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

An ontology for the recommendation of technically qualified teams in distributed software projects

Larissa Barbosa, Gledson Elias
Informatics Center– Federal University of 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.**

I. INTRODUCTION

IN the last decades, Software Engineering has been searching for methods, techniques, processes and tools to increase productivity and improve the quality of product development proportionally to the fast evolution of the hardware industry. With this goal in mind, several software development approaches have been proposed by academia and industry. An emphasis is due for the Distributed Software Development (DSD) approach, which favors the adoption of globally distributed software teams to develop components or modules of the software products, decreasing the development cost and/or time due to the hiring of cheaper workers in different locations, allowing for a fast team formation as well as the adoption of the 24 hours development strategy (*follow-the-sun*) [1][2]. Besides, DSD makes it viable to find qualified workers and domain specialists in third-party teams or even in subsidiary or branch teams in companies with global presence [3][4][5].

Consequently, in order to get the benefits of DSD, we should identify development teams with specific skills and technical knowledge required for the development of several software components and modules that compose the software product. In this context, it is of utmost importance to compare the skills and technical knowledge of the candidate development teams against the technical requirements specified for the implementation of the software project in order to become possible to identify those that are more qualified to develop each one of the software components and modules.

Nevertheless, considering the geographic dispersion involved in distributed software development projects, it may become difficult for the project manager to perform the evaluation of the technical skills of the candidate development teams, because, in most cases, the project manager does not develop any full-body activities with the teams, having neither direct personal contact nor drinking fountain talks [6]. Hence, it may be difficult for the project manager to get precise and up to date information on the skills and technical knowledge of the members of the remotest teams, given that the formal communication mechanisms based on documents or data repository do not react in such a fast way as the informal communication mechanisms.

Besides, even in the cases where the project manager knows a bit about the skills and abilities of the candidate teams, in large software projects the task of selecting teams may still be very complex and subject to evaluation errors, because different candidate teams may adopt different and ambiguous vocabulary and incompatible methods to evaluate and identify their abilities and knowledge.

In this scenario, in order to help project managers select and allocate teams, a recommendation framework [7] was developed by the members of the *Compose* research lab, affiliated to the Informatics Center at Federal University of Paraíba. As can be seen in Figure 1, this framework is made of three recommendation phases: *(i)* recommending software modules; *(ii)* recommending qualified teams; and *(iii)* recommending team allocation.

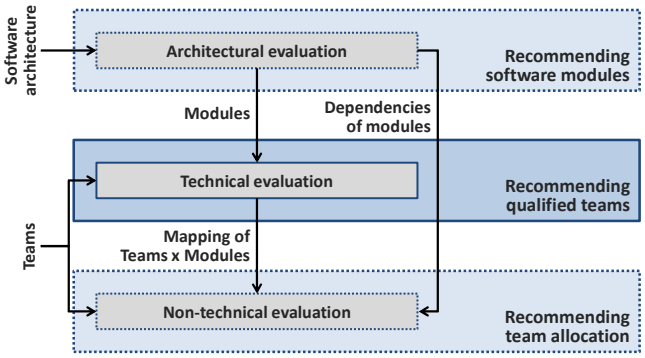

Figure 1 – Recommendation framework

The first phase, called *recommending software modules*, intends to cluster components into software modules, reducing dependencies between modules and hence minimizing the communication requirements between the distributed teams that will be responsible for their implementation [8]. In this phase, the component clustering decisions are made based on architectural metrics that quantify the coupling between software components based on their provided and required interfaces.

Next, using as input the software modules and the candidate development teams, the second phase, called *recommending qualified teams*, intends to identify the technically qualified teams to implement each software module. In order to do so, this phase considers the technologies required to implement the software modules, as well as the skills and technical knowledge of each of the candidate teams related to those technologies.

Finally, based on the technically qualified teams, the third phase, called *recommending team allocation*, intends to identify possible allocations of software modules to teams in a way to minimize inter-team communication requirements during the software modules implementation. In this phase, non-technical attributes of each team are evaluated, considering, for instance, cultural, geographic and temporal aspects.

In the context of this recommendation framework, this paper proposes, implements and evaluates the application ontology called *OntoDDS*, whose main goal is to support the selection of distributed development teams that are technically qualified to implement software modules in distributed software projects, allowing for direct or customized application in development teams selection processes. Hence, the *OntoDDS* ontology is part of the second phase of the recommendation framework which, as mentioned, is called *recommending qualified teams*.

The ontology here proposed has the following goals: *(i)* characterize the required technologies to implement each software project module; *(ii)* characterize the skills and technical knowledge of the teams according to the technologies that are required to implement the software modules; *(iii)* characterize selection policies that may be used in software project team selection processes; and *(iv)* characterize the suitability of the teams to implement software modules according to the selection policy adopted in the software project.

As main contribution, the results of several experiments performed instantiating the proposed ontology in three use cases, show that the *OntoDDS* achieves all the goals it is proposed to meet, modeling and formalizing in a systematic and structured way, an extremely complex problem, that is the selection of technically qualified distributed teams to implement software modules in distributed software projects.

