Revista de Sistemas de Informação da FSMA
n. 16 (2015) pp. 52-70

SISTEMAS DE INFORMAÇÃO
http://www.fsma.edu.br/si/sistemas.html

TRILHA PRINCIPAL

Uma Ontologia para Recomendação de Equipes Tecnicamente Qualificadas em Projetos Distribuídos de Software

Larissa Barbosa, Gledson Elias
Centro de Informática – Universidade Federal da Paraíba
larissa@compose.ufpb.br, gledson@ci.ufpb.br

***Abstract* — In Distributed Software Development, the adoption of globally distributed software development teams reduces cost and development time. In order to meet such benefits, it is important to find teams with specific technical background, required for implementing software components and modules that constitute software products. In such a context, it is a key aspect to contrast technical background of development teams against specified technical requirements for implementing the software project, making it possible to select the most skilled teams to develop each software component and module. Hence, this paper proposes, implements and evaluates an application ontology to support selection processes of distributed development teams, which are technically skilled to implement software modules in distributed software projects. As main contribution, experimental results show that the proposed ontology represents and formalizes an extremely complex problem in a systematic and structured way, allowing its direct or customized adoption in selection processes of globally distributed development teams.**

***Index Terms* — Ontology, Distributed Software Development, Knowledge Representation and Inference, Selection Process.**

***Resumo* — No contexto do Desenvolvimento Distribuído de Software, a adoção de equipes de desenvolvimento globalmente distribuídas reduz o custo e o tempo de desenvolvimento. Para alcançar tais benefícios, deve-se buscar equipes com habilidades e conhecimentos específicos, requeridos para o desenvolvimento dos diversos componentes e módulos que compõem os produtos de software. Neste contexto, é de suma importância contrastar as habilidades e conhecimentos das equipes com os requisitos técnicos especificados para implementar o projeto de software, de modo que seja possível identificar aquelas mais qualificadas para desenvolver os componentes e módulos de software. Nesta direção, este artigo propõe, implementa e avalia uma ontologia de aplicação para apoiar processos de seleção de equipes de desenvolvimento distribuídas, tecnicamente qualificadas para implementar módulos de software em projetos distribuídos de software. Como principal contribuição, resultados de experimentos mostram que a ontologia proposta modela e formaliza um problema extremamente complexo de maneira sistemática e estruturada, permitindo sua adoção direta ou customizada em processos de seleção de equipes de desenvolvimento globalmente distribuídas.**

***Index Terms* — Ontologia, Desenvolvimento Distribuído de Software, Representação e Inferência de Conhecimento, Seleção de Equipes.**

I. INTRODUÇÃO

NAS últimas décadas, a Engenharia de Software vem buscando métodos, técnicas, processos e ferramentas para aumentar a produtividade e melhorar a qualidade no desenvolvimento de produtos de software em proporções que tentam acompanhar a rápida evolução da indústria de hardware. Nesta direção, várias abordagens de desenvolvimento de software têm sido propostas pela academia e indústria, merecendo destaque a abordagem de Desenvolvimento Distribuído de Software (DDS), que favorece a adoção de equipes de desenvolvimento globalmente distribuídas para desenvolvimento dos componentes ou módulos dos produtos de software, reduzindo o custo e o tempo de desenvolvimento com a contratação de mão de obra mais barata em diferentes localidades e a possibilidade de rápida formação de equipes, bem como a adoção da estratégia de desenvolvimento com 24 horas contínuas (*follow-the-sun*) [1][2]. Além disso, DDS viabiliza a possibilidade de encontrar mão de obra qualificada e especialistas de domínio nas equipes terceirizadas ou mesmo equipes de subsidiárias ou filiais em empresas com abrangência global [3][4][5].

Consequentemente, para alcançar os benefícios do DDS, devem-se identificar equipes de desenvolvimento com habilidades e conhecimentos técnicos específicos, requeridos para o desenvolvimento dos diversos componentes e módulos de software que compõem os produtos de software. Neste contexto, é de fundamental importância contrastar as habilidades e conhecimentos técnicos das equipes de desenvolvimento candidatas com os requisitos técnicos especificados para implementar o projeto de software, de modo que seja possível identificar aquelas mais qualificadas para desenvolver cada um dos componentes e módulos do produto de software.

Todavia, considerando a dispersão geográfica envolvida nos projetos de desenvolvimento distribuído de software, torna-se bastante difícil para o gerente de projeto realizar a avaliação das habilidades técnicas das equipes de desenvolvimento candidatas, pois, na maioria dos casos, o gerente de projeto não desenvolve atividades presenciais com as equipes, sem contato direto pessoal e conversas de corredor [6]. Portanto,

torna-se difícil para o gerente de projeto obter informações precisas e atualizadas sobre as habilidades e conhecimentos técnicos dos membros destas equipes remotas, uma vez que os mecanismos formais de comunicação baseados em documentos ou repositórios de dados em geral não reagem de forma tão rápida quanto os mecanismos de comunicação informais.

Além disso, mesmo nos casos em que o gerente de projeto detém algum nível de informação sobre as habilidades e conhecimentos técnicos das equipes candidatas, ainda assim, em grandes projetos de software, a tarefa de selecionar equipes é bastante complexa e sujeita a erros de avaliação, pois as diversas equipes candidatas podem adotar diferentes vocabulários ambíguos e métodos incompatíveis para identificar e avaliar suas habilidades e conhecimentos.

Neste cenário, para auxiliar os gerentes de projetos na seleção e alocação de equipes, um *framework* de recomendações [7] foi desenvolvido pelos integrantes do laboratório de pesquisa *Compose*, filiado ao Centro de Informática da Universidade Federal da Paraíba. Como pode ser observado na Figura 1, este *framework* é dividido em três fases de recomendações: *(i)* recomendação de módulos de software; *(ii)* recomendação de equipes qualificadas; e *(iii)* recomendação de alocação de equipes.

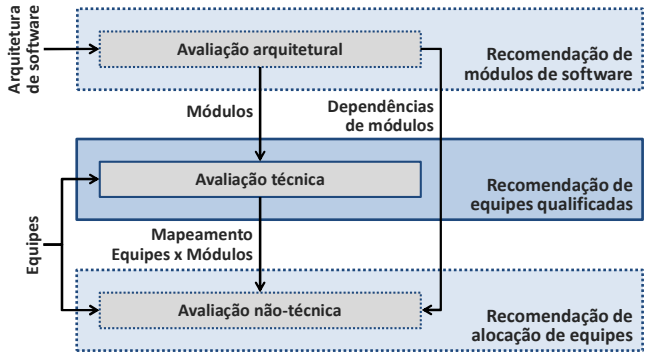

Figura 1 – Framework de recomendações

A primeira fase, denominada *recomendação de módulos de software*, tem por objetivo agrupar componentes em módulos de software, reduzindo as dependências externas entre módulos, e, assim, minimizando de forma significativa os requisitos de comunicação entre as equipes distribuídas que serão responsáveis pela implementação dos módulos [8]. Nesta fase, as decisões de agrupamento de componentes em módulos são guiadas por métricas arquiteturais que quantificam o acoplamento entre componentes de software a partir de suas interfaces providas e requeridas.

Em seguida, tomando como entrada os módulos de software e as equipes de desenvolvimento candidatas, a segunda fase, denominada *recomendação de equipes qualificadas*, tem por objetivo identificar as equipes tecnicamente qualificadas para implementar cada módulo de software. Para tal, esta segunda fase considera as tecnologias requeridas para implementar os módulos de software, bem como as habilidades e conhecimentos técnicos das equipes candidatas em relação a tais tecnologias.

Por fim, com base nas equipes tecnicamente qualificadas, a terceira fase, denominada *recomendação de alocação de equipes*, tem por objetivo identificar possíveis alocações de equipes aos módulos de software de modo a reduzir os requisitos de comunicação entre as equipes durante a implementação dos módulos de software. Nesta fase, atributos não-técnicos das equipes são avaliados, considerando, por exemplo, aspectos culturais, geográficos e temporais.

No contexto deste *framework* de recomendações, este artigo propõe, implementa e avalia a ontologia de aplicação, denominada *OntoDDS*, cujo principal propósito é apoiar processos de seleção de equipes de desenvolvimento distribuídas, tecnicamente qualificadas para implementar módulos de software em projetos distribuídos de software, permitindo sua aplicação direta ou customizada em processos de seleção de equipes de desenvolvimento. Portanto, a *OntoDDS* é parte integrante da segunda fase do *framework* de recomendações, que, como já mencionada, é denominada *recomendação de equipes qualificadas*.

A ontologia aqui proposta tem os seguintes objetivos: *(i)* caracterizar as tecnologias requeridas para implementar cada módulo do projeto de software; *(ii)* caracterizar as habilidades e conhecimentos técnicos das equipes em relação às tecnologias requeridas para implementar os módulos de software; *(iii)* caracterizar políticas de seleção que podem ser adotadas em processos de seleção de equipes em projetos de software; e *(iv)* caracterizar a adequabilidade técnica das equipes para implementar módulos de software de acordo com a política de seleção adotada no projeto de software.

Como principal contribuição, resultados de vários experimentos, conduzidos na instanciação da ontologia proposta em três casos de uso, mostram que a ontologia *OntoDDS* concretiza todos os objetivos a que se propõe, modelando e formalizando, de forma sistemática e estruturada, um problema extremamente complexo que é a seleção de equipes distribuídas tecnicamente qualificadas para implementação de módulos de software em projetos distribuídos de software.

O restante deste artigo está organizado da seguinte forma. A Seção II introduz os principais conceitos de ontologias e delineia a metodologia de desenvolvimento adotada, bem como justifica a escolha da ferramenta de edição e da linguagem de especificação adotadas. A Seção III apresenta e detalha a ontologia proposta, evidenciando seus conceitos e relacionamentos associados à seleção de equipes distribuídas tecnicamente qualificadas. Como forma de observar a usabilidade e aplicabilidade da ontologia proposta, a Seção IV apresenta um caso de uso. Em seguida, a Seção V apresenta e discute os trabalhos relacionados. Por fim, a Seção VI apresenta algumas considerações finais, identifica algumas limitações da *OntoDDS* e delineia alguns trabalhos futuros.

II. METODOLOGIA DE DESENVOLVIMENTO

Como definida em [9], uma ontologia é uma descrição explícita formal dos conceitos de um domínio, as propriedades de cada conceito que descrevem várias características e atributos do conceito, e as restrições sobre tais propriedades.

Os conceitos do domínio são representados por elementos denominados *classes*, que podem adotar a abstração de herança para formular uma hierarquia de classes, na qual cada classe herda propriedades de uma ou mais superclasses. Classes podem ter *instâncias*, que correspondem a objetos individuais no domínio modelado. Uma classe tem várias características, atributos e restrições que são representadas por elementos denominados *propriedades*.

Cada propriedade tem um *domínio* (*domain*) e *valores* (*range*), que podem ser de um determinado tipo e podem ter um conjunto de valores permitidos, variando de tipos simples até instâncias de classes. As propriedades podem ser divididas em *propriedades de objetos* e *propriedades de dados*. As propriedades de objetos relacionam instâncias de uma ou duas classes. Já as propriedades de dados relacionam uma instância de classe com valores de um certo tipo simples, como cadeias de caracteres e números. Cada instância pode ter valores concretos para as propriedades de sua classe.

No que se refere a metodologias de desenvolvimento de ontologias, existem na literatura diversas propostas no intuito de sistematizar a construção e evolução das ontologias. Em [10], Cristani e Cuel apresentam uma avaliação e classificação destas várias propostas, provendo um arcabouço que pode ser usado para auxiliar na escolha da metodologia a ser adotada, considerando as fases e seus artefatos de entrada e de saída. Entretanto, apesar das valiosas contribuições, nenhuma das metodologias propostas na literatura pode ser considerada a metodologia correta, pois ainda não possuem maturidade suficiente, e, portanto, não existe um consenso sobre a melhor, mais completa ou mais adequada metodologia que possa ser amplamente aplicável para quaisquer domínios e necessidades das aplicações.

Neste contexto, em função da sua simplicidade de documentação, facilidade de aplicação, amplo suporte ferramental e foco na construção de ontologias, optamos pela adoção da metodologia *Ontology Development 101* [9], que define um guia bastante simples baseado em uma abordagem iterativa que auxilia projetistas de ontologia, mesmo aqueles não especialistas, na criação de ontologias usando um ferramental de apoio, como, por exemplo, a ferramenta Protégé [11].

A metodologia *Ontology Development 101* foi desenvolvida por pesquisadores envolvidos diretamente no desenvolvimento de ferramentas de especificação de ontologias, como por exemplo a ferramenta Protègè, e, portanto, as diversas fases da metodologia são suportadas de forma plena por tais ferramentas. De forma contrária, embora seja uma metodologia bem mais robusta e sofisticada, a metodologia *Methondology* não tem como requisito a existência de suporte ferramental para automatizar todas as fases, o que pode dificultar sua adoção na prática, uma vez que existe uma combinação de descrições informais e a concretização formal em linguagens de ontologia, que são desenvolvidas em diferentes fases, aumentando a distância entre os modelos do mundo real e os sistemas executáveis.

