Conclusões

Softwares para robôs autônomos definitivamente fazem parte dos sistemas que proativamente buscam o aumento do seu nível de consciência para realizar suas tarefas. Softwares para controle de UAV, por sua vez, são um subconjunto desses sistemas que apresentam muitas operacionalizações que podem contribuir com a consciência de software de maneira mais ampla se replicadas em outros cenários.

Apesar de inicialmente não ter sido esperado, é natural que o UCAA apresente operacionalizações em todos os ramos do SIG proposto por Souza Cunha, visto que o TCM toma decisões sobre os estados e parâmetros contidos no SAL (Situation Awareness Layer). O termo Situational Awareness (Consciência Situacional) pode ser definido como a percepção dos elementos de um determinado volume de tempo e espaço, a compreensão do significado do que foi percebido e a projeção da situação do que foi percebido num

futuro próximo (Endsley, 1998). A definição básica foi estendida (Dominguez, et al., 1994) de forma que a Consciência Situacionbal deve incluir quatro elementos

- Extração de Informação do Ambiente
- Integração da informação extraída com o conhecimento interno relevante para criar uma imagem mental da situação corrente
- Uso desta imagem para dirigir a exploração perceptual adicional num ciclo perceptual contínuo
- Antecipar eventos futuros

De acordo com essas definições e com as operacionalizações do UCAA, podemos concluir que o estudo de caso do UCCA nos ajudou a modelar a Consciência Situacional no contexto de UAVs.

4.7 Predictive Monitoring of Business Processes

4.7.1 Introdução do Artigo

O artigo deste estudo de caso (Maggi, et al. 2014) propõe uma abordagem para analisar registros de eventos correspondentes a atividades (logs) de sistemas que suportam processos de negócio complexos, a fim de previamente monitorar os objetivos do negócio. Basicamente, é apresentado um framework que prevê violações de ações que não atingiriam um objetivo de negócio definido. Muito além da detecção de violações que aconteceram, quando uma atividade está sendo executada, o framework identifica se o valor de dado de entrada é mais (ou menos) aproximado a cada objetivo de negócio.

Baseado geração contínua de predições e recomendações de quais atividades devem ser executadas, e de quais valores de entrada de dados serão melhores, a fim de que violações de restrições dos objetivos de negócio sejam minimizadas. A qualquer momento o usuário pode especificar um objetivo de negócio utilizando a Lógica Temporal Linear (LTL), pois o framework continuamente oferece ao usuário estimativas da probabilidade de atingir cada objetivo de negócio para um determinado processo em execução.

O método principal de geração de predição é a parte central desse artigo. Mais especificamente, as técnicas de estimativa para cada atividade habilitada a ser executada em um caso, juntamente com cada valor de dado de entrada informada por essa atividade, terão uma probabilidade de desempenho para um objetivo de negócio. Ou seja, dado um processo em curso com algumas atividades habilitadas para serem executadas, o framework primeiro seleciona, do conjunto de rotas de execução, as atividades que tem um "prefixo similar" à rota do processo em curso (para coincidir o controle de fluxo). Em seguida, para cada rota selecionada, o framework produz um snapshot dos dados, atribuindo valor a cada atributo de dados até chegar ao prefixo similar à rota do processo em curso. Dado um objetivo de negócio, o framework classifica o snapshot dos dados como exemplo positivo ou negativo, baseado no objetivo que deve ser atingido na rota completa do processo em curso. Dessa forma, o framework mapeia as tarefas de predição em tarefas de classificação, onde o objetivo é determinar se o snapshot dos dados leva a atingir o objetivo de negócio e qual a probabilidade associada. Por fim, para definir o resultado da classificação, uma árvore de decisão é usada para estimar a probabilidade de que o objetivo de negócio será atingido para cada possível combinação de valores de atributos de entrada de dados.

Para realização dos testes, os autores do artigo assumiram ser verdade que:

- Estava disponível para o framework um conjunto de rotas de execução do passado à respeito do processo, a fim de que fossem extraídas informações sobre como o processo executou no passado. Dessa forma, o framework definiria as predições e recomendações para a rota em execução
- O processo de negócio é não-determinístico, ou pelo menos os mecanismos para tomar decisão durante a execução do processo não deveriam ser de conhecimento do usuário
- Os dados usados no processo são globalmente visíveis ao longo de todo o processo de execução

4.7.2 Motivação para Inclusão do Estudo de Caso

No framework proposto, Predictive Business Process Monitoring (PBPMF), identifica-se uma consciência sobre o auto comportamento do framework. Baseado no conhecimento das rotas passadas, o PBPMF é capaz de tomar decisões e sugeri-las para o usuário. Isso é possível pois há duas partes principais no PBPMF: Trace Processor que filtra e classifica as rotas de execuções passadas (similares a rota de execução corrente), e o Predictor Module que usa a saída do Trace Processor para gerar as predições e recomendações para o usuário final

4.7.3 Operacionalizações

A proposta contida no artigo utiliza a operacionalização de Consciência do Auto comportamento, pois identifica-se que algumas das perguntas padrão , apresentadas por Souza Cunha e relevantes para identificar esse tipo de consciência, são respondidas quando aplicadas a requisitos do framework em questão.

É importante destacar que, aplicado a esse artigo, a operacionalização de Consciência de Auto comportamento pode ser refinada em Modelagem de Contexto, e mais ainda em Definição de um Mecanismo de Identificação, uma vez que funções probabilísticas são usadas pelo framework para gerar planos de ação que serão apresentados para o usuário final. Além dessas operacionalizações, usando uma estratégia de analisar os requisitos do framework em tempo de execução e a forma que o framework monitora a satisfação das metas, observou-se que a operacionalização de Consciência do Auto comportamento pode ser refinada também por utilização de um mecanismo de feedback interno.

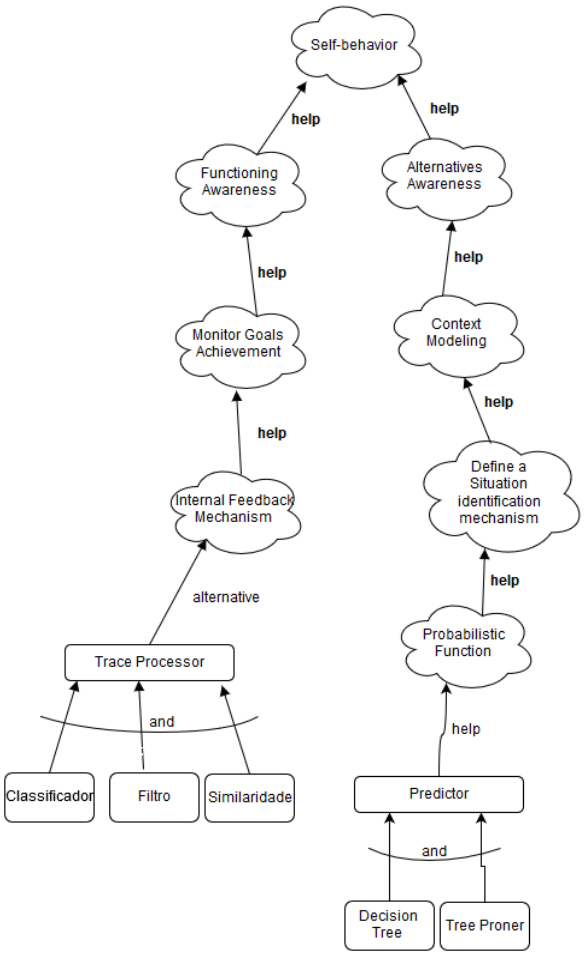

Figura 31 – Consciência do Auto Comportamento [PBPMF]

A operacionalização do Auto Comportamento responde às seguintes perguntas:

- Quais são as possíveis alternativas que atingirão os objetivos?

Baseado na sequencia de atividades que serão executadas em um plano de ação, e nos valores de dados atribuídos após cada execução de uma atividade do plano de ação, o framework (mais especificamente o Predictor Module) propõe e prevê, ao usuário, alternativas possíveis de atividades que minimizarão as violações de restrições dos objetivos de negócio.