The rest of this paper is organized as follows. Section II introduces the main concepts of ontology and defines the development methodology adopted, and also justifies the editing tool and specification language choice. Section III presents and details the proposed ontology, explaining all its concepts and relationships associated to the selection of technically qualified distributed teams. In order to observe the usability and applicability of the proposed ontology, Section IV presents a use case. Next, Section V presents and discusses the related works. Finally, Section VI presents some final considerations, identifies limitations of *OntoDDS* and presents some future works.

II. Development Methodology

As defined in [9], ontology is an explicit formal description of the concepts in a domain, the properties of each concept that describe its characteristics and attributes together with the restrictions over those properties.

The domain concepts are represented by elements called *classes*, which can adopt the inheritance abstraction to create a class hierarchy, in which each class inherits properties from one or more superclasses. Classes may have *instances*, which correspond to individual objects in the modeled domain. A class has many characteristics, attributes and restrictions that are represented by elements called *properties*.

Each property has a *domain* and a *range*, which can belong to a specific type and may have a set of permitted values, from simple types to class instances. Properties may be divided into *object properties* and *datatype properties*. The object properties are related to instances of one or two classes. On the other hand, the datatype properties create a relationship between a class instance and values of a simple type, such as strings and numbers. Each instance may have concrete values to its class properties.

On the subject of ontology development methodology, there are in the literature several proposals to systematize the construction and evolution of ontologies. Cristani and Cuel [10] present an evaluation and classification of several of those

proposals, providing a framework that can be used to help choose the adopted methodology, taking into consideration the phases and their input and output artifacts. Nevertheless, in spite of the valuable contributions, none of the methodologies presented in the literature can be considered as the correct one. Indeed, none of them has enough maturity and, therefore, there is no consensus on which is the best, the more complete or more adequate that can be widely applicable to any domain and application need.

In this context, given the simplicity of its documentation, its ease of use, the large number of tools and its focus in the construction of ontologies, we chose the methodology *Ontology Development 101* [9], which defines a very simple guide based on an iterative approach that helps ontology designers, even those who are not experts, to create an ontology using a specification tool such as Protégé [11].

The methodology *Ontology Development 101* was developed by researchers that work directly with the development of ontology specification tools, such as the Protègè tool, for instance. Hence, the many phases of the methodology are fully supported by such tools. On the other hand, even though it is more robust and sophisticated, the methodology *Methontology* does not have as a requirement tool support to automatize all its phases, which may make it more difficult to adopt it in a real case, given that there is a combination of informal descriptions and formal concretization in ontology languages that are developed in different phases, increasing the distance between the real world models and the executable systems.

Besides, differently from *Ontology Development 101*, some methodologies, such as *Dolce*, do not focus in the set of steps that must be followed to build the ontology. Instead, they focus only on the philosophical aspects or on the logic expressivity issues. In other methodologies, such as *Diligent*, a critical aspect is the need for several experts with different and complementary competences, which must be involved in the collaborative and distributed ontology development.

The methodology *Ontology Development 101* is based on seven iterative phases, as shown in Figure 2. In general, ontology development methodologies may be applied using the *top-down* or *bottom-up* approaches, or even a combination of them [9]. None of those approaches is inherently better that the others and the judgment depends on the personal view of the ontology designers on the domain. In spite of that, in the development of *OntoDDS*, we chose the *top-down* approach because it favors the control of the detail level, avoiding the excessive details present in the *bottom-up* approach, which can take to more rework, effort and inconsistencies, besides making it more difficult to identify relationships and similarities among different concepts [12].

The phases of the adopted methodology can be explained as follows:

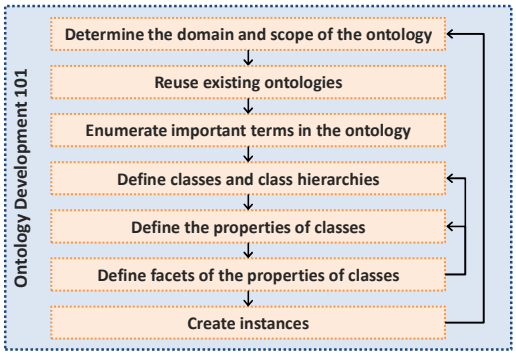

Figure 2 – Phases of the adopted methodology

Determine the domain and scope of the ontology. In the proposed ontology, the domain is the *representation of distributed software development projects*, and, in a more specific way, the scope is *the selection of technically qualified teams to implement software modules*. In this phase, it is also important to raise *competency questions*, which must be answered by the ontology to its users, which, in the case at hand, are the project managers. In *OntoDDS*, we identify the questions shown in Figure 3, which may be seen as the ontology requirements.

(i) Which technologies are required to implement the software modules?
(ii) Which are the distributed development team skills in the required technologies?
(iii) Which selection criteria may be used to identify the suitability of the teams for the software modules implementation?
(iv) Which teams can be recommended to the project managager as technically qualified to implement each software module?