Além disso, de forma diferente da *Ontology Development 101*, algumas metodologias, como por exemplo a metodologia *Dolce*, não focam diretamente no conjunto de etapas que devem ser seguidas para construir as ontologias. Ao invés disso, focam apenas em aspectos filosóficos ou problemas de expressividade lógica. Em outras metodologias, como por exemplo a metodologia *Diligent*, um aspecto crítico é a necessidade de vários especialistas com diferentes e complementares competências, que devem ser envolvidos na construção colaborativa e distribuída das ontologias.

A metodologia *Ontology Development 101* é baseada em sete fases iterativas, conforme ilustrado na Figura 2. Em geral, metodologias de desenvolvimento de ontologias podem ser aplicadas usando as abordagens *top-down*, *bottom-up* ou uma combinação delas [9]. Nenhuma destas abordagens é inerentemente melhor que as demais uma vez que dependem fortemente da visão pessoal dos projetistas da ontologia sobre o domínio. Apesar disso, no desenvolvimento da *OntoDDS*, optou-se pela abordagem *top-down* pois favorece o melhor controle do nível de detalhes, evitando o detalhamento excessivo presente na abordagem *bottom-up*, que pode levar a maior retrabalho, esforço e inconsistências, além de dificultar a identificação de relacionamentos e similaridades entre os diferentes conceitos [12].

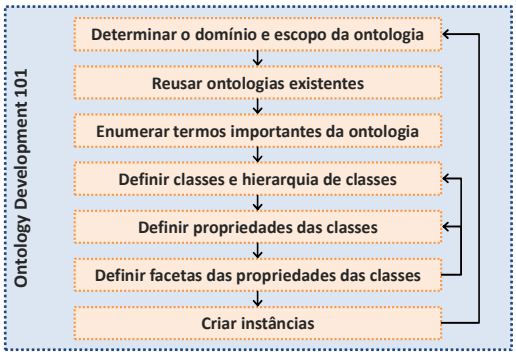

Figura 2 – Fases da metodologia adotada

Determinar o domínio e escopo da ontologia. Na ontologia proposta, o domínio é a *representação de projetos de desenvolvimento distribuído de software*, e, de forma mais específica, o escopo é a *seleção de equipes tecnicamente qualificadas para implementar módulos de software*. Nesta fase, é também importante levantar *questões de competência* que devem ser respondidas pela ontologia aos seus usuários, que, no caso em questão, são os gerentes de projetos. Na *OntoDDS*, identificamos as questões indicadas na Figura 3, que podem ser vistas como requisitos da ontologia.

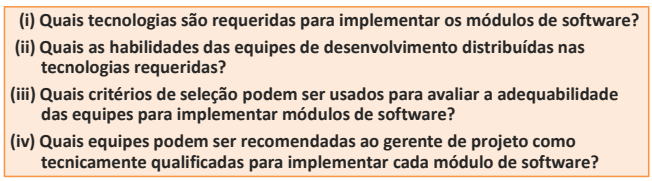

Figura 3 – Questões de competência

Reusar ontologias existentes. Como pode ser visto na Seção V, os trabalhos relacionados evidenciam a inexistência de ontologias para reusar no escopo da *OntoDDS*.

Enumerar termos importantes da ontologia. Dado o domínio e escopo da *OntoDDS*, inicialmente foram enumerados os principais termos, incluindo os conceitos de projeto de software, módulos de software, equipes de desenvolvimento, tecnologias requeridas, políticas de seleção, recomendações de equipes, requisitos tecnológicos, conhecimentos e habilidades das equipes, e adequabilidade técnica das equipes.

Definir classes e hierarquia de classes. Adotou-se uma abordagem *top-down* para especificação das classes, iniciando pelos conceitos de projeto de software, módulos de software que compõem o produto de software a ser desenvolvido, e equipes candidatas a implementar os módulos de software. Em seguida, iniciou-se a modelagem dos requisitos tecnológicos dos módulos, permitindo representar as tecnologias requeridas para implementar cada módulo de software. Posteriormente, foi realizada a modelagem das equipes de desenvolvimento a partir dos seus desenvolvedores. Após a modelagem da composição das equipes, realizou-se a modelagem das habilidades das pessoas nas diversas tecnologias, e, então, a modelagem das habilidades das equipes nestas tecnologias. Na iteração seguinte foi realizada a modelagem do conceito de política de seleção e suas regras de seleção. Por fim, concretizou-se a modelagem das recomendações de equipes a tecnologias e módulos de software.

Definir propriedades das classes. De forma iterativa, à medida que a modelagem *top-down* de classes foi sendo realizada, foi-se acrescentando as propriedades de objetos e dados de cada uma das classes, representando assim aspectos de composição e relacionamento entre instâncias das classes, tendo sempre em mente a representação de conhecimento que tornasse possível responder as questões de competência.

Definir facetas das propriedades das classes. Nesta fase, foi realizado o detalhamento dos tipos de dados, os domínios e valores possíveis, bem como as restrições de cardinalidade das propriedades das classes. Além disso, nesta fase, também foram criados os axiomas para inferência automática de propriedades das instâncias das classes. De forma iterativa, as três fases anteriores produzem como artefato de saída a ontologia *OntoDDS*, que é detalhada na Seção III.

Criar instâncias. Nesta última fase, foram criadas as instâncias das classes, e, posteriormente, as propriedades de objetos e dados associadas às classes. A instanciação da ontologia foi realizada em diferentes projetos de software com o intuito de avaliar se a mesma satisfaz os requisitos definidos em sua construção, mais especificamente, verificar se é possível responder as questões de competência. Vale lembrar que a Seção IV apresenta mais detalhes da instanciação da ontologia *OntoDDS*.

É importante destacar que o desenvolvimento da ontologia foi especificado usando a ferramenta Protégé [11], que provê suporte aos construtores da linguagem de especificação OWL [13], recomendada pelo W3C.

Na literatura, existem diversas linguagens e ferramentas para especificação de ontologias. No que se refere a linguagens, diversas propostas independentes de domínios foram desenvolvidas e disseminadas para representação do conhecimento, incluindo RDF, KIF, DAML+OIL e mais recentemente OWL.

RDF é uma linguagem de representação de dados baseada em XML que permite descrever informações de maneira estruturada, provendo facilidades básicas para definir vocabulários do domínio, mas com limitações de expressividade que dificultam raciocínio automático. Como resposta a tais limitações, KIF é uma linguagem de descrição de conhecimento baseada na lógica descritiva, que foi projetada para facilitar o processamento automático em detrimento da legibilidade para humanos. Além disso, a rica expressividade de KIF torna a complexidade computacional muito alta, inviabilizando sua adoção para raciocínio automático em larga escala.

Buscando melhorar a expressividade e legibilidade, DAML+OIL foi proposta como uma linguagem derivada da combinação de recursos das linguagens DAML e OIL, ambas baseadas em RDF. DAML+OIL pode ser vista como uma fina camada sobre RDF, com semântica formal baseada na lógica descritiva, que incrementa a expressividade de RDF, melhorando assim limitações em relação ao raciocínio automático. Nesta linha de evolução, OWL é considerada uma linguagem sucessora, padronizada pelo W3C, que incorpora lições aprendidas no projeto e aplicação da linguagem DAML+OIL, incluindo um rico conjunto de construtores para representação do conhecimento em diferentes níveis de expressividade. Por representar uma evolução das demais linguagens em termos de legibilidade, interoperabilidade e expressividade, OWL foi escolhida como a linguagem de especificação da *OntoDDS*.

No quesito ferramentas, também existem disponíveis diversas opções para especificação de ontologias, incluindo OilEd, Swoop, OntoEdit e Protégé. Em geral, como funcionalidade básica, essas ferramentas permitem a criação e edição de ontologias. Como funcionalidade diferencial, é desejável que a ferramenta permita a manipulação visual da ontologia com base em uma interface gráfica, abstraindo detalhes da geração da ontologia, que pode ser importada ou exportada em diferentes linguagens de especificação. Além disso, para avaliação e validação das ontologias, torna-se imprescindível o suporte a raciocinadores lógicos.

Considerando esse conjunto de funcionalidades, uma avaliação das ferramentas disponíveis revela que a Protégé

atende praticamente todos os requisitos mencionados anteriormente, que nem sempre são atendidos de forma plena pelas demais opções [14]. Mais especificamente, a ferramenta Protégé atua como uma plataforma para criação, edição, visualização gráfica e validação de ontologias, capaz de importar e exportar especificações OWL e RDF. Como facilidade diferencial, tem suporte a diferentes raciocinadores lógicos, tais como Pellet e FaCT++. Além disso, tem sido adotada por uma grande comunidade de usuários, que colabora na constante evolução de suas funcionalidades. Em função do exposto, a ferramenta Protègè foi selecionada para a modelagem da *OntoDDS*.

III. OntoDDS

No contexto do presente trabalho, um projeto distribuído de software é formado por um conjunto de módulos de software, que podem ser desenvolvidos por um conjunto de equipes candidatas distribuídas globalmente.

Para representar os projetos, seus módulos e as equipes candidatas, conforme ilustrado na Figura 4, a ontologia proposta adota as classes denominadas *Projeto*, *Modulo* e *Equipe*, respectivamente.

Na Figura 4, a propriedade de objeto, denominada *compostoDe*, representa a relação entre projetos e seus módulos, indicando que um projeto de software é composto de diversos módulos de software. Por sua vez, a propriedade de objeto, denominada *temCandidata*, representa a relação entre projetos e equipes candidatas, indicando que um projeto de software tem diversas equipes candidatas a implementar os seus diferentes módulos de software.

Nos mapas conceituais, os conceitos e relacionamentos são representados da seguinte forma: elipses representam as classes; retângulos representam instâncias; setas azuis representam as propriedades de objetos; setas verdes representam as propriedades de dados; e setas pretas representam subtipos.

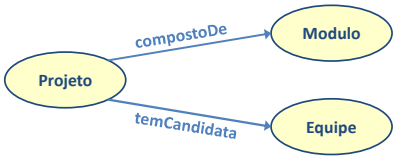

Figura 4 – Projetos, Módulos e Equipes

Um projeto de software é caracterizado ainda por adotar políticas de seleção de equipes, que baseadas em diferentes critérios podem recomendar equipes diferentes para cada um dos módulos a ser implementado.

As políticas de seleção, por sua vez, apresentam pontos de corte, que estabelecem um valor de nível de adequabilidade permitido para que as equipes sejam consideradas adequadas a implementar os módulos de software.

Para representar os projetos, suas políticas e os pontos de corte, conforme mostrado na Figura 5, a ontologia proposta adota as classes denominadas *Projeto*, *Politica* e *PontoDeCorte*.

Na Figura 5, a propriedade de objeto, denominada *adotaPolitica*, representa a relação entre projetos e políticas de seleção de acordo com as necessidades específicas do projeto. Já a propriedade de objeto *temPontoDeCorte* representa a relação entre projetos e seus respectivos pontos de corte, indicando que um projeto de software apresenta pontos de corte distintos para cada uma das possíveis políticas via a propriedade de objeto *naPolitica*.

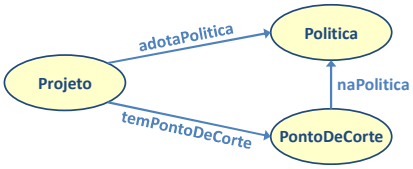

Figura 5 – Projetos, Políticas de Seleção e Pontos de Corte

Considerando que a ontologia *OntoDDS* está voltada a processos de seleção de equipes, dois tipos de recomendações são providos pela ontologia para a seleção de equipes adequadas a implementar módulos de software.

O primeiro tipo de recomendação, representado pela classe denominada *Recomendacao*, é caracterizado por avaliar as equipes candidatas em relação aos módulos de software e suas tecnologias requeridas com o intuito de identificar a adequabilidade das equipes em relação às tecnologias requeridas para implementação destes módulos de software. O segundo tipo de recomendação, representado pela classe *RecomendacaoFinal*, é caracterizado por selecionar as equipes adequadas a implementar o módulo de software baseado no ponto de corte adotado pelo projeto de software.

Para representar os projetos e suas recomendações, conforme mostrado na Figura 6, a ontologia proposta adota as classes denominadas *Projeto*, *Recomendacao* e *RecomendacaoFinal*.

Na Figura 6, as propriedades de objetos, denominadas *temRecomendacao* e *temRecomendacaoFinal*, representam as relações entre projetos e suas respectivas recomendações, indicando que um projeto de software é composto de diversas recomendações.