- Qual o impacto de escolher cada uma das alternativas?

Depende do objetivo de negócio que deve ser atingido. Dado que uma alternativa é sugerida pelo framework de tal forma que minimize as violações de restrição dos objetivos de negócio, descartar uma alternativa sugerida é o mesmo que descartar um plano de ação viável, que atingiria o objetivo de negócio proposto pelo usuário ao framework, ou seja, descartar a possibilidade de alcançar o objetivo proposto.

- Como avaliar as alternativas atuais de acordo com o contexto corrente?

Analisando as rotas de atividades em tempo de execução, e usando as rotas de execução de processos passados (mais especificamente o Trace Processor), o framework (mais especificamente o Predictor Module) continuamente presenta ao usuário estimativas probabilísticas relacionadas a atingir cada objetivo de negócio segundo o plano de ação de atividades traçado para a rota de execução em curso.

4.7.4 Conclusões

Destacar que o framework PBPMF apresenta requisitos de Consciência de Auto Comportamento é relevante, pois agrega valor ao projeto proposto (PBPMF) e acrescenta conhecimento ao mapa de consciência apresentado por Souza Cunha. Novas operacionalizações, Trade Processor e Predictor Module, que são úteis para mapear características de consciência em outros projetos semelhantes ao PBPMF, foram apresentadas. E foi possível identificar que o framework possui características de software consciente (a ação do Trade Processor entregando informação para que o Predictor Module estime e preveja as rotas de execução de atividades). Essa consciência permite, então, que o software tome melhores decisões, e as apresente como sugestão ao usuário final.

4.8 Design of the Outer-tuning framework: self-tuning and ontology for relational

4.8.1 Introdução do Estudo de Caso

O artigo (Oliveira, et al., 2015) disserta acerca da arquitetura do framework chamado Outer-tuning, que suporta a sintonia fina semiautomática de um SGBD através da ontologia proposta por de Almeida (de Almeida, 2013). A sintonia fina, ou seja, o tuning, tem como objetivo otimizar o desempenho nas consultas ou atualização dos dados. A otimização ocorre com a criação ou eliminação de índices ou visões materializadas de forma dinâmica no banco de dados em função das estatísticas mantidas pelo próprio banco de dados.

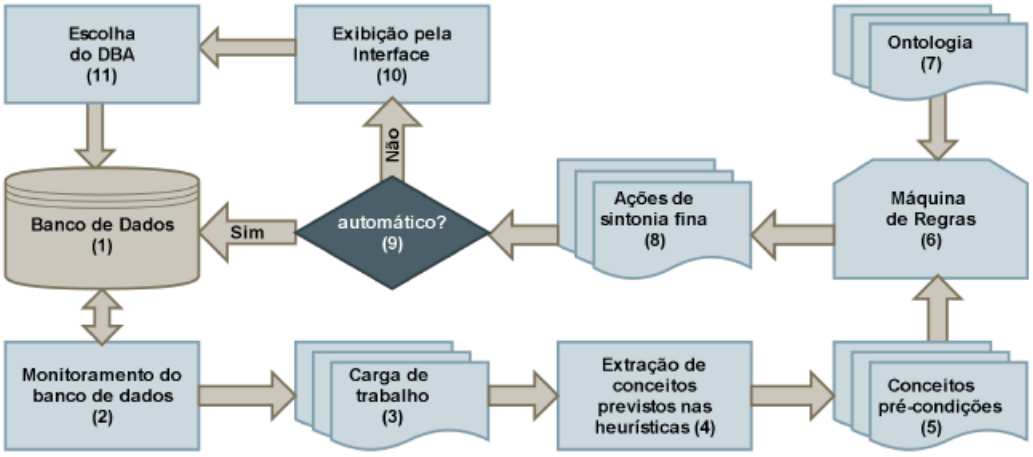

Figura 32 – Fluxo de execução planejado para o Outer-Tuning

4.8.2 Motivação para Inclusão do Estudo de Caso

O trabalho monitora as métricas do banco de dados e, através de uma ontologia OWL-DL[1] e da linguagem SWRL[2], extrai as otimizações para serem aplicadas ao banco de dados, desde que tais otimizações não conflite com as regras manuais estabelecidas pelo DBA. Este trabalho enriquece nossa pesquisa, porque não é um trabalho pensado no MAPE-K (3 ou em sistemas auto adaptativos, propondo-se em ser uma ferramenta de auxílio ao DBA.

1 Web Ontology Language – Description Logic

2 Semantic Web Rule Language

4.8.3 Operacionalizações

O Outer-Tuning realiza as seguintes operacionalizações:

- Consciência de Contexto
- Consciência do Auto Comportamento
- Consciência das Relações Sociais

4.8.3.1 Consciência de Contexto

A Consciência de Contexto é alcançada pelo Ambiente Computacional, refinado pelos Modelos de Recursos e Informações, e, pelos Padrões de Trocas de Informações. O componente Capturador de Carga de Trabalho (monitor), utiliza a linguagem SQL para recuperar a carga de trabalho. Em conjunto com a ontologia Outer-Tuning estes componentes respondem as seguintes perguntas: Como os recursos podem ser acessados, Como as informações podem ser acessadas, Como as informações são armazenadas e Como as informações são trocadas entre diferentes atores.

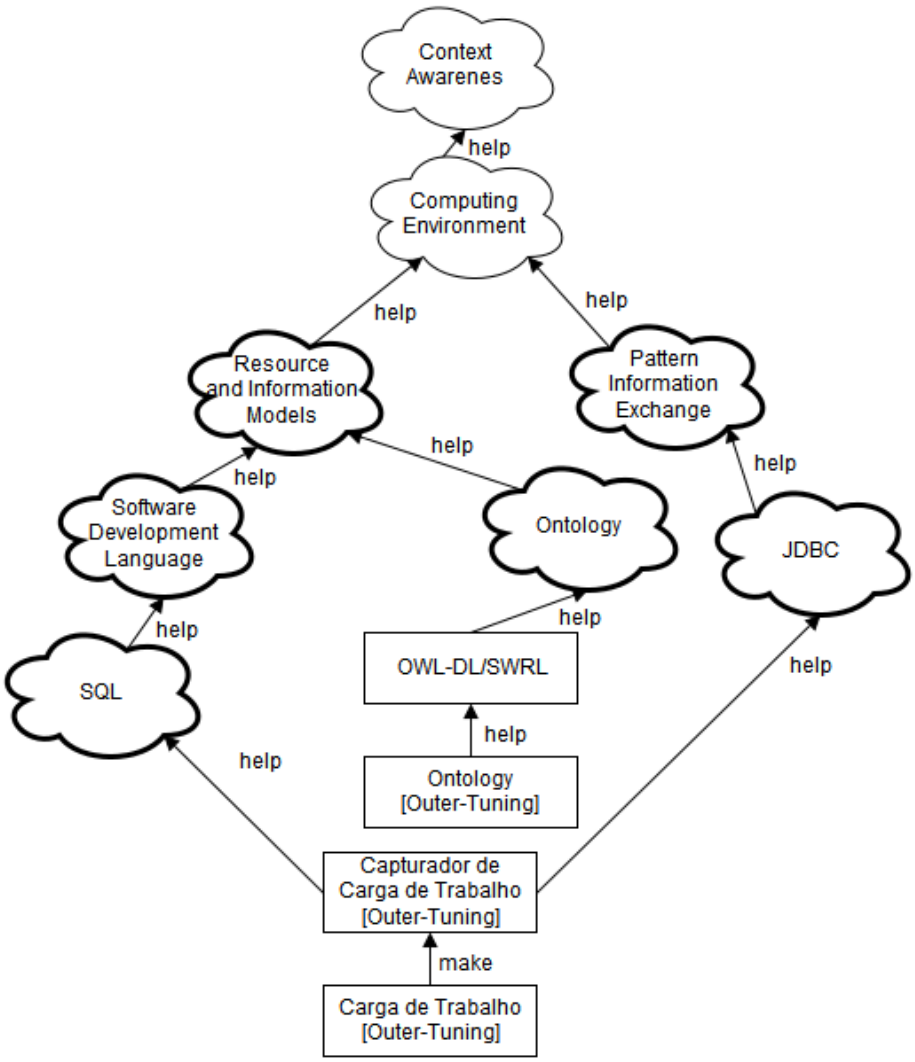

Figura 33 – Consciência de Contexto [Outer-Tuning]

4.8.3.2 Consciência do Auto Comportamento

A consciência do auto comportamento é realizada pela Consciência d Alternativas, refinada pela Avaliação de Alternativas sob o Impacto do Contexto.