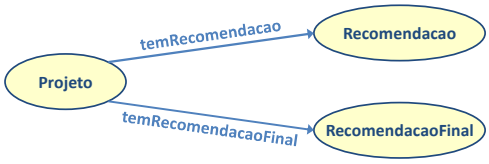

Figura 6 – Projetos e Recomendações

A Figura 7 apresenta a ontologia de forma completa, integrando e expandindo os mapas conceituais ilustrados anteriormente. Observe que, por um lado, as classes *Projeto*, *Modulo*, *Equipe*, *Politica*, *PontoDeCorte*, *Recomendacao* e *RecomendacaoFinal*, já introduzidas nos mapas conceituais das Figuras 4, 5 e 6, agora podem ser vistas de forma integrada na Figura 7. Por outro lado, as classes *Requisito*, *Habilidade*, *Tecnologia*, *Pessoa* e *Regra* representam expansões relacionadas aos mapas conceitos das Figuras 4, 5 e 6, e serão introduzidas e detalhadas nas próximas subseções.

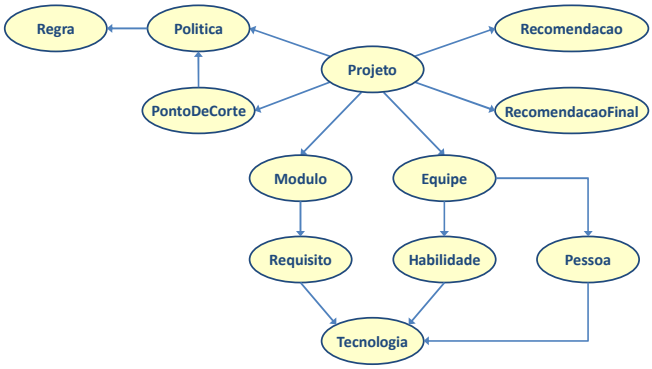

Figura 7 – Mapa Conceitual da Ontologia OntoDDS

É importante salientar que a *OntoDDS* é descrita em OWL, e, assim, todas as classes indicadas na Figura 7 são definidas usando o construtor *owl:Class*. Por exemplo, a Figura 8 ilustra a definição das classes *Projeto*, *Modulo* e *Equipe*. Outro ponto a destacar é que a *OntoDDS* não faz uso do construtor *owl:disjointWith* para criar classes disjuntas, e também não faz uso do construtor *rdfs:subClassOf* para criar hierarquias de classes, exceto quando necessário em restrições de cardinalidade das propriedades de objetos e dados.

```
<owl:Class rdf:ID="Projeto“ />
<owl:Class rdf:ID=“Modulo“ />
<owl:Class rdf:ID=“Equipe“ />
```

Figura 8 – Definição de classe

No que se refere a propriedades de objetos e dados, suas definições adotam os construtores *owl:ObjectProperty* e *owl:DatatypeProperty*, sendo suas restrições de domínios e valores definidas com os construtores *rdfs:domain* e *rdfs:range*.

A Figura 9 mostra a definição da propriedade de objeto *temCandidata*, tendo a classe *Projeto* como domínio e a classe *Equipe* como valor. Outros tipos de construtores também foram usados em algumas propriedades de objetos e dados, mas somente serão indicados nas subseções seguintes.

```
<owl:ObjectProperty rdf:ID="temCandidata">
   <rdfs:domain rdf:resource="#Projeto" />
   <rdfs:range rdf:resource="#Equipe" />
</owl:ObjectProperty>
```

Figura 9 – Definição de propriedade de objeto

As subseções a seguir detalham as propriedades de objetos e dados das classes, bem como suas restrições de cardinalidade e axiomas de inferência. Cabe ressaltar que a estrutura das subseções foi organizada considerando as questões de competência que devem ser respondidas pela ontologia, conforme indicado na Figura 3.

A. Caracterização de Módulos de Software

A avaliação das habilidades e conhecimentos das equipes de desenvolvimento, requeridos para implementar os módulos de software, impõe a necessidade de representar informações dos módulos de software quanto aos aspectos técnicos requeridos para implementá-los. Portanto, a caracterização de módulos de software deve identificar as tecnologias requeridas para implementar cada um dos módulos do projeto de software, bem como os seus respectivos níveis de conhecimentos necessários. Para realizar esta caracterização de módulos, o gerente de projeto deve contar com o apoio do arquiteto e engenheiros de software, responsáveis pela especificação da arquitetura do software.

Na ontologia aqui proposta, o número de níveis e os termos usados para descrever os níveis de conhecimentos necessários podem ser redefinidos pelo gerente de projeto, caso julgar adequado. A proposta inicial define os níveis de conhecimento requeridos usando os seguintes termos: *“Baixo”*, *“Medio”* e *“Alto”*.

A caracterização dos requisitos técnicos deve ser realizada para cada módulo de software a ser implementado no projeto de software. Como pode ser visto na Figura 10, na *OntoDDS*, a caracterização dos requisitos técnicos dos módulos de software é realizada através de indivíduos das classes *Modulo*, *Requisito* e *Tecnologia*, relacionados através das propriedades de objetos denominadas *temRequisito* e *naTecnologia*.

Figura 10 – Módulos, Requisitos e Tecnologias

Por um lado, a propriedade de objeto *temRequisito* associa um dado módulo de software *m* com um determinado requisito *r*, que, através de sua propriedade de dados *termoReq*, sinaliza o nível de conhecimento requerido *n*, cujos valores inicialmente propostos são *“Baixo”*, *“Medio”* e *“Alto”*.

Por outro lado, a propriedade de objeto *naTecnologia* associa o requisito *r* a uma dada tecnologia *t*. Portanto, conjuntamente, tais classes e propriedades representam que o módulo *m* possui o requisito *r* na tecnologia *t*, com nível de conhecimento requerido *n*.

É digno de nota que, na definição de uma classe, também podem ser definidas restrições de cardinalidade para as propriedades de objetos e dados destas classes usando os construtores *owl:minCardinality*, *owl:cardinality* e *owl:maxCardinality*.

Como ilustrado na Figura 10, instâncias da classe *Modulo* possuem no mínimo um requisito associado via a propriedade de objeto *temRequisito*. De forma similar, instâncias da classe *Requisito* possuem no mínimo uma tecnologia associada via a propriedade de objeto *naTecnologia*. Além disso, cada instância da classe *Requisito* possui exatamente um único termo textual associado via a propriedade de dados *termoReq*. A Figura 11 destaca a restrição de cardinalidade da propriedade de objeto *temRequisito* para a classe *Modulo*, indicando que cada módulo deve ter pelo menos um requisito associado.

```
<owl:Class rdf:ID="Modulo">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#temRequisito" />
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
        1
      </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figura 11 – Restrição de cardinalidade

B. Caracterização de Equipes de Desenvolvimento

Depois de caracterizar os requisitos técnicos dos módulos de software, é necessário coletar informações das equipes em relação às tecnologias requeridas para implementá-los. Na *OntoDDS*, a habilidade e conhecimento técnicos que cada equipe possui em cada tecnologia devem ser mensurados e então representados por um número real no intervalo [0, 1].

Para cada tecnologia, três dados devem ser coletados: *anos de experiência*, *número de projetos desenvolvidos* e *número de títulos*. Como pode ser visto em [15][16], em geral, os anos de experiência em uma dada tecnologia, bem como os títulos obtidos nesta tecnologia (incluindo certificações e cursos), podem ser utilizados para avaliar se um indivíduo é um especialista em determinada tecnologia. Em [17], Weiss explica que um fator importante que determina se alguém é especialista em algum domínio é a sua capacidade de discriminação, tido como a capacidade de identificar diferenças sutis em contextos similares. Contudo, essa capacidade de discriminação somente pode ser obtida através da observação de eventos passados, uma vez que é tipicamente um dado empírico.

No entanto, é possível inferir, até certo grau, a capacidade de discriminação de uma pessoa através do número de projetos que tenha participado. Ou seja, uma vez que a capacidade de discriminação é adquirida através da participação em diversos projetos diferentes, mas com contextos similares, quanto mais alguém participa de projetos, maior é a probabilidade de que tenha presenciado domínios com pequenas diferenças, aumentando assim sua capacidade de discriminação [18]. Desta forma, podemos dizer que o dado *número de projetos desenvolvidos* possui alta correlação com a capacidade de discriminação e pode ser usado como seu simulacro.

Desse modo, as informações relativas aos anos de experiência, número de projetos desenvolvidos, e número de títulos, são adotadas na ontologia *OntoDDS* para caracterizar os atributos técnicos das equipes. Vale ressaltar que essas informações são amplamente discutidas na literatura relacionada a diferentes áreas, tais como: identificação de especialistas [16][17], seleção de equipes [18], recomendação de especialistas [19][20], alocação de recursos humanos [21][22][23], alocação de tarefas [24] e recrutamento de profissionais [25][26].

Como pode ser observado na Figura 12, na ontologia proposta, as equipes são representadas por indivíduos das classes *Equipe*, *Pessoa* e *Tecnologia*, relacionados através das propriedades de objetos denominadas *possuiPessoa*, *temExperiencia* e *temProjeto*, bem como pela propriedade de dados denominada *temTitulo*.

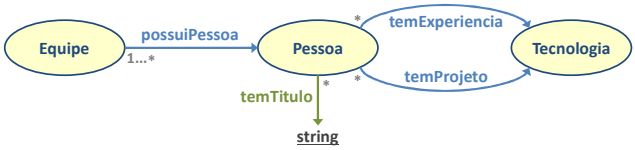

Figura 12 – Equipes, Pessoas e Tecnologias

A propriedade de objeto denominada *possuiPessoa* associa um dado membro *m* a uma determinada equipe *e*, que, através de sua propriedade de dados *temTitulo* e suas propriedades de objetos *temExperiencia* e *temProjeto*, associam o membro *m* a uma dada tecnologia *t*. Portanto, conjuntamente, tais classes e propriedades representam que a equipe possui um ou mais membros com títulos, projetos e experiências nas diferentes tecnologias requeridas no projeto de software.

As propriedades *temExperiencia* e *temProjeto*, ilustradas na Figura 13, possuem subpropriedades que representam, respectivamente, os anos de experiência de um membro *m* em uma dada tecnologia *t*, bem como o número de projetos desenvolvidos pelo membro *m* em uma dada tecnologia *t*.

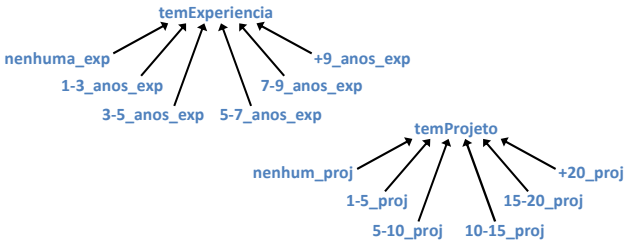

Figura 13 – Subpropriedades para Anos de Experiência e Número de Projetos

É importante frisar que as subpropriedades *temExperiencia* e *temProjeto* são definidas com os construtores *owl:ObjectProperty* e *rdfs:subPropretyOf*, conforme exemplificado na Figura 14 para a subpropriedade *nenhuma_exp*.