O componente Executor de Ações aplica a alternativa selecionada pela máquina de regras dentro das possibilidades do atual contexto, que por sua vez é definido pela biblioteca de funções do domínio e da carga de trabalho capturada, enquanto que a Ontologia provê o mecanismo para escrever os conceitos em linguagem formal para que a Máquina de Regras possa inferir a melhor alternativa. O conjunto de componentes responde as seguintes perguntas: Como colocar as alternativas no software, Como avaliar as alternativas atuais de acordo com o contexto corrente e, parcialmente, Qual o impacto de escolher cada alternativa, visto que o Outer-Tuning consegue estabelecer um custo/benefício esperado para cada alternativa, mas não sabe como avaliar se o custo/benefício esperado valeu ou não a pena, ou seja, o Outer-Tuning pode decidir que vale a pena criar um índice num determinado momento, mas não sabe dizer se a médio ou longo prazo, este mesmo índice ainda continuará valendo a pena, visto que nada impede que um índice seja criado e algum tempo depois, excluído, de forma que o custo de criação e exclusão do índice diminua o custo/benefício alcançado durante seu uso.

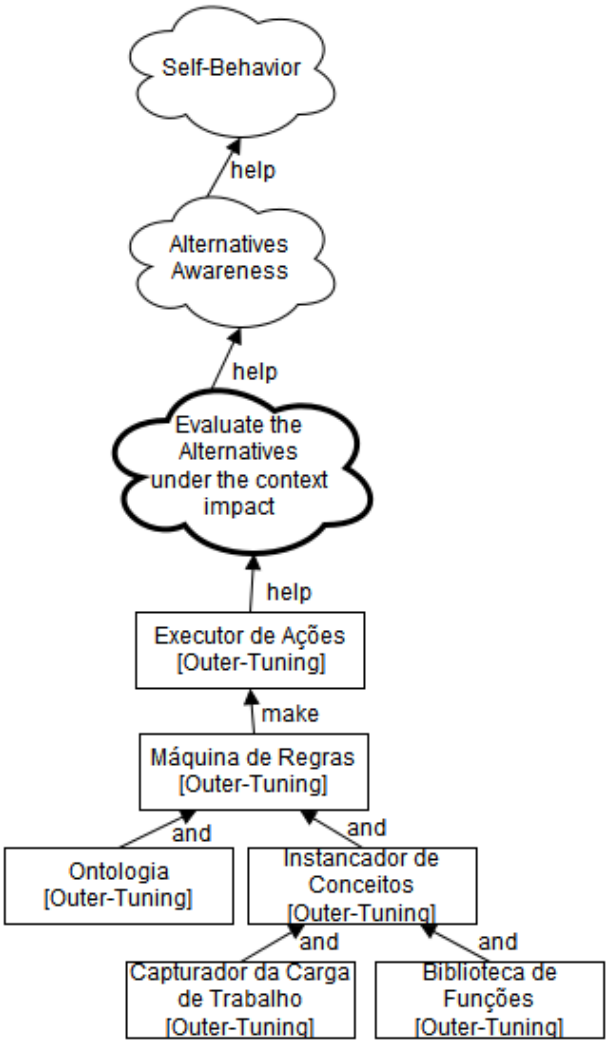

Figura 34 – Consciência do Auto Comportamento [Outer-Tuning]

4.8.3.3 Consciência do Contexto Social

A Consciência do Contexto Social, fica clara no momento em que o Outer-Tuning leva em conta as preferências do DBA, antes de aplicar ou não um ajuste. A interface do DBA responde as perguntas: Quais são as preferências dos usuários e Quais ações os usuários realizaram.

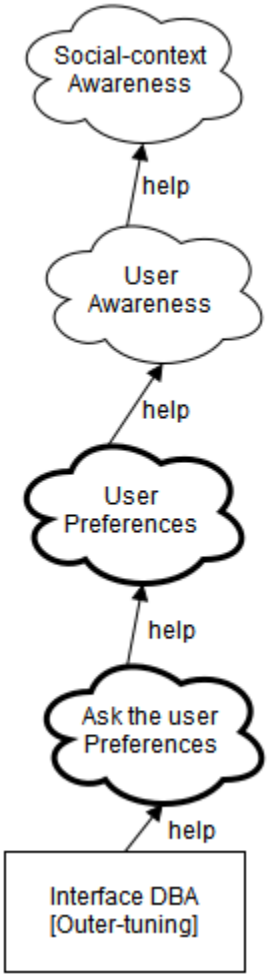

Figura 35 – Consciência de Relações Sociais [Outer-Tuning]

4.8.4 Conclusões

O estudo de caso mostrou-nos um software consciente em um software que não tinha o objetivo inicial de ser auto adaptativo. Entretanto, a consciência de software claramente mostrou uma melhora significativa no desempenho dos bancos de dados utilizados nos experimentos.

Falta ao Outer-Tuning o "K" (Knowledge) do MAPE-K (3 . Com um log de ações realizadas, o Outer-Tuning poderia medir a eficácia das mudanças aplicadas, o que lhe permitiria responder melhor a pergunta: Qual o impacto de escolher cada alternativa e, ao mesmo tempo, adicionaria ao Outer-Tuning a Consciência Funcional, porque ele responderia a pergunta: O quão efetivo o software está sendo.

4.9 From Unknown to Known Impacts of Organizational Changes on Socio-technical Systems

4.9.1 Introdução do Estudo de Caso

Ferreira, Maiden e Leite propõem um framework para antecipar possíveis impactos de mudanças organizacionais tanto na própria organização quanto nos requisitos do sistema sócio técnicos (Ferreira, et al., 2015). Os impactos organizacionais podem implicar na geração de novos requisitos, mudança de existentes. Os autores organizam uma coleção de ferramentas manuais que permitem modelar o estado atual da organização (AS IS model) para que através do modelo dinâmico organizacional (DOM), que propõe e apoiado em banco de questões, se possa modelar o(s) cenários futuros da organização (TO BE model) e como eles poderiam impactar nos requisitos de software.

4.9.2 Motivação para Inclusão do Estudo de Caso

A proposta visa dar suporte aos interessados para que entendam sobre mudanças organizacionais e seu fluxo de impactos. Esse maior entendimento traria maior consciência sobre a organização e acreditamos que há uma contribuição para a criação ou para as mudanças em requisitos de software com maior qualidade.

4.9.3 Operacionalizações

4.9.3.1 Consciência de Contexto Social

Strategy dimension identification: Serve para modelar os elementos organizacionais relacionados à estratégia da organização como por exemplo as políticas, normas, objetivos entre outros. Nesse sentido, ajuda na operacionalização de Norm-Awareness. Essa operacionalização ajuda também na formação do DOM.

Dynamic Organizational Model (DOM): é um modelo organizacional a ser usado como base para à identificação dos elementos da organização que podem mudar ou ser impactados por mudanças advindas do plano estratégico. Essa operacionalização ajuda na consciência social do contexto.

Impact Uncertainty Grid: É uma ferramenta manual que serve para que os interessados tenham consciência das incertezas e tendências que podem ocorrer e provocar mudanças na organização. Esta operacionalização se encaixa no softgoal *Define dependencies that can be established* porque captura tendências e incertezas que podem ocorrer quando a organização interage no seu ambiente social.

Essa grid precisa do Software change source taxonomy.

Software change source taxonomy: é uma taxonomia criada por o (McGee, Sharon, & Greer, 2012) baseada em triggers e incertezas organizacionais que podem ocorrer e mudar o software.

Culture dimension identification: Serve para modelar os elementos organizações relacionados à cultura da organização, como por exemplo: os valores, crenças, sentimentos, mitos, ideologias, entre outros. Nesse sentido, este elemento ajuda na operacionalização da consciência dos relacionamentos sociais da organização como também na formação do DOM.

Market dimension identification: Serve para modelar os elementos externos da organizações, por exemplo: os aspectos políticos, legais, éticos, sócio-culturais, tecnológicos, em função do mercado. Este componente do framework ajuda na consciência dos relacionamentos sociais da organização como também na formação do DOM.

Stakeholders dimension identification: Ajuda na identificação de usuários, como os empregados, clientes, fornecedores, e patrocinadores entre outros. Bem como seus objetivos e necessidades. Essa operacionalização ajuda na consciência dos usuários como também na formação do DOM.

Framing Checklist: é uma ferramenta manual que ajuda na identificação dos interessados que participaram no planejamento estratégico. Essa operacionalização também se alinha com a consciência dos usuários. Também contribui para a consciência sobre os objetivos do projeto do planejamento estratégico que também contribui para a consciência própria dos objetivos. Essa operacionalização ajuda a identificar situações do contexto e também com o plano estratégico.

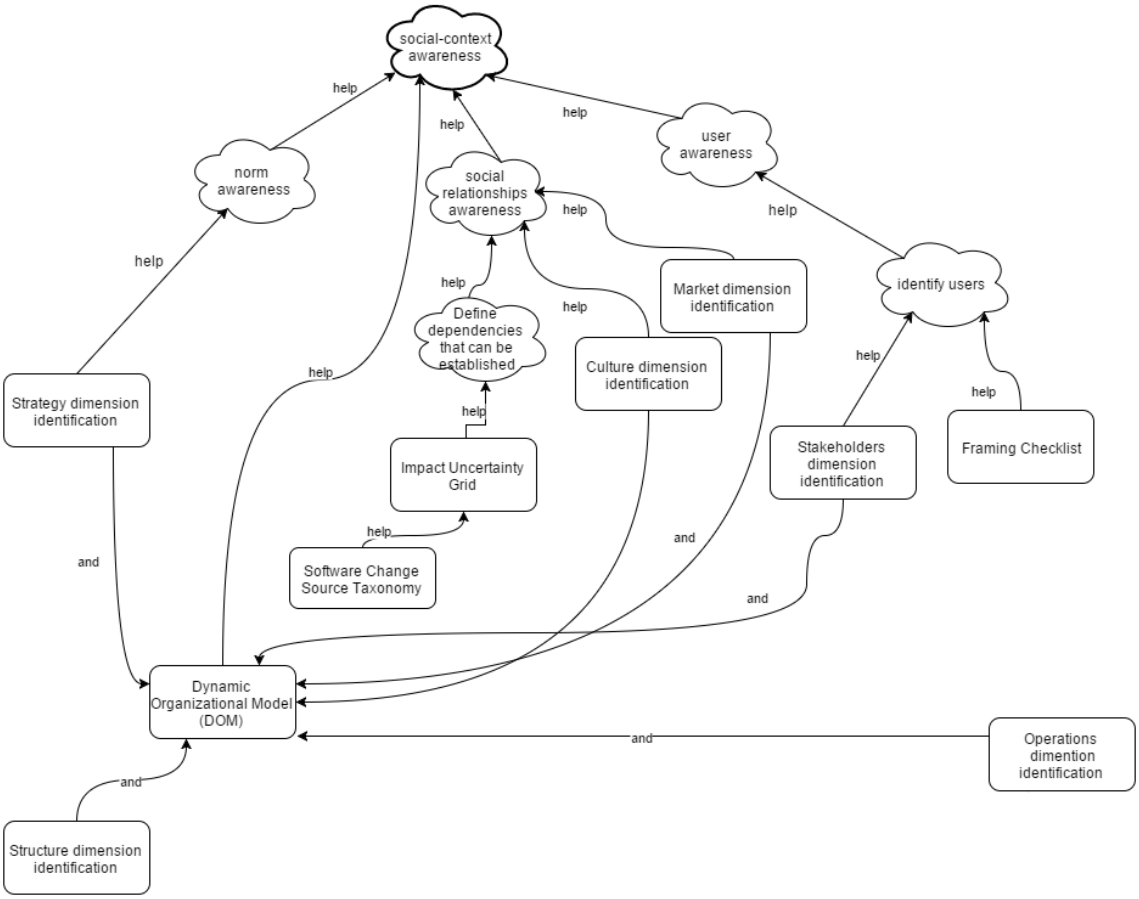

Figura 36 – Consciência de Contexto Social

4.9.3.2 Consciência de Contexto

Structure dimension identification: Serve para modelar os elementos organizacionais relacionados à estrutura da organização, por exemplo: a estrutura dos processos, funções, os serviços, produtos, etc. Nesse sentido, este elemento ajuda na consciência do contexto,

especificamente no softgoal location com um assunto de domínio. Essa operacionalização ajuda também na formação do DOM.

Domain-assumption: Essa operacionalização foi reusado de um dos casos de esta monografia. O modelagem do aspecto dinâmico é ajudado pelo DOM.

Operations dimension identification: Serve para modelar os elementos organizações relacionados às operações da organização como por exemplo as entradas das atividades, recursos, atividades, regras, e saídas. Nesse sentido, ajuda na consciência das funções para a própria consciência da organização. Essa operacionalização ajuda também na formação do DOM.

Impact Operational Scenarios: Com o DOM, auxilia na análise e na avaliação de cenários futuros da organização. Essa operacionalização ajuda na identificação das alternativas que a ajudam na consciência da organização.

To be models: é uma operacionalização que através dos modelos orientado a objetivo ajuda na consciência dos objetivos da organização no futuro.

Database of Question (DBQ): é um conjunto de questões que contribuem para a formação dos modelos de informação e de recursos que compõem o ambiente computacional.

Scenario Cockpit: é uma ferramenta manual que ajuda no monitoramento do contexto no ambiente físico para identificar quais são as estratégias que devem ser colocadas em prática.

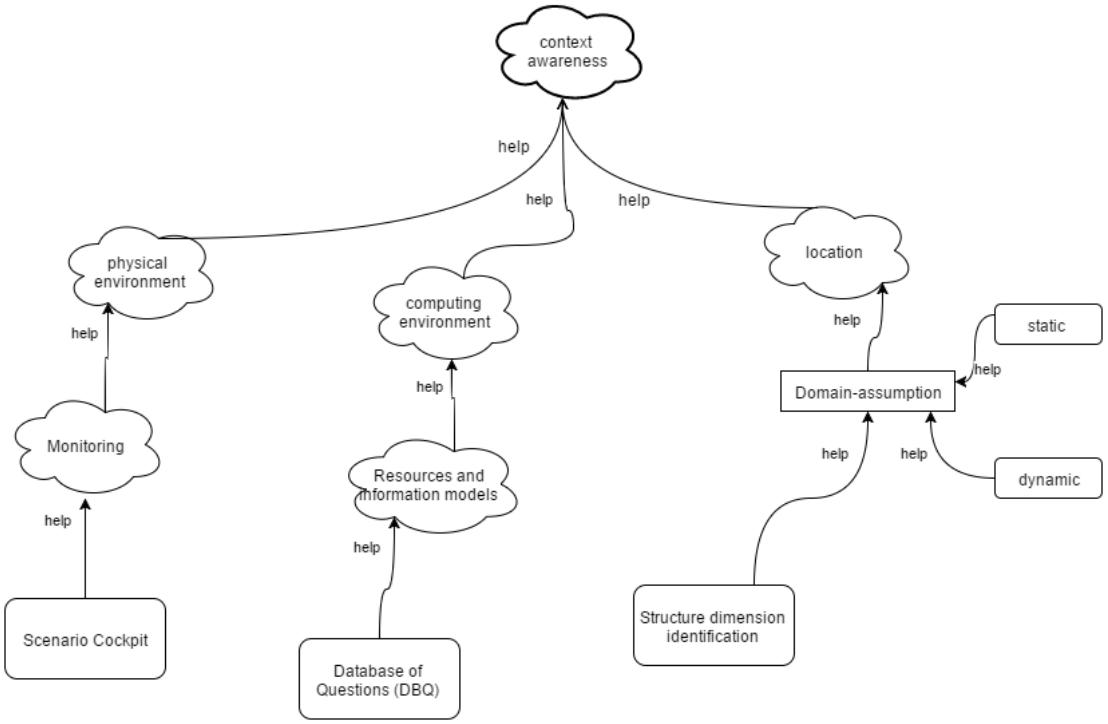

Figura 37 – Consciência de contexto

4.9.3.3 Consciência de Auto Comportamento

As is models: é uma operacionalização que através dos modelos orientado a objetivo ajuda na consciência dos objetivos da organização no momento atual.