```
<owl:ObjectProperty rdf:ID="nenhuma_exp">
  <rdfs:subPropertyOf rdf:resource="#temExperiencia" />
</owl:ObjectProperty>
```

Figura 14 – Definição de subpropriedade

Neste ponto, a partir da caracterização dos conhecimentos e habilidades dos desenvolvedores nas diversas tecnologias, o gerente de projeto pode derivar e mensurar empírica ou matematicamente as habilidades técnicas das equipes em relação às tecnologias requeridas pelos módulos de software. Deve-se destacar que, nos estudos de caso realizados, a abordagem matemática proposta por Santos [15] foi adotada para derivar as habilidades técnicas das equipes. Nesta abordagem matemática, a partir de formulários preenchidos por cada desenvolvedor a respeito dos anos de experiência, número de projetos e títulos em cada tecnologia, as respostas são ponderadas em um conjunto de equações para derivar o nível de conhecimento de cada desenvolvedor em cada tecnologia. Em seguida, a partir do nível de conhecimento dos

membros da equipe em uma dada tecnologia, deriva-se matematicamente o nível de conhecimento da própria equipe naquela tecnologia.

Uma vez derivadas e mensuradas as habilidades técnicas das equipes, agora é necessário representá-las na ontologia. Como pode ser observado na Figura 15, na ontologia *OntoDDS*, as habilidades são representadas por indivíduos das classes *Equipe*, *Habilidade* e *Tecnologia*, relacionados através das propriedades de objetos *temHabilidade* e *naTecnologia*. Por um lado, a propriedade *temHabilidade* associa uma dada equipe *e* a uma ou várias habilidades *h*, que, através de sua propriedade de dados *valorHab*, sinaliza o valor numérico real no intervalo [0, 1] que representa a habilidade da equipe. Por outro lado, a propriedade *naTecnologia* associa uma dada habilidade *h* a uma determinada tecnologia *t*. Portanto, conjuntamente, tais classes e propriedades de objetos e dados representam que a equipe *e* possui habilidade *h* na tecnologia *t*.

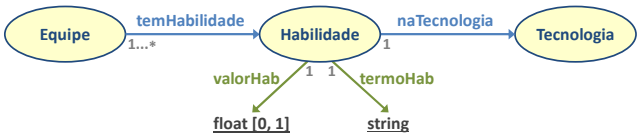

Figura 15 – Equipes, Habilidades e Tecnologias

Embora o grau de habilidade técnica das equipes seja calculado como sendo um número real no intervalo [0, 1], a ontologia também representa a habilidade técnica usando termos textuais como: *"Nenhuma"*, *"Baixa"*, *"Media"* e *"Alta"*. Tais termos passam também a representar a habilidade técnica da equipe em uma dada tecnologia. Como pode ser visto na Figura 15, as habilidades técnicas e seus respectivos termos são representados através das propriedades de dados *valorHab* e *termoHab*, que representam, respectivamente, o valor numérico e o termo textual que caracterizam a habilidade técnica que uma equipe *e* apresenta em alguma tecnologia *t*.

C. Caracterização de Políticas de Seleção

Uma vez identificadas e representadas as tecnologias requeridas para implementar os módulos de software, bem como as habilidades técnicas das equipes em cada uma destas tecnologias, é necessário definir uma política que será responsável por selecionar as equipes que são tecnicamente qualificadas a implementar os módulos de software.

De acordo com as necessidades do projeto de software, políticas diferentes podem ser adotadas, variando assim a maneira como as equipes podem ser selecionadas. Por exemplo, caso um projeto esteja com o cronograma atrasado, selecionar as equipes mais qualificadas para implementar os módulos pode ser uma boa opção, viabilizando assim um tempo menor para codificá-los. Porém, selecionar as equipes mais qualificadas nem sempre é a escolha ideal, visto que selecioná-las pode acarretar desperdício de conhecimento e custo mais elevado, caso o nível de conhecimento técnico requerido pelos módulos de software seja muito inferior em relação às habilidades técnicas das equipes. Sendo assim, é importante selecionar uma política que tente aproximar os níveis de conhecimento requeridos pelos módulos de software e as habilidades técnicas das equipes de forma a evitar o desperdício de conhecimento e reduzir os custos do projeto.

Portanto, a maneira de definir as políticas de seleção é determinada pelas necessidades específicas do projeto e o contexto organizacional em que está inserido o projeto de software. Consequentemente, é fundamental permitir que os gerentes de projetos possam ajustar as políticas adotadas ou criar novas políticas de acordo com as necessidades dos diferentes projetos de software.

Uma política de seleção pode ser compreendida como uma tabela de regras de condições do tipo **SE-ENTÃO**, que relacionam os termos das linhas e colunas, definindo regras que produzem os resultados desejados, representados nas células correspondentes. A Tabela I apresenta um possível exemplo de política de seleção. Perceba que o número de regras de uma política é equivalente ao produto do número de linhas e colunas.

Tabela I – Política de seleção

			Módulo		
			Nível de Conhecimento Requerido		
			Baixo	**Médio**	**Alto**
Equipe	**Nível de Habilidade Técnica**	**Nenhum**	Médio	Baixo	Nenhum
		Baixo	Alto	Médio	Baixo
		Médio	Médio	Alto	Médio
		Alto	Baixo	Médio	Alto

Na Tabela I, pode-se entender a aplicação das regras com o seguinte exemplo: **SE** *Nível de Habilidade Técnica* da equipe é *"Nenhum"* **E** *Nível de Conhecimento Requerido* pelo módulo é *"Médio"* **ENTÃO** *Nível de Adequabilidade* da equipe ao módulo é *"Baixo"*.

A ontologia proposta representa as políticas como indivíduos das classes *Politica* e *Regra*, relacionadas pela propriedade de objeto *temRegra*, como pode ser visto na Figura 16. Observe que uma dada política *p* deve estar associada com um conjunto de regras $\{r_1, r_2, ..., r_n\}$, modelando cada uma das células presentes na tabela que representa a política de seleção em questão.

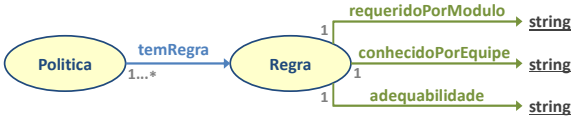

Figura 16 – Políticas e Regras

Por sua vez, as regras são modeladas usando as propriedades de dados denominadas *requeridoPorModulo*, *conhecidoPorEquipe* e *adequabilidade*, que representam, respectivamente, o nível de conhecimento requerido pelo módulo *m* em uma determinada tecnologia *t*, o nível de habilidade técnica ou conhecimento de uma equipe *e* em uma determinada tecnologia *t*, e, consequentemente, o nível de adequabilidade técnica da equipe *e* para implementar o módulo *m* do ponto de vista da tecnologia *t* avaliada.

D.Caracterização das Equipes Tecnicamente Hábeis

Uma vez conhecidas as informações relativas às tecnologias requeridas para implementar os módulos de software, bem como as habilidades das equipes nestas tecnologias, é preciso aplicar a política de seleção para descobrir a adequabilidade técnica da equipe para implementar cada módulo. A ontologia proposta representa a adequabilidade técnica das equipes na forma de recomendações. Como discutido anteriormente, a ontologia contempla dois tipos de recomendações, representadas pelas classes *Recomendacao* e *RecomendacaoFinal*.

Recomendação de Equipes às Tecnologias Requeridas

O mapa conceitual ilustrado na Figura 17 detalha a caracterização da classe *Recomendacao*. Considerando uma dada equipe *e*, um módulo de software *m* e uma tecnologia *t* requerida para implementá-lo, o propósito de uma recomendação é identificar qual regra *r* da política de seleção *p* deve ser adotada.

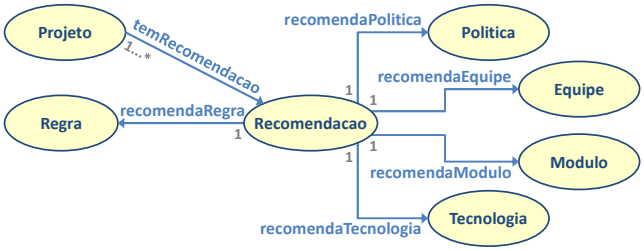

Figura 17 – Recomendação de Equipes às Tecnologias

Para representar essa relação entre a recomendação, a política de seleção adotada, a equipe avaliada, o módulo de software em questão, a tecnologia considerada e a regra de seleção instanciada, cada instância da classe *Recomendacao* possui um conjunto de propriedades de objetos, a saber: *recomendaPolitica*, *recomendaEquipe*, *recomendaModulo*, *recomendaTecnologia* e *recomendaRegra*. Conjuntamente, essas propriedades de objetos representam, respectivamente, a política de seleção adotada no projeto de software, a equipe de desenvolvimento que está sendo avaliada, o módulo de software que está sendo avaliado, a tecnologia requerida para implementá-lo, e, por fim, a regra da política de seleção que deve ser considerada.

Ressalta-se que todas as propriedades de objetos cujo domínio é a classe *Recomendacao* são representadas como propriedades funcionais usando o construtor *owl:FunctionalProperty*, conforme ilustrado na Figura 18 para a propriedade de objeto *recomendaRegra*. Conjuntamente, estas propriedades sinalizam que cada recomendação deve estar associada a uma única política, equipe, módulo, tecnologia e regra.

```
<owl:FunctionalProperty rdf:ID="recomendaRegra">
   <rdfs:domain rdf:resource="#Recomendacao" />
   <rdfs:range rdf:resource="#Regra" />
</owl:FunctionalProperty>
```

Figura 18 – Propriedade funcional

Observe que as seguintes propriedades de objetos *recomendaPolitica*, *recomendaEquipe*, *recomendaModulo* e *recomendaTecnologia* podem ser derivadas a partir das informações já presentes na ontologia, a saber: caracterização dos módulos e suas tecnologias requeridas; caracterização das equipes e seus conhecimentos nessas tecnologias requeridas; e caracterização da política de seleção adotada no projeto de software em questão. Por exemplo, para derivar a propriedade *recomendaEquipe*, basta fazer uma consulta à *OntoDDS* para identificar todas as equipes candidatas associadas ao projeto, e, em seguida, instanciar uma nova recomendação e associá-la a cada equipe candidata via a propriedade *recomendaEquipe*. As demais propriedades são derivadas de forma similar, e, ao final, tem-se uma recomendação para cada combinação de política, equipe, módulo e tecnologia.

Diferentemente, a propriedade de objeto *recomendaRegra* deve ser inferida a partir da política de seleção adotada, considerando a equipe de desenvolvimento avaliada, o módulo de software em questão e a tecnologia associada. Neste ponto, a inferência da regra indicada por cada recomendação é realizada pelo axioma representado na Figura 19. Para inferir a regra da política de seleção a ser acionada, é preciso identificar: *(i)* a política de seleção *po* adotada pelo projeto *pr*; *(ii)* o nível de conhecimento requerido *vreq* pelo módulo *m* na tecnologia *t*; e *(iii)* o nível de habilidade *vh* da equipe *e* na tecnologia *t*.

```
Projeto(?pr), Politica(?po), Regra(?r),
adotaPolitica(?pr, ?po), temRegra(?po, ?r),
Modulo(?m), Requisito(?req), Tecnologia(?t),
temRequisito(?m, ?req), naTecnologia(?req, ?t), termoReq(?req, ?vreq),
Equipe(?e), Habilidade(?h),
temHabilidade(?e, ?h), naTecnologia(?h, ?t), termoHab(?h, ?vh),
Recomendacao(?re), temRecomendacao(?pr, ?re),
recomendaPolitica(?re, ?po), recomendaEquipe(?re, ?e),
recomendaModulo(?re, ?m), recomendaTecnologia(?re, ?t),
conhecidoPorEquipe(?r, ?vh), requeridoPorModulo(?r, ?vreq)
 -> recomendaRegra(?re, ?r)
```

Figura 19 – Axioma para Recomendação de Regra de Seleção

No axioma da Figura 19, observe que a política *po* adotada no projeto *pr* é inferida de forma direta avaliando a propriedade de objeto *adotaPolitica(?pr, ?po)*, modelada no mapa conceitual ilustrado anteriormente na Figura 5.

Para identificar o nível de conhecimento requerido *vreq* pelo módulo *m* na tecnologia *t*, o axioma avalia algumas propriedades de objetos e dados. Inicialmente, as propriedades *temRequisito(?m, ?req)* e *naTecnologia(?req, ?t)* identificam um determinado requisito *req*, representando o fato do módulo *m* requerer a tecnologia *t*.

Em seguida, considerando o requisito *req*, a propriedade de dados *termoReq(?req, ?vreq)* identifica o nível de conhecimento *vreq*, requerido pelo módulo *m* na tecnologia *t*.

Agora, para identificar o nível de habilidade *vh* da equipe *e* na tecnologia *t*, o axioma também considera algumas propriedades de objetos e dados, modeladas no mapa conceitual mostrado anteriormente na Figura 15.

Em primeiro lugar, as propriedades de objetos *temHabilidade(?e, ?h)* e *naTecnologia(?h, ?t)* identificam uma determinada habilidade *h*, que representa o fato da equipe *e* possuir conhecimento na tecnologia *t*. Então, considerando a habilidade *h*, a propriedade de dados *termoHab(?h, ?vh)* identifica o nível de habilidade *vh* da equipe *e* na tecnologia *t*.

Neste ponto, conhecendo a política *po* a ser adotada, o nível de conhecimento *vreq* requerido pelo módulo na tecnologia avaliada e o nível de habilidade *vh* da equipe nesta mesma tecnologia, o axioma pode inferir a regra *r* a ser adotada, avaliando a propriedade de objeto *temRegra(?po, ?r)* e as propriedades de dados *requeridoPorModulo(?r, ?vreq)* e *conhecidoPorEquipe(?r, ?vh)*, modeladas no mapa conceitual ilustrado anteriormente na Figura 16.

Por fim, tendo identificado a regra *r* a ser adotada, o axioma infere a propriedade de objeto *recomendaRegra(?re, ?r)*, representando que a recomendação *re* deve adotar a regra *r*, que, por sua vez, representa em sua propriedade de dados *adequabilidade*, o nível de adequabilidade da habilidade técnica da equipe avaliada em relação ao nível de conhecimento requerido pelo módulo de software na tecnologia em questão.

Recomendação de Equipes aos Módulos de Software

A partir da recomendação das equipes de desenvolvimento a cada tecnologia requerida pelos módulos é possível agora mensurar a adequabilidade das equipes em relação aos módulos de software. Para tal, o gerente de projeto deve adotar alguma forma empírica ou matemática para derivar essa adequabilidade da equipe aos módulos a partir da adequabilidade da equipe a cada tecnologia requerida pelos respectivos módulos.

Esta adequabilidade da equipe aos módulos de software é representada na ontologia pela classe *RecomendacaoFinal*, cujo mapa conceitual é apresentado na Figura 20. Cada recomendação final é caracterizada por possuir as propriedades de objetos denominadas *recomendaPolitica*, *recomendaEquipe*, *recomendaModulo* e as propriedades de dados *valorAdeq*, *termoAdeq* e *adequada*.

As propriedades de objetos representam, respectivamente, a política que foi selecionada no projeto de software, a equipe que está sendo avaliada como possível candidata, e o módulo a ser implementado. Observe que todas as propriedades de objetos cujo domínio é a classe *RecomendacaoFinal* são representadas como propriedades funcionais usando o construtor *owl:FunctionalProperty*, sendo especificadas de forma similar ao exemplo ilustrado anteriormente na Figura 18. Conjuntamente, estas propriedades sinalizam que cada recomendação final deve estar associada a uma única política, equipe e módulo.

Por sua vez, as propriedades de dados *valorAdeq* e *termoAdeq* representam, respectivamente, o valor numérico real no intervalo [0, 1] e o termo textual da adequabilidade da equipe em relação ao módulo a ser implementado, consolidando assim as possíveis equipes candidatas a implementar os vários módulos de um determinado projeto de software.

Figura 20 – Recomendação de Equipes aos Módulos de Software

Observe que propositadamente a propriedade de dados denominada *adequada* não foi discutida, visto que tem relação com a aplicação do ponto de corte, que será visto posteriormente ainda nesta seção.

Depois de calculada a habilidade técnica individual de cada membro da equipe de desenvolvimento e a habilidade técnica da equipe, esses valores numéricos de adequabilidade são convertidos para termos textuais vagos através de axiomas para que seja possível determinar a adequabilidade final das equipes, como pode ser visto nas Figuras 21, 22, 23 e 24.