Strategic cenario matrix: è uma ferramenta manual que ajuda a identificar cenários de futuros organizacionais. Essa operacionalização ajuda a identificar situações do contexto e também com o plano estratégico.

Strategic Plan: é o plano estratégico que ajuda na avaliação das alternativas sob impactos do contexto. Serve de entrada para a construção dos modelos to-be.

SWOT Analysis: É uma atividade que ajuda na descrição do contexto e na construção do plano estratégico identificando as forças, fraquezas, oportunidades e ameaças da organização.

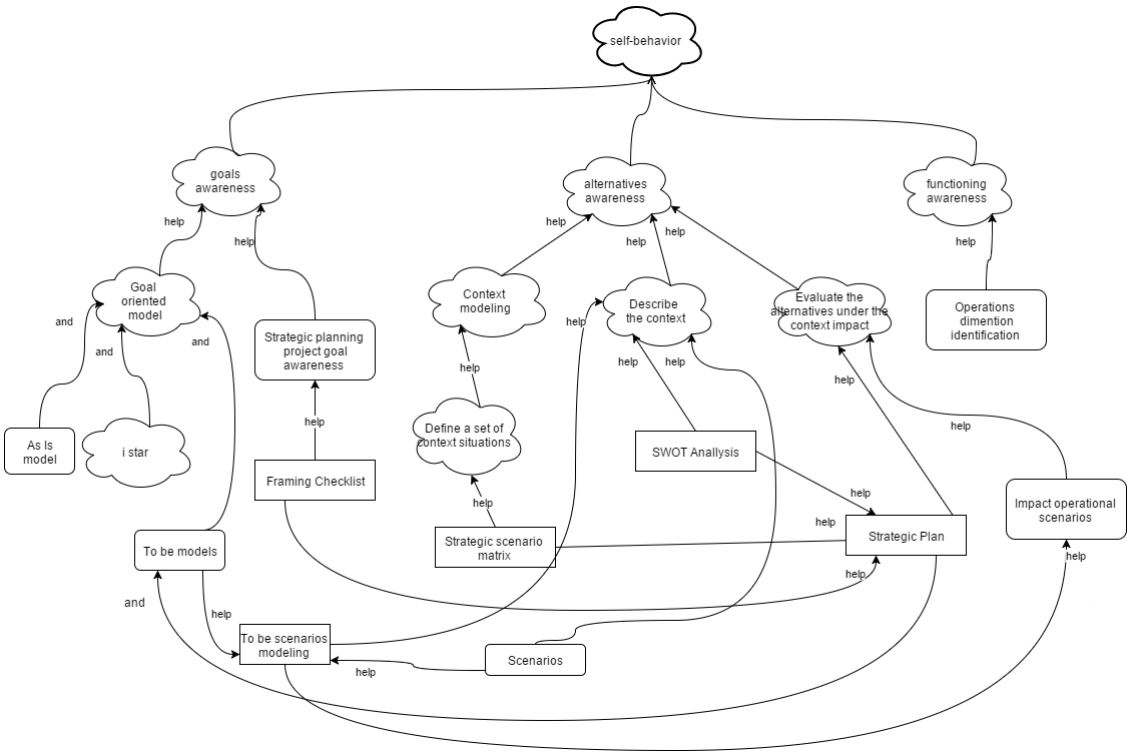

Figura 38 – Consciência do Auto Comportamento

4.9.4 Conclusões

As atividades de mapeamento auxiliou na identificação de como a proposta de (Ferreira, Maidem, e Leite 2015) ajuda na consciência organizacional e classificá-lo desde o ponto de vista da consciência de software consciente proposto por Souza Cunha.

4.10 JAT A Test Automation Framework for Multi-Agent Systems

4.10.1 Introdução do Estudo de Caso

A tecnologia de agentes emergiu como uma solução promissora para tratar sistemas complexos e baseados em rede sendo rapidamente aplicada à vários domínios incluindo: e-commerce, gerenciamento de conhecimento, games, etc. Contudo, se por um lado a tecnologia de agentes apoia os requisitos das aplicações correntes, por outro lado, suas

características inerentes ao software desenvolvido neste paradigma, tais como: autonomia, habilidade social, reatividade, pró-atividade, comunicação assíncrona e baseada na troca de mensagens, trazem novos obstáculos a testabilidade dessa categoria de software.

No artigo escrito por Coelho et. al., foi proposta uma abordagem para testes de sistemas multiagentes os quais foram desenvolvidos com o framework Jade. A solução apresentada fornece um conjunto de assertivas no estilo JUnit (conhecido framework de testes para Java) para verificação do comportamento executado pelo agente que está sendo testado. A interação entre os agentes é simulada com o uso de agentes "mock", ou seja, implementações falsas de agentes reais criadas com o estrito propósito de testar a comunicação entre os agentes.

Na abordagem de testes proposta por Coelho et al., foi utilizada a programação orientada a aspectos (POA) para monitorar a execução do agente em teste (agent under test - AUT) e a troca de mensagens deste com os agentes mock. A linguagem ASPECTJ foi utilizada para implementar os aspectos responsáveis por armazenar em estruturas de dados especiais a transição dos estados internos do AUT.

4.10.2 Motivação para Inclusão do Estudo de Caso

A testabilidade, de forma geral, se apoia em duas características cruciais: (i) a controlabilidade - que é a capacidade de controlar a entrada do componente que está sendo testado e a (ii) observabilidade - que é a capacidade de observar o resultado obtido sobre o componente que está sendo testado. Neste sentido, a tarefa de testar o software baseado no paradigma de agentes torna-se ainda mais desafiador do que o teste de "softwares tradicionais". Características inerentes aos agentes de software, tais como: autonomia e comunicação assíncrona entre os agentes são exemplos de duas características que tornam o teste de agentes desafiador e diferentes dos testes de sistemas tradicionais. No geral, em um sistema multiagente, cada agente atua para alcançar seus próprios objetivos sem, necessariamente, conhecer a existência de outros agentes do sistema. A escolha deste trabalho visa verificar como os componentes de uma ferramenta de testes de agentes podem corroborar na operacionalização da consciência do software como um todo.

4.10.3 Operacionalizações

Segundo Truong (Truong, 2009), a consciência de contexto permite ao software além de ser capaz de lidar com as mudanças no meio ambiente em que o software opera, também melhorar a resposta de sua utilização. Essa afirmação vem de encontro à natureza adaptativa dos sistemas multiagentes situados em ambientes dinâmicos e distribuídos.

4.10.3.1 Consciência de Contexto

Nesse artigo é possível identificar dentro da consciência de contexto, a consciência do ambiente computacional, conforme descrito no modelo SIG proposto por Herbert et al. A presença dos componentes JAT, Monitor, TestReporter, AgentStateMachine e XPI são responsáveis por ajudar na operacionalização da consciência de software MAS Middleware, conforme apresentado na figura X.