```
Projeto(?pr), RecomendacaoFinal(?rf),
temRecomendacao(?pr, ?rf),
Politica(?po), Equipe(?e), Modulo(?m),
recomendaPolitica(?rf, ?po), recomendaEquipe(?rf, ?e), recomendaModulo(?rf, ?m),
valorAdeq(?rf, ?v), greaterThanOrEqual(?v, 0.0f), lessThan(?v, 0.15f)
-> termoAdeq(?rf, "Nenhuma")
```

Figura 21 – Axioma da Adequabilidade Final Nenhuma

```
Projeto(?pr), RecomendacaoFinal(?rf),
temRecomendacao(?pr, ?rf),
Politica(?po), Equipe(?e), Modulo(?m),
recomendaPolitica(?rf, ?po), recomendaEquipe(?rf, ?e), recomendaModulo(?rf, ?m),
valorAdeq(?rf, ?v), greaterThanOrEqual(?v, 0.15f), lessThan(?v, 0.45f)
-> termoAdeq(?rf, “Baixa")
```

Figura 22 – Axioma da Adequabilidade Final Baixa

```
Projeto(?pr), RecomendacaoFinal(?rf),
temRecomendacao(?pr, ?rf),
Politica(?po), Equipe(?e), Modulo(?m),
recomendaPolitica(?rf, ?po), recomendaEquipe(?rf, ?e), recomendaModulo(?rf, ?m),
valorAdeq(?rf, ?v), greaterThanOrEqual(?v, 0.45f), lessThan(?v, 0.75f)
-> termoAdeq(?rf, "Media")
```

Figura 23 – Axioma da Adequabilidade Final Média

```
Projeto(?pr), RecomendacaoFinal(?rf),
temRecomendacao(?pr, ?rf),
Politica(?po), Equipe(?e), Modulo(?m),
recomendaPolitica(?rf, ?po), recomendaEquipe(?rf, ?e), recomendaModulo(?rf, ?m),
valorAdeq(?rf, ?v), greaterThanOrEqual(?v, 0.75f), lessThan(?v, 1.00f)
-> termoAdeq(?rf, “Alta")
```

Figura 24 – Axioma da Adequabilidade Final Alta

Observe novamente na Figura 20 que, as propriedades de objetos denominadas *recomendaPolitica*, *recomendaEquipe* e *recomendaModulo* podem ser derivadas a partir das informações já presentes na ontologia, relativas a caracterização dos módulos, equipes e política de seleção adotada. Nesse ponto, a inferência do termo textual da adequabilidade referente ao seu valor numérico pode ser automaticamente realizada pelos axiomas. Para inferir o termo

de adequabilidade é preciso identificar; *(i)* a política *po* adotada pelo projeto *pr*; e *(ii)* o valor numérico da adequabilidade *rf* da equipe *e* no módulo *m*.

Por exemplo, no axioma da Figura 21, observe que o termo textual da adequabilidade é inferido de forma direta avaliando a propriedade *valorAdeq(?rf, ?v)*, modelada no mapa conceitual ilustrado anteriormente na Figura 20.

Para identificar o termo textual da adequabilidade de uma equipe *e* no módulo *m* baseado na política *po*, o axioma avalia algumas propriedades de objetos e dados, modeladas no mapa conceitual da Figura 20. Inicialmente, as propriedades de objetos denominadas *recomendaPolitica(?rf, ?po)*, *recomendaEquipe(?rf, ?e)* e *recomendaModulo(?rf, ?m)* identificam uma recomendação final *rf* associada à uma determinada política *po*, uma determinada equipe *e*, e um determinado módulo *m*. Em seguida, a propriedade de dados *valorAdeq(?rf, ?v)* identifica o valor numérico da adequabilidade *v* da equipe *e* em implementar o módulo *m* utilizando a política *po*. Por fim, considerando o valor numérico da adequabilidade *v*, o axioma infere o termo textual da adequabilidade da recomendação final *rf*. Para tal, o axioma realiza uma análise comparativa entre o valor numérico da adequabilidade *v* e os limites de faixa estipulados para cada um dos termos textuais. Por exemplo, no axioma da Figura 21, o termo textual é *"Nenhuma"*. Para que uma equipe possua esse termo textual de adequabilidade é necessário que o valor numérico de adequabilidade seja maior ou igual a *0* e menor que *0,15*.

Os axiomas dos demais termos de adequabilidade (*"Baixa"*, *"Media"* e *"Alta"*), podem ser vistos nas Figuras 22, 23 e 24, respectivamente, de acordo com os limites definidos para as faixas de valores numéricos dos termos textuais.

Aplicação do Ponto de Corte

Com o intuito de filtrar as equipes, que por ventura possuem um nível de adequabilidade muito baixo, deve ser utilizado um ponto de corte definido pelo gerente de projeto. Essa etapa consiste simplesmente em eliminar aquelas equipes que não atingem o ponto de corte. Para tal, deve-se atualizar as instâncias da classe *RecomendacaoFinal*, atribuindo o valor a sua propriedade de dados denominada *adequada*, conforme ilustrado anteriormente na Figura 20. Cabe destacar que a atualização da propriedade *adequada* é inferida de forma automática via axioma da ontologia, como será detalhada adiante na seção.

Como pode ser observado na Figura 5, na ontologia *OntoDDS*, cada projeto adota uma política de seleção e define o seu próprio ponto de corte, representados por relacionamentos entre indivíduos das classes *Projeto*, *Politica* e *PontoDeCorte*, através das propriedades de objetos *adotaPolitica*, *temPontoDeCorte* e *naPolitica*. Observe que, para ser possível adotar diferentes pontos de cortes para diferentes políticas de seleção, foi necessário definir um relacionamento entre indivíduos das classes *PontoDeCorte* e *Política*, através da propriedade de objeto *naPolitica*, conforme melhor ilustrado na Figura 25.

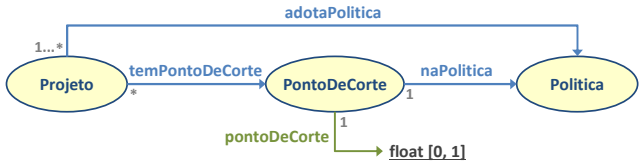

Figura 25 – Detalhamento do Ponto de Corte

Por um lado, a propriedade de objeto *temPontoDeCorte* associa um dado projeto *p* a um determinado ponto de corte *pc*, que, através de sua propriedade de dados *pontoDeCorte*, sinaliza um valor numérico real *n* no intervalo [0, 1], estipulado pelo gerente de projeto para determinar se uma equipe está habilitada ou não a implementar um determinado módulo de software. Por outro lado, a propriedade de objeto *naPolitica* associa o ponto de corte *pc* a uma dada política *po*. Portanto, conjuntamente, tais classes e propriedades representam que o projeto *p* possui o ponto de corte *pc* na política *po*, com valor *n* de ponto de corte.

É importante destacar que a propriedade de objeto *naPolitica* é representada como propriedade funcional usando o construtor *owl:FunctionalProperty*, sendo especificada de forma similar ao exemplo já mostrado na Figura 18.

As propriedades de objetos *temPontoDeCorte* e *naPolitica* e a propriedade de dados *pontoDeCorte* podem ser derivadas diretamente a partir das informações já presentes na ontologia, relativas a caracterização do projeto e da sua política de seleção adotada. Nesse ponto, a inferência do valor da adequabilidade em relação ao ponto de corte é feita pelo axioma representado na Figura 26. Para inferir se o valor de adequabilidade da equipe é aceitável em relação ao ponto de corte é preciso identificar: *(i)* a política *po* adotada pelo projeto *pr*; *(ii)* o valor numérico da adequabilidade *va* da equipe *e* no módulo *m*; e *(iii)* o valor numérico do ponto de corte *vpc* adotado na política *po*.

```
Projeto(?pr), PontoDeCorte(?pc), Politica(?po),
temPontoDeCorte(?pr, ?pc), naPolitica(?pc, ?po), pontoDeCorte(?pc, ?vpc),
RecomendacaoFinal(?rf),
temRecomendacao(?pr, ?rf), recomendaPolitica(?rf, ?po),
valorAdeq(?rf, ?va), greaterThanOrEqual(?va, ?vpc)
-> adequada(?rf, true)
```

Figura 26 – Axioma do Ponto de Corte

Para identificar se o valor numérico da adequabilidade *va* de uma equipe *e* em implementar um módulo *m* é considerada aceitável considerando o ponto de corte *pc*, o axioma avalia algumas propriedades de objetos e dados, modeladas no mapa conceitual visto anteriormente na Figura 25. Inicialmente, as propriedades de objetos *temPontoDeCorte(?pr, ?pc)* e *naPolitica(?pc, ?po)* identificam um determinado ponto de corte *pc* e uma determinada política *po*, que representam o ponto de corte *pc* adotado pela política *po*. Em seguida, considerando a política *po*, a propriedade de dados *pontoDeCorte(?pc, ?vpc)* identifica o valor numérico do ponto de corte *vpc*, requerido pela política *po*.

Neste momento, conhecendo o valor numérico de ponto de corte v*pc* requerido pela política adotada, o axioma pode avaliar se o valor numérico da adequabilidade da equipe *va* é igual ou superior ao ponto de corte *vpc*. Para tal, o axioma avalia a propriedade de dados *valorAdeq(?rf, ?va)*, e, por fim, infere a propriedade de dados *adequada(?rf, true)*, representando que a recomendação *rf* é considerada adequada ao ponto de corte.

IV. Caso de Uso

Como forma de avaliar a usabilidade e aplicabilidade da ontologia proposta, foram desenvolvidos três casos de uso com base no projeto de duas linhas de produtos de software. Detalhes dos estudos de caso podem ser encontrados em [27].

Os dois primeiros casos foram desenvolvidos utilizando uma linha de produtos de software hipotética na área de comércio eletrônico (e-commerce), documentada em [28]. Esses dois primeiros casos de uso foram organizados em duas iterações de desenvolvimento, contemplando as fases de engenharia de domínio e engenharia de aplicação da linha de produtos. Em seguida, outro caso de uso foi desenvolvido tendo como base um projeto real de uma linha de produtos de middleware para dispositivos móveis, denominada *Multi-MOM* [29], cuja instanciação será brevemente ilustrada a seguir ao longo desta seção.

Na condução dos diversos casos de uso, primeiramente, a ontologia *OntoDDS* foi completamente especificada e validada na ferramenta Protégé [11], contemplando classes, propriedades de objetos e dados, restrições e axiomas. Em seguida, cada caso de uso foi também instanciado e validado na ferramenta Protégé, contemplando indivíduos dos diversos elementos da ontologia *OntoDDS*.

A ferramenta Protégé provê suporte à linguagem de especificação de ontologia OWL [13], recomendada pelo W3C. Utilizando a ferramenta foi possível criar e modelar classes, propriedades de objetos e dados, restrições e axiomas, bem como criar instâncias das classes. Além disso, a ferramenta Protégé permite realizar consultas e visualizar os resultados automaticamente produzidos pelos diversos axiomas da ontologia.

A. Caracterização dos Módulos

O *Multi-MOM* [29] consiste numa linha de produtos de middleware para computação móvel que está essencialmente focada na funcionalidade de comunicação. A partir de sua arquitetura de componentes, apresentada na Figura 27, foram definidos cinco módulos de software, segundo a fase de *recomendação de módulos de software* [30] do framework de seleção e alocação de equipes [7], brevemente delineado na Seção I deste artigo.

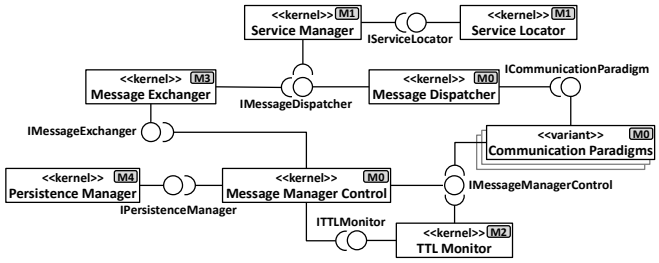

Figura 27 – Arquitetura do Multi-MOM

Como pode ser observado na Figura 27, foram identificados cinco módulos distintos, indicados nos pequenos retângulos rotulados com os termos *M0*, *M1*, *M2*, *M3* e *M4*. A caracterização das tecnologias requeridas pelos módulos foi realizada pelo próprio arquiteto de software que concebeu e projetou o middleware *Multi-MOM*. A título de exemplificação, a Figura 28 ilustra a instanciação da ontologia *OntoDDS* para caracterização das tecnologias requeridas para implementar o módulo *M1*.

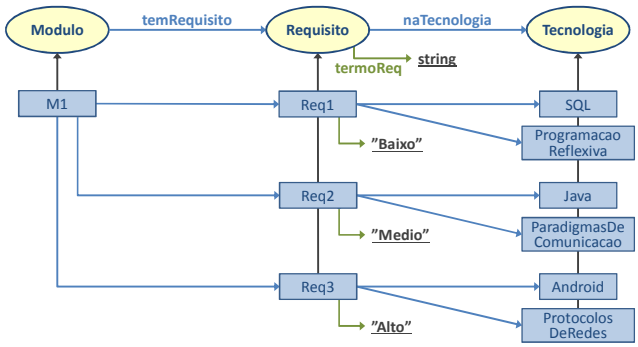

Figura 28 – Caracterização do Módulo M1

Como pode ser observado na Figura 28, o módulo *M1* requer as tecnologias *SQL* e *ProgramacaoReflexiva* com nível de conhecimento *"Baixo"*. Já para as tecnologias *Java* e *ParadigmasDeComunicacao*, este módulo requer nível de conhecimento *"Medio"*. Por fim, considerando as tecnologias *Android* e *ProtocolosDeRedes*, o nível de conhecimento requerido pelo módulo é *"Alto"*.

B. Caracterização das Equipes

Considerando a dificuldade de encontrar equipes de desenvolvimento reais para os casos de uso, a formação das equipes de desenvolvimento foi realizada com base em desenvolvedores do mercado local e estudantes dos cursos de ciências da computação da região, que responderam a um questionário, contemplando todas as tecnologias requeridas para os casos de uso, de acordo com os módulos a serem implementados nas respectivas linhas de produtos de software. A aplicação desse questionário foi realizada de forma *online*, resultando em um conjunto de 179 desenvolvedores participantes. Os formulários adotados e as respectivas respostas dos desenvolvedores podem ser encontrados em [18].

Em seguida, os questionários respondidos foram utilizados para realizar a caracterização das habilidades e conhecimentos

técnicos dos 179 desenvolvedores em cada tecnologia requerida pelos módulos. A título de exemplificação, a Figura 29 ilustra a instanciação da ontologia proposta para caracterização das habilidades e conhecimentos técnicos do desenvolvedor *D1* que pertence a equipe *E1* na tecnologia *Java*. Como pode ser observado, o desenvolvedor *D1* tem cinco a sete anos de experiência em *Java*, já participou de até cinco projetos que adota *Java*, e possui os certificados *SCJA* e *SCJP*.

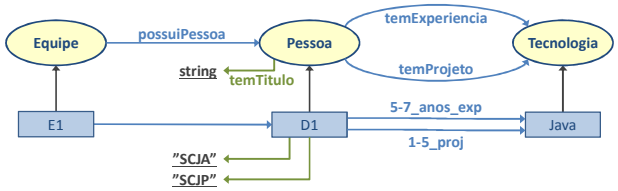

Figura 29 – Caracterização do Desenvolvedor D1 na Tecnologia Java

A partir do conjunto de 179 desenvolvedores, foi definido um conjunto de 22 equipes, com diferentes tamanhos entre 2 e 18 membros, sendo cada membro escolhido de forma aleatória até completar todas as equipes. A composição final das equipes ficou da seguinte forma: 1 equipe com 2 membros; 3 equipes com 3 membros; 5 equipes com 5 membros; 4 equipes com 8 membros; 2 equipes com 9 membros; 3 equipes com 10 membros; 3 equipes com 15 membros; e 1 equipe com 18 membros.

Em seguida, a partir das habilidades e conhecimentos técnicos de cada desenvolvedor, é possível caracterizar as habilidades técnicas das suas respectivas equipes para cada tecnologia requerida pelos diversos módulos de software. Como exemplo, a Figura 30 ilustra a instanciação da *OntoDDS* para caracterização das habilidades da equipe *E1* na tecnologia *Java*. Como pode ser visto na Figura 30, considerando as habilidades e conhecimentos técnicos de seus desenvolvedores, a equipe *E1* tem nível de habilidade técnica com valor *0,61* na tecnologia *Java*, que, de acordo com as faixas de valores adotadas, caracteriza habilidade mediana, representada pelo termo textual *"Medio"*.

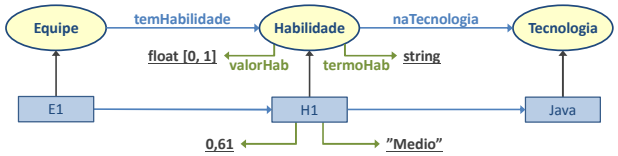

Figura 30 – Caracterização da Equipe E1 na Tecnologia Java

C. Caracterização das Políticas de Seleção

Na instanciação da ontologia proposta, foram inicialmente especificadas quatro políticas de seleção distintas, criadas a partir de observações e análises apresentadas em outros trabalhos da literatura [21][22][23][25]. As quatro políticas propostas são:

a) **Política de qualificação equivalente:** seleciona equipes que possuem as habilidades técnicas mais próximas das requeridas para implementar os módulos de software;
b) **Política de equipes mais hábeis:** seleciona equipes que possuem as maiores habilidades técnicas, de forma independente do nível de conhecimento requerido pelos módulos de software;
c) **Política de qualificação mínima:** seleciona equipes que possuem as habilidades técnicas mínimas requeridas para implementar os módulos de software;
d) **Política de provisão de treinamento:** seleciona equipes que possuem habilidades técnicas abaixo do requerido para implementar os módulos de software.

Por exemplo, considerando a política de seleção de qualificação equivalente, anteriormente definida na Tabela I, a instanciação da regra representada pelo cruzamento da terceira linha com a segunda coluna da Tabela I, aqui denominada *R8*, é apresentada na Figura 31. De acordo com esta política, a regra instanciada é interpretada da seguinte forma: **SE** *Nível de Habilidade Técnica* é *"Medio"* **E** *Nível de Conhecimento Requerido* é *"Medio"* **ENTÃO** *Nível de Adequabilidade* é *"Alto"*. Vale ressaltar que, neste caso de uso, as 12 regras da Tabela I foram numeradas de *R1* até *R12*, percorrendo as linhas da esquerda para a direita e de cima para baixo.

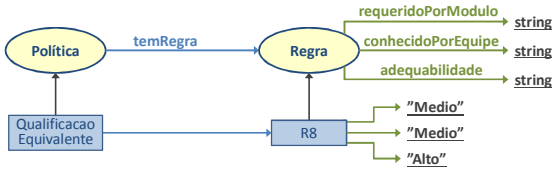

Figura 31 – Caracterização da Regra R8 da Política de Seleção

Quanto ao ponto de corte, como pode ser observado na Tabela II, diferentes pontos de corte foram utilizados para cada política de seleção adotada. A partir dos casos de uso realizados, percebeu-se que os valores de adequabilidade das equipes variavam de acordo com a política de seleção adotada, fato que já era esperado visto que as diferentes políticas atribuem diferentes adequabilidades às equipes. No entanto, em uma análise experimental, onde cada caso de uso foi avaliado em relação a cada política de seleção, percebeu-se a tendência da política de provisão de treinamento apresentar valores de adequabilidade maiores que todas as demais políticas. Por sua vez, observou-se que a política de qualificação mínima tende a apresentar valores maiores que a política de qualificação equivalente e a política de equipes mais hábeis. Por fim, também se detectou que a política de qualificação equivalente tende a gerar valores maiores que a política de equipes mais hábeis. Em função dessa evidência empírica, optamos por adotar diferentes pontos de corte para cada uma das políticas de seleção consideradas nos casos de uso.

Tabela II – Pontos de Corte

Política de Seleção	Ponto de Corte
Qualificação Equivalente	0,60
Equipes mais Hábeis	0,55
Qualificação Mínima	0,70
Provisão de Treinamento	0,75

Para exemplificar a instanciação dos pontos de corte na ontologia, a Figura 32 mostra a representação do ponto de corte com valor *0,60*, adotado na política de seleção *QualificacaoEquivalente* utilizada no projeto *Multi-MOM*.

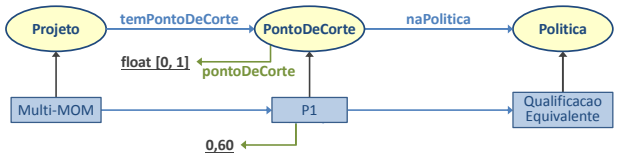

Figura 32 – Ponto de Corte da Política de Qualificação Equivalente

D.Avaliação da Adequabilidade das Equipes

Neste ponto, considerando as tecnologias requeridas pelos módulos, as habilidades técnicas das equipes nas tecnologias, e a política de seleção adotada no projeto, pode-se inferir a adequabilidade técnica das equipes em relação às tecnologias requeridas pelos módulos de acordo com a política de seleção. A Figura 33 mostra um exemplo de inferência da adequabilidade técnica da equipe *E1* na tecnologia *Java* requerida no módulo *M1* de acordo com a política de seleção *QualificacaoEquivalente*.

Como pode ser observado na Figura 33, a referida adequabilidade é definida pela aplicação da regra *R8*, cuja instanciação na ontologia proposta foi exemplificada na Figura 31. Vale ressaltar que a inferência da regra de seleção adotada é realizada pelo axioma da Figura 19.

Neste ponto, é possível mensurar empírica ou matematicamente a adequabilidade das equipes aos módulos de software. Para tal, nestes casos de uso, a abordagem matemática proposta em [15] foi adotada para derivar a adequabilidade das equipes aos módulos a partir da adequabilidade das equipes a cada tecnologia requerida pelos módulos de software.

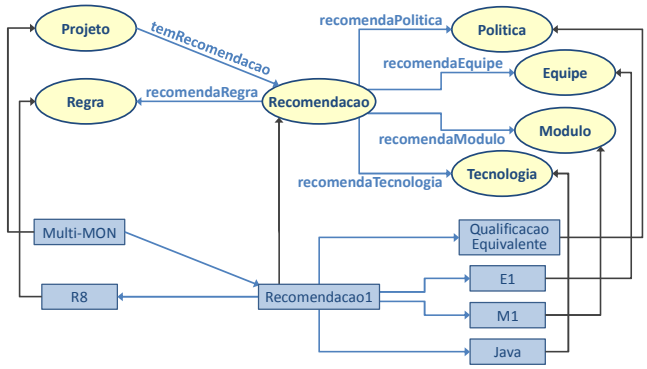

Figura 33 – Adequabilidade técnica da Equipe E1 em Java no Módulo M1

Como exemplo, a Figura 34 ilustra a recomendação final da equipe *E1* ao módulo *M1*, cujo valor numérico de adequabilidade é *0,64*. Em complemento, aplicando os axiomas das Figuras 21, 22, 23 e 24, é possível inferir os termos textuais que representam a adequabilidade. Na Figura 34, o termo textual da adequabilidade é "*Medio*".

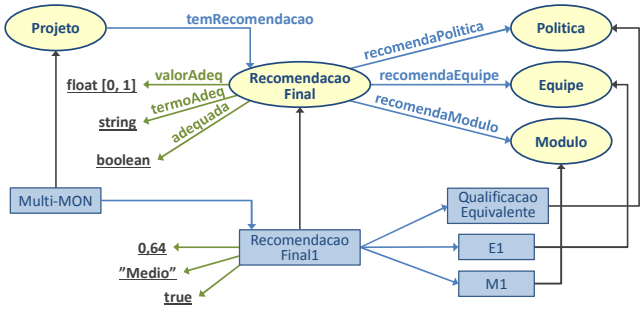

Figura 34 – Recomendação da Equipe E1 ao Módulo M1

Por fim, baseado no axioma da Figura 26, pode-se inferir as equipes tecnicamente habilitadas em cada módulo de software a partir da avaliação do ponto de corte definido no projeto de software para a política de seleção em questão, consolidando assim as possíveis equipes candidatas a implementar os módulos do projeto de software. Observe que, na propriedade de dados *adequada*, a Figura 34 já mostra o resultado da inferência da adequabilidade da equipe *E1* ao módulo *M1* na política *QualificacaoEquivalente*.