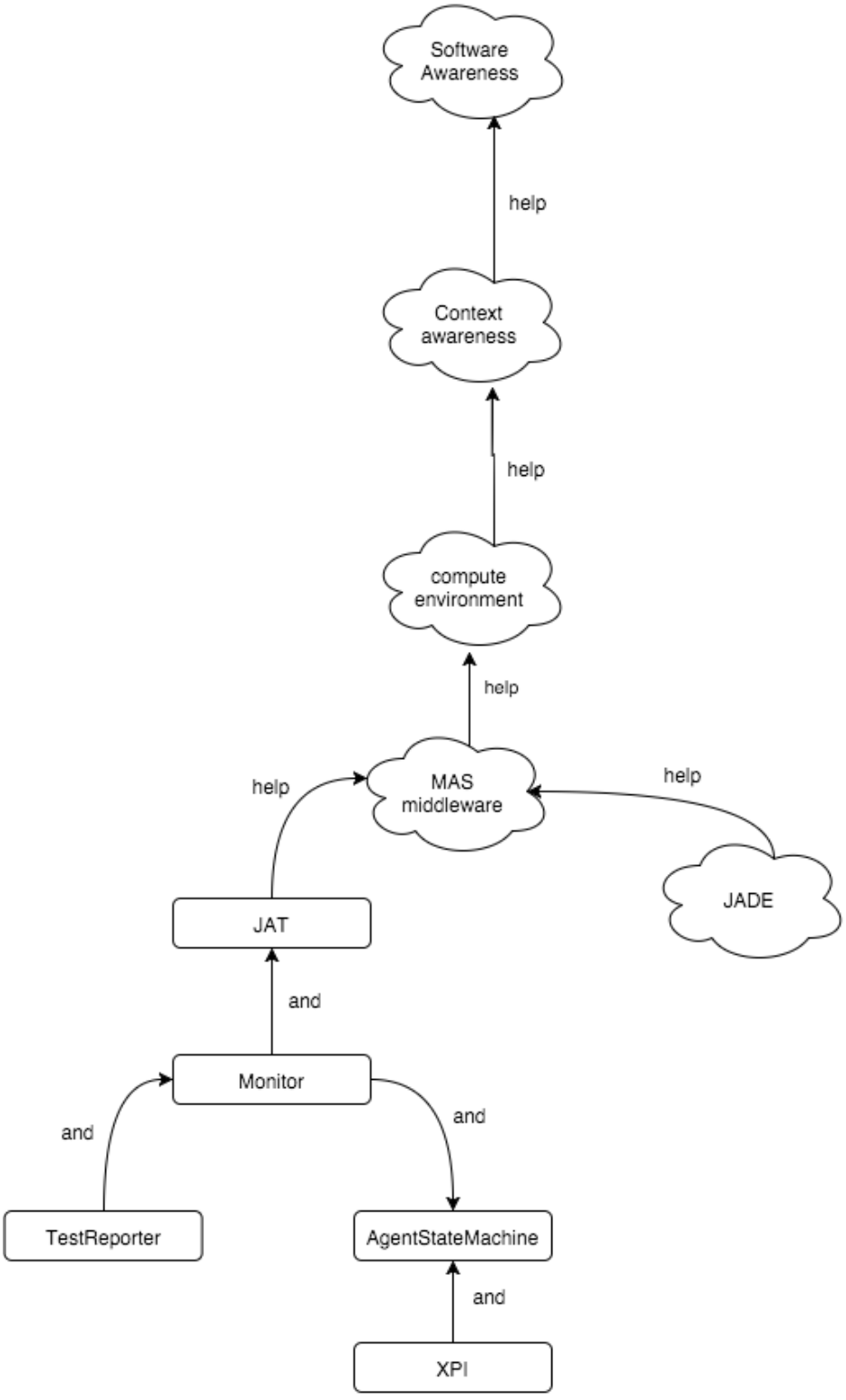

Figura 39 – Consciência de contexto (MAS – Middleware) [JAT]

Na solução proposta pelo artigo, os componentes ASPECTJ, AgentStateMachine e XPI que ajudam na operacionalização da consciência de "Software Development Language". A linguagem ASPECTJ é utilizada para interceptar a execução dos agentes e da plataforma coletando as informações executadas durante o teste. O componente AgentStateMachine é o aspecto responsável por armazenar o estado interno dos agentes (monito-

rando o ciclo de vida do agente). O componente XPI é o elemento que acrescenta modularidade e flexibilidade a solução proposta, uma vez que define as interfaces necessárias da solução. Para operacionalizar a consciência de Ontologia temos o componente JadeTestCase que fornece o conjunto de assertivas para verificação do comportamento executado pelo agente durante o teste. O componente a JadeTestCase é uma especialização do framework de testes JUnit e também utiliza um modelo de faltas. O modelo de faltas define quais faltas são relevantes para o teste assim como em que condições estas ocorrem. A figura X apresenta os componentes desta operacionalização.

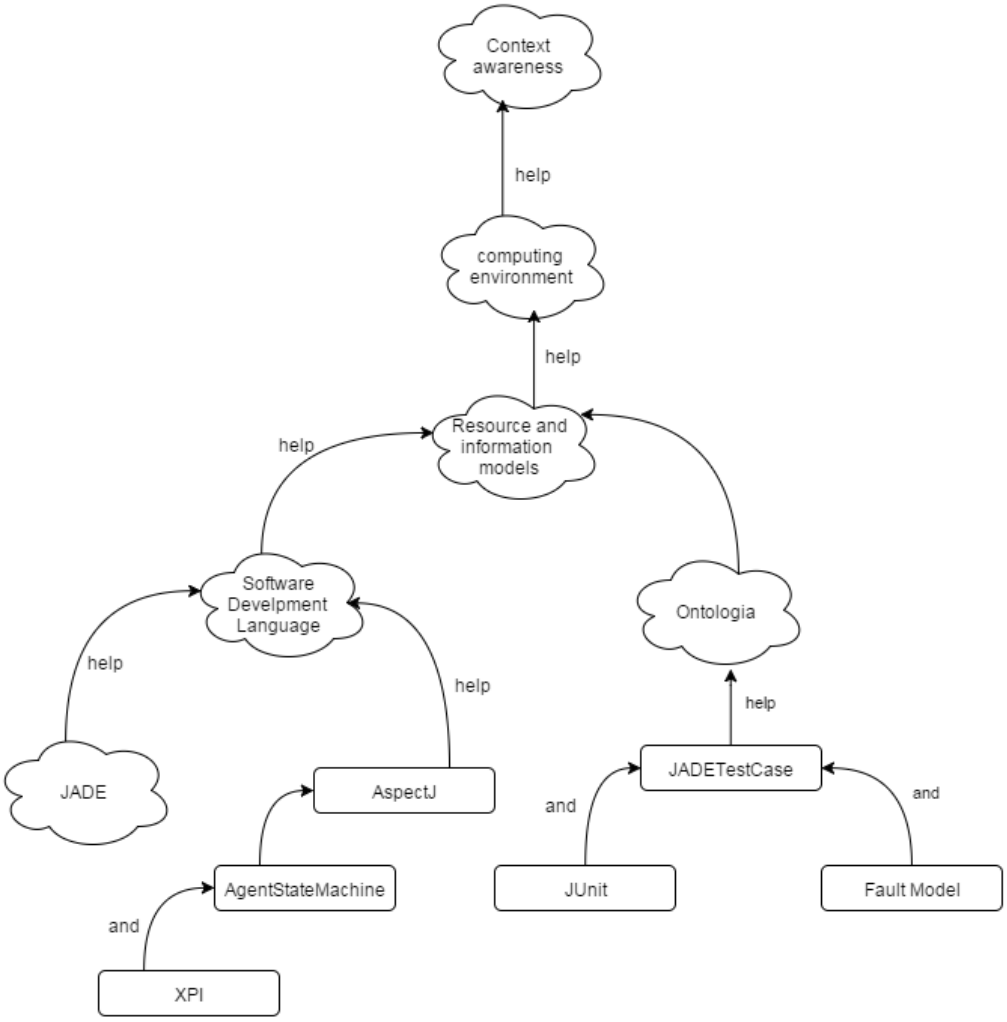

Figura 40 – Consciência de Contexto (Resource and Information Models) [JAT]

Finalizando, temos os componentes AgentStateMachine, XPI e o componente JadeMockAgent que depende os componentes ACL, Synchronizer e MockGenerator para ajudar na operacionalização da consciência de "Pattern to Information Exchange". O componente JadeMockAgent é responsável por criar "agentes falsos" (fake) com o propósito de testar a comunicação do agente em teste com outros agentes do sistema. Esta estratégia permite maior controle sobre o contexto em que ocorrem as trocas de mensagens entre os agentes e, permite ainda, monitorar o comportamento do agente sob teste. Os falsos agentes representam agentes reais do sistema e são criados estritamente para testar a comunicação dos agentes. O componente Synchronizer é responsável por "orquestrar" a execução dos agentes mock. O componente ACL define o padrão de comunicação estabelecido entre os agentes definido pela Foundation for Intelligent Physical

Agents (FIPA). O MockGenerator é o componente que permite o reuso de agentes mock para diferentes cenários de testes e também faz uso da ACL para comunicação. A operacionalização desta parte está representada na figura X.

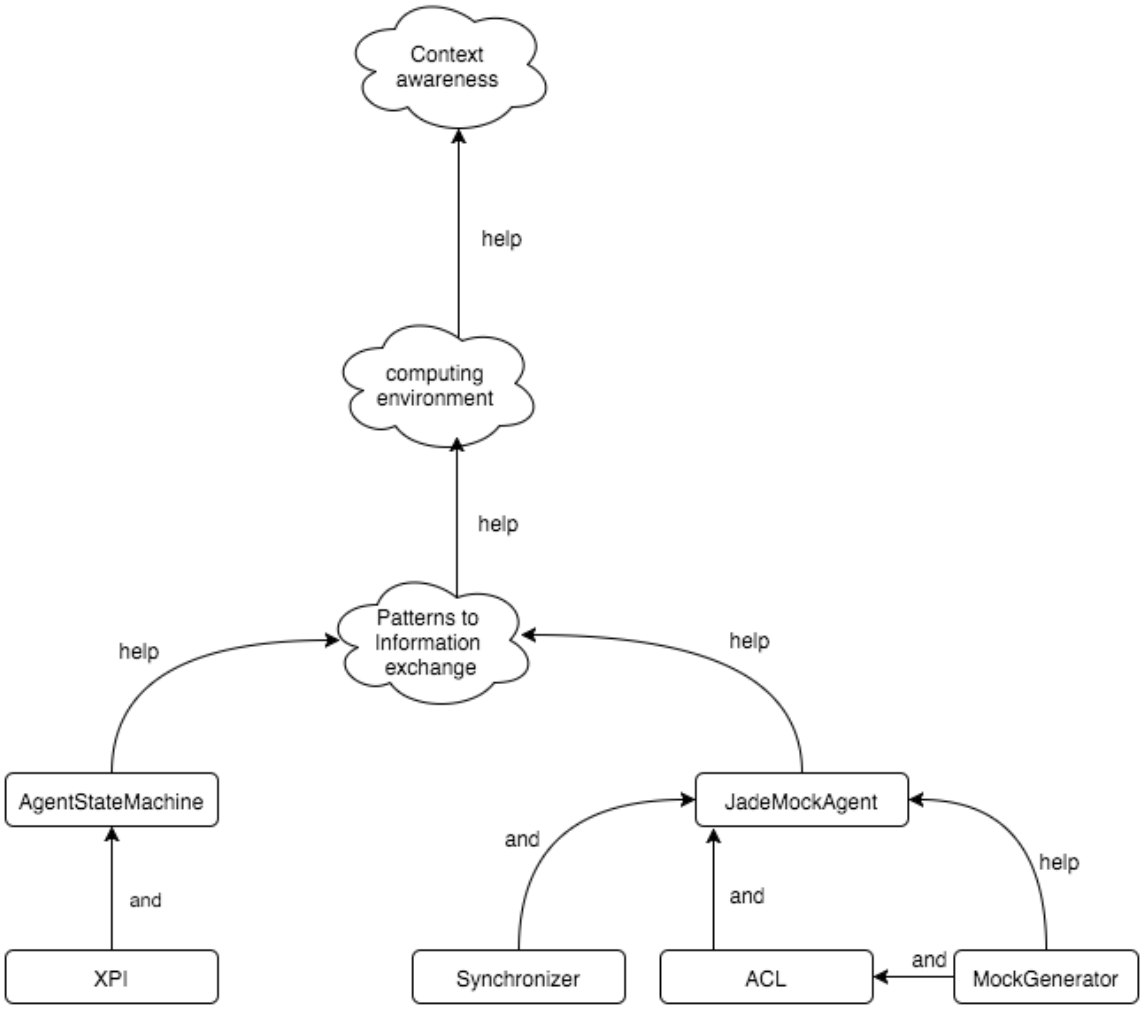

Figura 41 – Consciência de contexto (Patterns to Information Exchange) [JAT]

4.10.4 Conclusões

Para o artigo apresentado, o software apresentou diferentes requisitos de consciência, os quais certamente, melhora a qualidade do software proposto. Intuitivamente, a presença de requisitos de consciência parecem ser uma característica marcante no desenvolvimento utilizando o paradigma de agentes de software. Contudo, essa é apenas uma observação baseada na intuição sendo necessária a exploração de outros trabalhos dessa área.

5 Comparativo entre os Estudos de Caso

Nesta seção, fizemos um comparativo entre as operacionalizações de cada estudo de caso.

Analisando a Tabela 1, podemos perceber que somente os estudos de caso 4.4 [MORPH] e 4.7 [PBPMF] não apresentam Consciência de Contexto. Ambos os estudos de caso são propostas de arquiteturas que encapsulam a monitoração de sensores ou dados, ou seja, já a considera como certa.

Como observamos no MAPE-K (3), não há adaptação, sem que haja uma monitoração prévia. Algo precisa ser monitorado para que ocorra ou uma reação às mudanças do estado do sistema ou que este se adapte proativamente a uma situação identificada pela monitoração do estado.

Os demais estudos de caso são experimentos implementados ou modelagens bastante completas que anteciparam a monitoração de algum sensor físico ou de software.

O estudo de caso 4.3 [LUA Profiler] não tem adaptação, somente monitoração.

Uma observação mais detalhada da Tabela 1 nos permite concluir que softwares que realizam monitoração, ainda que não tenham utilizado o MAPE-K como paradigma de modelagem, apresentam Consciência de Contexto, enquanto que os softwares que se adaptam o fazem no aspecto de Consciência do Auto Comportamento ou no aspecto de Consciência do Contexto Social.

	Estudo de Caso									
Aspecto de Consciência	4.1	4.2	4.3	4.4	4.5	4.6	4.7	4.8	4.9	4.10
Contexto - Físico	x					x			x	
Contexto - Localização	x					x			x	
Contexto - Ambiente Computacional		x	x		x	x		x	x	x
Tempo - Intervalo	x	x								
Tempo - Linear	x	x				x				
Auto Comportamento - Metas	x				x	x			x	
Auto Comportamento - Alternativas	x			x	x	x	x	x	x	
Auto Comportamento - Funcional	x			x	x	x	x		x	
Contexto Social - Normas									x	
Contexto Social - Relações Sociais						x			x	
Contexto Social - Usuários		x			x			x	x	

Tabela 1 – Comparativo entre os estudos de caso

6 Conclusões

Mostramos vários estudos de caso, selecionados por pesquisadores de áreas diversas, alguns sem conhecimento prévio do trabalho de Souza Cunha.