No caso de uso do projeto da linha de produto *Multi-MOM*, após a aplicação do ponto de corte, dentre as 22 equipes candidatas, foram recomendadas 5, 11, 12, 21 e 19 equipes para os módulos *M0*, *M1*, *M2*, *M3* e *M4*, respectivamente.

Considerando as quatro políticas de seleção definidas e os três casos de uso desenvolvidos para avaliar a usabilidade e aplicabilidade da ontologia proposta, cada caso de uso resultou em quatro recomendações de adequabilidade das equipes aos módulos, sendo uma recomendação para cada política de seleção. Portanto, considerando todos os casos de uso, foram gerados ao final 12 recomendações distintas, cujos detalhes podem ser encontrados em [27].

V.Trabalhos Relacionados

Nesta seção são apresentadas e discutidas três abordagens identificadas na literatura, que estão em certo grau relacionadas ao presente trabalho. As três abordagens consideradas são: *(i) OntoDiSEN* [31] – uma ontologia para compartilhamento de informações em projetos de DDS; *(ii) Proposta de Burbeck* [32] – uma ontologia para estabelecimento de contratos eletrônicos; e *(iii) ICARE* [19] – sistema de recomendação de especialistas que adota uma ontologia para caracterizar usuários e especialistas.

É importante mencionar a dificuldade de encontrar propostas na literatura que sejam diretamente relacionadas à seleção de equipes distribuídas tecnicamente qualificadas. Consequentemente, apesar das abordagens identificadas não terem o propósito específico de apoiar à seleção de equipes em projetos distribuídos de software, os trabalhos discutidos apresentam aspectos relacionados à *OntoDDS* em função da adoção de ontologias para representação de informações associadas aos ambientes de DDS, à definição de critérios para contratação de serviços eletrônicos, e à caracterização de usuários e especialistas.

Para guiar a comparação das abordagens avaliadas com a abordagem *OntoDDS* aqui proposta, sintetizamos as principais características relacionadas na Tabela III. A seguir, apresentamos uma breve descrição de cada trabalho relacionado, juntamente com uma discussão comparativa em relação à *OntoDDS*.

A *OntoDiSEN* [31] é uma ontologia de aplicação, cujo objetivo é descrever os conceitos e os elementos contextuais, que são representados, armazenados e compartilhados por uma ferramenta de disseminação de informações, permitindo a comunicação e cooperação entre indivíduos de equipes geograficamente dispersas, e, desta forma, aumentando a percepção dos membros das equipes sobre ações relacionadas aos artefatos de colaboração produzidos. Neste cenário de colaboração, a ontologia *OntoDiSEN* é o elemento responsável pela representação das informações contextuais, promovendo a disseminação do contexto de maneira uniforme e padronizada entre as equipes distribuídas.

Na *OntoDiSEN*, as informações relativas às habilidades e conhecimentos requeridos são associadas às fases do processo. Portanto, a *OntoDiSEN* adota uma entidade alvo de granularidade mais grossa que a ontologia *OntoDDS*, que associa tais informações a módulos de software, cuja granularidade é mais fina.

No que concerne a caracterização das habilidades e conhecimentos requeridos pelas fases do processo, a *OntoDiSEN* permite a instanciação de múltiplos atributos não valorados, por exemplo, representando os requisitos em diferentes tecnologias, ferramentas ou processos, mas não quantificando tais necessidades. De forma similar, a *OntoDDS* também permite a instanciação de múltiplos atributos para caracterizar habilidades e conhecimentos requeridos pelos módulos de software, mas, diferentemente, tais necessidades são quantificadas em diferentes níveis.

Seguindo uma abordagem similar, na *OntoDiSEN*, os conhecimentos e habilidades dos usuários são também caracterizados pela instanciação de múltiplos atributos não valorados, enquanto que a *OntoDDS* caracteriza tais informações instanciando múltiplos atributos para caracterização das habilidades e conhecimentos dos desenvolvedores de software, porém valorados, ou seja, quantificados em diferentes níveis. Além disso, ao invés de caracterizar apenas indivíduos (usuários no caso da *OntoDiSEN*), a *OntoDDS* também caracteriza as habilidades e conhecimentos das equipes de desenvolvimento a partir dos seus respectivos desenvolvedores.

A *OntoDiSEN* não define qualquer mecanismo ou método para coletar os conhecimentos e habilidades que os usuários possuem e que as fases do processo demandam, deixando a cargo do gerente de projeto definir como obtê-los. De forma diferente, a *OntoDDS* estabelece a adoção das tabelas de implementação para representar as tecnologias requeridas para implementar os módulos de software, e, no caso dos desenvolvedores, baseia-se em formulários para a coleta de informações relativas aos anos de experiência, número de projetos desenvolvidos e número de títulos.

A segunda abordagem avaliada é a proposta de Burbeck [32], que apresenta uma ontologia de apoio à contratação de serviços, cujo propósito é representar informações a serem usadas durante o estabelecimento de contratos eletrônicos, tanto de forma genérica, quanto especificamente no contexto de DDS. A ontologia proposta por Burbeck permite a especificação de requisitos não funcionais de QoS (*Quality-of-Service*) relacionados aos serviços eletrônicos, bem como informações necessárias aos clientes e fornecedores para que possa ser possível avaliar se os requisitos são atendidos. Portanto, a ontologia pode ser aplicada para apoiar o estabelecimento de contratos eletrônicos, podendo ser usada para representar os conceitos e relacionamentos envolvidos na negociação entre as empresas participantes em uma possível contratação no processo de desenvolvimento de software.

Embora os propósitos da *OntoDDS* e da proposta de Burbeck sejam diferentes, podemos indiretamente correlacioná-las. Enquanto a *OntoDDS* é usada para selecionar equipes de desenvolvimento para implementar módulos de software, a proposta de Burbeck é usada para contratar fornecedores para execução de serviços eletrônicos. Consequentemente, pode-se considerar que as entidades alvo possuem granularidade similar, pois módulos de software e serviços eletrônicos podem ser considerados correlatos.

Considerando a caracterização dos atributos de QoS dos serviços eletrônicos, a proposta de Burbeck permite a instanciação de múltiplos atributos valorados, que são quantificados em diferentes níveis. Portanto, neste aspecto, pode ser considera similar a *OntoDDS*, que também permite a instanciação de múltiplos atributos valorados para caracterizar diferentes níveis de habilidades e conhecimentos requeridos para implementar os módulos de software.

De modo igualmente comparável, na proposta de Burbeck, os atributos de QoS assegurados pelos fornecedores são caracterizados pela instanciação de múltiplos atributos valorados, que são quantificados em diferentes níveis. De modo análogo, agora do ponto de vista da *OntoDDS*, os conhecimentos e habilidades dos desenvolvedores e equipes são também instanciados em múltiplos atributos valorados, representando seus níveis de conhecimento nas respectivas tecnologias, métodos, processos ou domínios de aplicação.

Tabela III – Comparativo dos Trabalhos Relacionados

Proposta	Entidade Alvo	Caracterização da Entidade Alvo	Recurso Alvo		Caracterização do Recurso Alvo	Captura de Dados de Entrada	Política de Seleção
OntoDiSEN	Fases do Processo	Múltiplos atributos não valorados	Usuários		Múltiplos atributos não valorados de usuários	–	–
Burbeck	Serviços	Múltiplos atributos valorados	Fornecedores		Múltiplos atributos valorados de fornecedores	–	Implícito Imutável
ICARE	Usuários	Múltiplas palavras-chave não valoradas	Especialistas		Múltiplos atributos não valorados de especialistas	–	Implícito Imutável
OntoDDS	Módulos de Software	Múltiplos atributos valorados	Equipes		Múltiplos atributos valorados de desenvolvedores e equipes	Tabelas Formulários	Explícito Configurável

De forma similar à *OntoDiSEN*, a proposta de Burbeck também não define qualquer mecanismo ou método para coletar os atributos de QoS que os serviços demandam e que os fornecedores asseguram, deixando a cargo do gerente de projeto definir a forma de obtê-los. Diferentemente, como já indicado, a *OntoDDS* adota tabelas de implementação e formulários para representar as tecnologias requeridas e os conhecimentos e habilidades dos desenvolvedores e equipes.

A terceira abordagem avaliada é o *ICARE* (*Intelligent Context Awareness for Recommending Experts)* [19], um sistema de recomendação de especialistas em determinados assuntos de interesse, caracterizados por palavras-chave fornecidas pelos usuários e levando em consideração o contexto atual de usuários e especialistas. Note que, para caracterizar os especialistas e usuários, bem como as informações contextuais e relacionamentos entre palavras-chave e assuntos de interesse, o *ICARE* adota uma ontologia de domínio que permite inferir recomendações personalizadas de diferentes especialistas para diferentes usuários em variados assuntos de interesse.

No *ICARE*, como pode ser percebido, o propósito não é a seleção de equipes. No entanto, podemos indiretamente correlacionar as habilidades e conhecimentos requeridos pelos módulos de software na *OntoDDS* com os assuntos e domínios de interesse dos usuários no *ICARE*. Portanto, em ambos os casos, são necessidades de suas entidades alvo, ou seja, módulos de software na *OntoDDS* e usuários no *ICARE*.

Em relação às necessidades dos usuários em termos de assuntos e domínios de interesse, o *ICARE* permite aos usuários informarem múltiplas palavras-chave não valoradas, não adotando assim qualquer tipo de quantificação da importância de cada palavra-chave informada. Portanto, em relação à *OntoDDS*, o *ICARE* pode ser considerado menos sofisticado ou realista, uma vez que a *OntoDDS* permite a instanciação de múltiplos atributos valorados, que caracterizam os diferentes níveis de habilidades e conhecimentos requeridos pelos módulos de software.

De forma análoga, o *ICARE* também caracteriza os conhecimentos técnicos dos especialistas instanciando múltiplos atributos não valorados, e, portanto, não considera diferenças nos níveis de conhecimento destes especialistas. Já na *OntoDDS*, os conhecimentos e habilidades dos desenvolvedores e suas equipes são caracterizados instanciando múltiplos atributos valorados, considerando diferenças nos níveis de conhecimento destes desenvolvedores e suas respectivas equipes.

O *ICARE* compartilha da mesma deficiência da *OntoDiSEN* e da proposta de Burbeck no sentido de não definir qualquer mecanismo ou método para capturar os conhecimentos técnicos dos especialistas, deixando a cargo do gerente de projeto definir a forma de obtê-los. De modo diferente, a *OntoDDS* estabelece tabelas de implementação e formulários para representar as tecnologias requeridas e os conhecimentos e habilidades dos desenvolvedores.

Por fim, percebe-se que a *OntoDiSEN* não necessita e assim não representa o conceito de política de seleção, pois não tem como propósito a seleção de qualquer tipo de recurso alvo, mas sim o compartilhamento de informações contextuais em projetos de DDS. De forma diferente, tanto na contratação de fornecedores de Burbeck quanto na recomendação de especialistas do *ICARE*, faz-se necessária a adoção do conceito de política de seleção. No entanto, em ambas as propostas, a política de seleção fica implícita e imutável no modelo ontológico, provavelmente representada através de regras para o motor de inferência. Diferentemente, de forma bem mais explícita e configurável, na *OntoDDS*, diferentes políticas de seleção podem ser definidas na ontologia pelo gerente de projeto, cujas regras de seleção são automática e transparentemente tratadas pelos axiomas da ontologia no motor de inferência.

VI. CONSIDERAÇÕES FINAIS

Neste artigo foi apresentada uma ontologia de aplicação para apoiar processos de seleção de equipes distribuídas tecnicamente qualificadas para implementação de módulos de software em projetos de software.

A ontologia proposta é parte integrante de um framework de recomendações [7] que tem como objetivo principal apoiar os gerentes de projetos no processo de alocação de equipes distribuídas às tarefas de implementação de módulos de software em projetos de linhas de produtos de software.

A ontologia *OntoDDS* possui quatro blocos de conceitos inter-relacionados que permitem realizar a caracterização dos seguintes elementos de um projeto de software: *(i)* tecnologias requeridas para implementar os módulos de software; *(ii)* habilidades e conhecimentos técnicos das equipes de desenvolvimento nas diferentes tecnologias requeridas pelos módulos de software; *(iii)* políticas de seleção; e *(iv)* adequabilidade técnica das equipes de desenvolvimento aos módulos de software.