Nossas contribuições são: (i) um conjunto de exemplos de como utilizar o catálogo de Consciência de Software e suas operacionalizações proposto por Souza Cunha, (ii) mostramos que Consciência de Software e Software Adaptativo são conceitos diferentes, uma vez que softwares de natureza não adaptativa e que sequer utilizaram o modelo MAPE-K apresentaram Consciência de Software em vários aspectos, e, (iii) nosso trabalho mostrou que a monitoração está ligada à Consciência de Contexto ou de Tempo, enquanto que a adaptação está ligada à Consciência do Auto Comportamento ou do Contexto Social.

7 Trabalhos Futuros

Mais estudos de caso são necessários para comprovar nossa conclusão acerca da monitoração estar vinculada à Consciência de Contexto ou Tempo, enquanto que a adaptação está ligada à Consciência do Auto Comportamento ou do Contexto Social.

Referências Bibliográficas

Alexander, I., & Beus-Dukic, L. (2009). In: *Discovering Requirements: How to Specify Products and Services* (pp. 17-18). Wiley.

Alrajeh, D., Kramer, J., Russo, A., & Uchitel, S. (2015). Automated Support for Diagnosis and Repair. *Communications of the ACM, 58(2)*, pp. 65-72.

Braberman, V., D'Ippolito, N., Kramer, J., Sykes, D., & Uchitel, S. (2015). Morph: A reference architecture for configuration and behaviour self-adaptation. *Proceedings of the 1st International Workshop on Control Theory for Software Engineering, ACM*, pp. 9-16.

Brooks, F. (2009). A Science of Design is a Misled and Misleading Goal. In: M. J. Mylopoulos (Ed.), *In Perspectives Workshop: Science of Design: High-Impact Requirements for Software-Intensive Systems,*. Dagstuhl Seminar Proceedings.

Chung, L. e. (2000). Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers.

Coelho, R., Cirilo, E., Kulesza, U., v. S., A., R., & A., L. C. (2007). Jat: A test automation framework for multi-agent systems. *IEEE International Conference on Software Maintenance*, pp. 425-434.

de Almeida, A. C. (2013). *Framework para apoiar a sintonia fina de banco de dados.* Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro .

de Souza Cunha, H. (2014). *Desenvolvimento de Software Consciente com Base em Requisitos.* Pontíficia Universidade Católica do Rio de Janeiro. Rio de Janeiro : Pontifícia Universidade Católica do Rio de Janeiro .

do Prado Leite, J. C., Werneck, V., Oliveira, A. D., Cappelli, C., Cerqueira, A. L., de Souza Cunha, H., & Gonzalez-Baixauli, B. (2007). Understanding the Strategic Actor Diagram: an Exercise of Meta Modeling. *InWER*, pp. 2-12.

Dominguez, C., Vidulich, M., Vogel, E., & McMillan, G. (1994). *Situation awareness: Papers and annotated bibliography. Armstrong Laboratory.* Armstrong Laboratory, Human System Center, ref. AL/CF-TR-1994-0085.

ELLIS, C. A., GIBBS, S. J., & REIN, G. (1991). Groupware - Some Issues and Experiences. *Communications of the ACM, Vol. 34, No. 1*, pp. 38-58.

Endsley, M. R. (1998). A comparative analysis of SAGAT and SART for evaluations of situation awareness. *Proceedings of the Human Factors and Ergonomics Society 42nd Annual Meeting*, (pp. 82-86). Santa Monica, CA, USA.

Ferreira, M. G., Maiden, N., & do Prado Leite, J. C. (2015). *From Unknown to Known Impacts of Organizational Changes on Socio-technical Systems.* Acesso em 30 de 11 de 2015, disponível em https://pdfs.semanticscholar.org/3552/9dfa4f9ece179f6434753b1b7735af7c7362.pdf

Filieri, A., Maggio, M., Angelopoulos, K., D'Ippolito, N., Gerostathopoulos, I., H. A., & Krikava, F. (2015). Software engineering meets control theory. *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (pp. 71-82). IEEE Press.

Fuks, H., Raposo, A., Gerosa, M., & Lucena, C. (2005). Applying the 3C Model to Groupware Development. *International Journal of Cooperative Information Systems (IJCIS)*, pp. 299-328.

Goguen, J. A. (2003). "*Consciousness studies*" *Encyclopedia of Science and Religion.* MacMillan Reference USA.

Grupo de Engenharia de Requisitos da PUC-Rio. (2012). *Catálogo Transparência*. Acesso em 15 de 12 de 2015, disponível em http://transparencia.inf.puc-rio.br/wiki/index.php/Cat%C3%A1logo_Transpar%C3%AAncia

Ierusalimschy, R. (2013). *Programming in Lua* (Third Edition ed.). Lua.Org.

Istarwiki. (s.d.). *i* Framework - Wiki of the Computer Science Group at UofT*. Fonte: http://istar.rwth-aachen.de/tiki-view_articles.php

Kephart, J., & Chess, D. (2003). The vision of autonomic computing. *Computer 36.1*, 41-50.

Lehman, M. M., & Ram, J. F. (2001). Rules and Tools for Software Evolution Planning and Management. *Annals of Software Engineering, Volume 11, Issue 1*, 15-44.

Leite, J. (2012). Software Requirements Are Soft. *In Requirements Management–Novel Perspectives and Challenges - Dagstuhl Seminar 12442*.

Maggi, F. M., Di Francescomarino, C., Dumas, M., & Ghidini, C. (2014). Predictive monitoring of business processes. *Advanced Information Systems Engineering, Springer International Publishing*, pp. 457-472.

McGee, Sharon, & Greer, D. (2012). Towards an understanding of the causes and effects of software requirements change: two case studies. *Requirements Engineering 17.2* (pp. 133-155). Springer.

Musa, P. M., & Ierusalimschy, R. (2015). *Profiling. Memory in Lua.* Rio de Janeiro: Pontíficia Universidade Católica do Rio de Janeiro.

Oliveira, R. P., Lifschitz, S., Almeida, A. C., & Haeusler, H. (2015). Design of the Outer-tuning framework: self-tuning and ontology for relational databases. *XI Brazilian Symposium of Information System*, pp. 171-178.

PUC-Rio, Laboratório de Engenharia de Software (LES). (s.d.). *Catálogo de Transparência-Wiki do Grupo*. Fonte: http://www.er.les.inf.pucrio.br/~wiki/index.php/Transparencia-de-Software

Serrano, M., & Leite, J. (2011). Capturing transparency-related requirements patterns through argumentation. *First International Workshop on Requirements Patterns (RePa)*, (pp. 32-41).

Silva Souza, V. E., & Mylopoulos, J. (2015). Designing an adaptive computer-aided ambulance dispatch system with Zanshin: an experience report. *Software: Practice and Experience 45.5*, 689-725.

Silva Souza, V., Lapouchnian1, A., Robinson, W. N., & Mylopoulos, J. (2012). *Awareness Requirements.* Extended version of the article "Awareness Requirements for Adaptive Systems" published in Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS ”11).

Simon, H. A. (1996). *The Sciences of the Artificial* (3rd ed.). Cambridge, MA, USA: MIT Press.

Sourour, M., Adel, B., & Tarek, A. (2009). Environmental awareness intrusion detection and prevention system toward reducing false positives and false negatives. *Computational Intelligence in Cyber Security*, pp. 107-114.

Sterritt, R., & Hinchey, M. G. (2006). Biologically-inspired concepts for self-management of complexity. *11th IEEE International Conference on Engineering of Complex Computer Systems*, (p. 6).

Tian, X., Bar-Shalom, Y., Chen, G., Blasch, E., & Pham, K. (2012). A unified cooperative control architecture for UAV missions. *International Society for Optics and Photonics, SPIE Defense, Security, and Sensing*, pp. 83920X-83920X.

Truong, H. L. (2009). A survey on context-aware web service systems. *nternational Journal of Web Information Systems, 5(1)*, pp. 5-31.

Vipin, M., Sarad, A. V., & Sankar, K. (2008). A multi way tree for token based authentication. *Computer Science and Software Engineering, International Conference , Vol. 3. IEEE*, 1011-1014.