Observe que os quatro blocos de conceitos representam a concretização de cada uma das questões de competência da ontologia proposta, mencionadas Figura 3 da Seção II. Portanto, como pode ser percebido, a ontologia *OntoDDS* concretiza todos os objetivos a que se propunha.

Como principal contribuição do trabalho, adotando a estratégia *dividir para conquistar*, a ontologia proposta modela e formaliza, de maneira sistemática e estruturada, um problema extremamente complexo que é a seleção de equipes distribuídas tecnicamente qualificadas para implementação de módulos de software em projetos distribuídos de software. A estrutura geral da *OntoDDS* pode ser percebida no mapa conceitual da Figura 7, onde toda a problemática é modelada usando apenas 12 classes, inter-relacionadas por 23 propriedades de objetos e 11 propriedades de dados, que, quando instanciadas, podem sistematizar o processo de tomada de decisão do gerente de projeto, principalmente quando relativizado ou observado sob a ótica da alta complexidade do problema, principalmente quando tratado de maneira *ad-hoc*. Além disso, a ontologia proposta facilita a comunicação entre o gerente de projeto e os membros das equipes, uma vez que define um vocabulário comum entre os envolvidos no processo de seleção.

A instanciação da *OntoDDS* para um projeto distribuído de software pode requerer um esforço considerável para a criação das instâncias e suas propriedades de objetos e dados, sendo, portanto, sujeita a erros e ocasionando desperdício de tempo. Por exemplo, considerando o caso de uso do *Multi-MOM*, apresentado na Seção IV, cujo projeto arquitetural foi agrupado em 5 módulos de software, com requisitos em 7 tecnologias distintas, e avaliado em relação à adequabilidade de 22 equipes em 4 políticas de seleção, o número de instâncias de classes (3.267), propriedades de objetos (19.150) e propriedades de dados (1.982) é impressionante, requerendo um esforço notável para manipulá-las na ferramenta Protégé.

No entanto, como facilidade adicional, deve-se observar que os seis axiomas da *OntoDDS* permitem a inferência automática de uma propriedade de objeto e duas propriedades de dados. No caso de uso do *Multi-MOM*, os axiomas inferem 2.376 propriedades de objetos e 880 propriedades de dados, representando uma cobertura de aproximadamente 12,5% das propriedades de objetos e 44,4% das propriedades de dados.

É também necessário mencionar que, apesar do elevado número de instâncias e suas respectivas propriedades de objetos e dados, a ontologia proposta tem alto potencial de reuso em diferentes cenários. Por exemplo, uma vez que um dado projeto de software é instanciado com seus módulos de software, tecnologias requeridas, equipes candidatas e política de seleção adotada, a avaliação com base em outra política de seleção pode facilmente reusar todas as instâncias e propriedades de objetos e dados relacionadas aos módulos de software, tecnologias requeridas e equipes candidatas. De forma mais significativa, se imaginarmos uma base de dados de projetos de software anteriores, incluindo a maioria das tecnologias comumente requeridas para implementar módulos de software, um grande conjunto de equipes candidatas, e as principais políticas de seleção adotadas, a avaliação de um novo projeto de software pode também reusar todas as instâncias e propriedades de objetos e dados relacionadas às tecnologias, equipes e políticas de seleção.

Mesmo considerando o alto potencial de reuso da ontologia proposta, ainda assim, a instanciação manual requer um esforço considerável para identificar e manipular as instâncias e suas propriedades de objetos e dados que podem ser reusadas e aquelas que precisam ser criadas. No sentido de reduzir este esforço, a instanciação da ontologia poderia ser realizada de forma programática, explorando a API da ferramenta Protégé, evitando erros e poupando tempo. Apenas como ilustração do impacto extremamente positivo da abordagem programática, à primeira vista, considerando uma aplicação onde o usuário sinaliza em um conjunto de tabelas os módulos de software, as tecnologias requeridas para implementá-los, as equipes candidatas e seus membros, praticamente 100% das instâncias de classes e suas propriedades de objetos e dados poderiam ser criadas de forma automática e transparente.

Considerando os pontos discutidos, podemos sintetizar os seguintes benefícios diretos ou contribuições adicionais da adoção da *OntoDDS* no problema de seleção de equipes:

i. Melhor entendimento do domínio do problema;
ii. Facilidade de comunicação entre os envolvidos no processo de seleção de equipes, uma vez que define um vocabulário comum;
iii. Formalização de conceitos e relacionamentos associados ao processo de seleção de equipes;
iv. Possibilidade de realizar inferências sobre o domínio, quando apoiado por ferramental com suporte a motores de inferência;

v. Reuso de informações associadas a módulos, equipes, tecnologias e políticas de seleção em diferentes cenários e projetos de software.

Apesar dos relevantes benefícios e contribuições, algumas limitações foram observadas nos casos de uso e mencionadas na discussão anterior, a saber:

i. Não adoção da API da ferramenta Protégé para manipular a base de dados ontológica de forma programática em Java;
ii. Casos de uso não realizados com equipes de desenvolvimento reais da indústria de software;
iii. Adoção de termos incertos da lógica *fuzzy* sem modelagem do processo de tomada de decisão.

Em primeiro lugar, sem o uso da API da ferramenta Protégé, a criação das classes, propriedades de objetos e dados, e as instâncias foi realizada de forma puramente manual, sendo, portanto, sujeita a erros e ocasionando desperdício de tempo.

Em segundo lugar, as equipes de desenvolvimento consideradas nos casos de uso são equipes fictícias baseadas em desenvolvedores da indústria de software local e alunos de cursos da área de computação, o que não proporciona uma validação real da ontologia proposta, embora torne possível a avaliação da sua usabilidade e aplicabilidade da ontologia. Por fim, embora os requisitos, habilidades e adequabilidades sejam representados usando termos incertos da lógica *fuzzy*, o processo de tomada de decisão modelado nas políticas de seleção não contempla integralmente as etapas de *fuzzificação*, inferência e *defuzzificação* da lógica *fuzzy*, que são baseadas nos conjuntos e funções de pertinência *fuzzy*.

Tendo em vista as limitações percebidas, alguns trabalhos futuros foram identificados. Dentre eles, podemos citar:

i. Adoção da API da ferramenta Protégé com o intuito de manipular a base de dados ontológica de forma programática, por exemplo, via o framework Jena [33];
ii. Validação da ontologia em um projeto real com equipes de desenvolvimento distribuídas globalmente;
iii. Avaliação da complexidade lógica da ontologia;
iv. Modelagem integral do processo de tomada de decisão da lógica *fuzzy* nas políticas de seleção.

AGRADECIMENTO

Este trabalho foi apoiado pelo Instituto Nacional de Ciência e Tecnologia para Engenharia de Software (INES – www.ines.org.br) e financiado pelo CNPq, processo 573964/2008-4.

REFERÊNCIAS

[1] R. Martignoni, Global sourcing of software development: a review of tools and services, 4th International Conference on Global Software Engineering (ICGSE 2009), pp. 303-308, 2009.

[2] E. Carmel, Y. Dubinsky, and A. Espinosa, Follow the sun software development: new perspectives, conceptual foundation, and exploratory field study, 42nd Hawaii International Conference on System Sciences (HICSS 2009), 2009.

[3] J. Herbsleb, and D. Moitra, Global software development, IEEE Software, pp. 16-20, March-April 2001.

[4] P. Ovaska, M. Rossi, and P. Marttiin, Architecture as a coordination tool in multi-site software development, Software Process Improvement and Practice, vol. 8, no. 4, pp. 233–247, October-December 2003.

[5] R. Prikladnicki, J. L. N. Audy, and R. Evaristo, Global software development in practice: lessons learned, Software Process Improvement and Practice, vol. 8, no. 4, pp. 267–281, October-December 2003.

[6] A. Mockus, and J. Herbsleb, Challenges of global software development, 7th International Symposium on Software Metrics, 2001.

[7] T. A. B. Pereira, V. S. Santos, B. L. Ribeiro, and G. Elias, A recommendation framework for allocating global software teams in software product line projects, 2nd International Workshop on Recommendation Systems for Software Engineering, 2010.

[8] T. Burity, and G. Elias, A quantitative, evidence-based approach for recommending software modules, 30th Annual ACM Symposium on Applied Computing (SAC 2015), pp. 1449-1456, 2015.

[9] N. F. Noy, and D. L. McGuinness, Ontology Development 101: a guide to creating your first ontology, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, March 2001.

[10] M. Cristani, and R. Cuel, A survey on ontology creation methodologies, International Journal on Semantic Web and Information Systems, vol. 1, no. 2, pp. 48-68, April-June 2005.

[11] Protégé, disponível em http://protege.stanford.edu, último acesso em 08/09/2015.

[12] M. Uschold and M. Gruninger, Ontologies: principles, methods and applications, Knowledge Engineering Review, vol. 11, no. 2, June 1996.

[13] OWL Web Ontology Language Guide, disponível em http://www.w3.org/TR/owl-guide, último acesso em 08/09/2015.

[14] S. C. Buraga, L. Cojocaru, and O. C. Nichifor, Survey on web ontology editing tools, Transactions on Automatic Control and Computer Science, pp. 1-6, 2006.

[15] V. Santos, Uma abordagem para recomendação de módulos para projetos de desenvolvimento distribuído de linhas de produto de software. V Workshop de Desenvolvimento Distribuído de Software (WDDS 2010), 2010.

[16] J. Shanteau, D. J. Weissb, R. P. Thomasa, and J. C. Poundsc, Performance-based assessment of expertise: how to decide if someone is an expert or not. European Journal of Operational Research, v. 136, pp. 253-263, 2002.

[17] D. J. Weiss, J. Shanteau, P. Harries, People who judge people, Journal of Behavioral Decision Making, vol. 19, p. 441-454, 2006.

[18] V. S. Santos, Uma abordagem para seleção de equipes tecnicamente qualificadas para implementação de projetos de software, Dissertação de Mestrado, Universidade Federal da Paraíba, 2014.

[19] H. Petry, ICARE: um sistema de recomendação de especialistas sensível a contexto, Dissertação de Mestrado, Universidade Federal de Pernambuco, 2007.

[20] H. Kagdi, M. Hammad, J. I. Maletic, Who can help me with this source code change?, IEEE International Conference on Software Maintenance, Beijing, 2008.

[21] A. S. Barreto, Apoio à alocação de recursos humanos em projetos de software: uma abordagem baseada em satisfação de restrições, Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, COPPE, 2005.

[22] M. A. Silva, WebAPSEE-Planner: auxílio à alocação de pessoas em projetos de software através de políticas, Dissertação de Mestrado, Universidade Federal do Pará, 2007.

[23] D. A. Callegari, L. Foliatti, R. M. Bastos, MRES – ferramenta para seleção de recursos para tarefas de projetos de software via abordagem difusa e multicritérios, Simpósio Brasileiro de Engenharia de Software (SBES), Sessão de Ferramentas, 2009.

[24] J. Duggan, J. Byrne, G. Lyons, A task allocation optimizer for software construction, IEEE Computer Society Press, vol. 21, 2004.

[25] J. Collofello *et al.*, A system dynamics software process simulator for staffing policies decision support, 31st Annual Hawaii International Conference on System Sciences, 1998.

[26] N. A. Ruskova, Decision support system for human resources appraisal and selection, 1st International IEEE Symposium "Intelligent Systems", 2002.

[27] L. Barbosa, Uma abordagem ontológica para recomendação de equipes qualificadas em projetos de software, Dissertação de Mestrado, Universidade Federal da Paraíba, 2014.

[28] H. Gomaa, Designing software product lines with UML: from use cases to pattern-based software architectures, Addison Wesley, Object-Oriented Technology Series, 2004.

[29] Y. M. Bezerra, Multi-MOM: um middleware multi-paradigma, extensível e orientado a mensagens para computação móvel, Dissertação de Mestrado, Universidade Federal da Paraíba, 2010.

[30] T. Burity, Uma abordagem para recomendação de módulos para projetos de desenvolvimento distribuído de linhas de produto de software, Dissertação de Mestrado, Universidade Federal da Paraíba, 2011.

[31] A. P. Chaves, DiSEN-CSE: um modelo baseado em context-awareness para disseminação de informações em um ambiente de desenvolvimento distribuído de software, Dissertação de Mestrado, Universidade Estadual de Maringá, 2011.

[32] S. Burbeck, The tao of e-business services: the evolution of web applications into service-oriented components with web services, IBM Software Group, 2000.

[33] Apache Jena, disponível em http://jena.apache.org, último acesso em 06/04/2015.