Figura 3 – Competency questions

Reuse existing ontologies. As can be seen in Section V, the related works show evidence of the lack of ontologies that can be reused in the scope of *OntoDDS*.

Enumerate important terms in the ontology. Given the domain and scope of *OntoDDS*, initially we enumerated the main terms, including the concepts of software project, software modules, development teams, required technologies, selection policies, team recommendation, technological requirements, team knowledge and skills, and team technical suitability.

Define classes and class hierarchies. We adopted a *top-down* approach for the specification of classes, beginning with the concepts of software project, software modules that make the software product under development and candidate teams for the implementation of software modules. Next, we began the modeling of the technological requirements for the modules, allowing us to represent the required technologies to implement each software module. Afterwards, we modeled the development teams based on their developers. After modeling team composition, we modeled the personal skill in several technologies and then, the team skills in those technologies. In the next iteration we modeled the concept of selection policies with their selection rules. Finally, we modeled the team

recommendation to technologies and software modules.

Define the properties of classes. We performed the *top-down* modeling of the classes in an iterative way, adding the object and datatype properties for each of the classes and thus representing composition and relationship aspects among instances of the classes and keeping in mind that the knowledge representation should make it possible to answer the competency questions.

Define facets of the properties of classes. In this phase we detailed the data types, the domains and possible values, as well as the cardinality restrictions for each class property. Besides, in this phase we also created the axioms for automatic inference of properties for instances of the classes. In an iterative way, the three previous phases create as output artifact the ontology *OntoDDS*, which is detailed in Section III.

Create instances. In this final phase, we create the instances of the classes and afterwards, the object and datatype properties associated to the classes. The ontology instantiation was performed in different software projects in order to evaluate if it satisfies the requirements defined during its construction and, more specifically, if it is possible to answer the competency questions. It is worth remembering that Section IV presents details on the instantiation of the ontology *OntoDDS*.

It is important to point out that the ontology development was specified using the Protégé tool [11], which supports the constructors of the OWL [13] specification language, recommended by W3C.

In the literature there are several languages and tools for the specification of ontologies. Several domain independent languages were proposed and disseminated to represent knowledge, including RDF, KIF, DAML+OIL and, more recently, OWL.

RDF is a data representation language based on XML that allows us to describe information in a structured way, providing basic facilities to define domain vocabularies but with expressivity limitations that make it difficult to execute automatic reasoning. As an answer to those limitations, KIF is a knowledge description language that is based on descriptive logic and was designed to facilitate automatic reasoning, even if it would become less legible to humans. Besides, the rich expressivity of KIF makes its computational complexity very high, turning it not viable to large scale automatic reasoning.

Seeking to improve expressivity and legibility, DAML+OILD was proposed as a language derived from the combination of the resources available at the languages DAML and OIL, both based on RDF. DAML+OILD can be seen as a thin layer over RDF, with formal semantics based on descriptive logic, which increments the expressivity of RDF, improving on its limitations concerning automatic reasoning. In this line of evolution, OWL is considered as a successor language, standardized by the W3C, which incorporates the lessons learned in the design and application of the language DAML+OILD, including a rich set of constructors for the representation of knowledge in different expressivity levels. Because it represents an evolution of the other languages in terms of legibility, interoperability and expressivity, OWL was chosen as the specification language for *OntoDDS*.

There are several tools to specify ontologies, including OilEd, Swoop, OntoEdit and Protégé. In general, as basic functionality, these tools allow for the creation and editing of ontologies. As a differential, it is desirable that a tool allows the visual manipulation of the ontology with a graphic interface, abstracting details of the generation of the ontology, which may be imported or exported into different specification languages. Besides, for the evaluation and validation of ontologies, it is required the support of logical reasoning.

Considering this set of functionalities, an evaluation of available tools comes to the conclusion that Protégé fulfills practically all the requirements mentioned above, which are not always fulfilled completely by the other options [14]. More specifically, the Protégé tool is a platform for the creation, editing, graphic visualization and ontology validation, being able to import and export specifications in OWL and RDF. As a differential, it also supports different logical engines, such as Pellet and FaCT++. Besides, it has been adopted by a large user community that cooperates in the constant evolution of its functionalities. Due to the exposed, we chose the Protégé tool to model *OntoDDS*.

III. ONTODDS

In the context of this paper, a distributed software project is composed of a set of software modules that can be developed by a set of candidate teams that are globally distributed.

In order to represent the software project, its modules and the candidate teams, as illustrated in Figure 4, the proposed ontology adopts the classes called *Projeto* (Project), *Modulo* (Module) and *Equipe* (Team), respectively[1].

In Figure 4, the object property called *compostoDe* (madeOf) represents the relationship between projects and their modules, indicating that a software project is made of several software modules. The object property *temCandidata* (hasCandidate) represents the relationship between projects and candidate teams, indicating that a software project has many associated candidate teams that can implement its many software modules.

In the conceptual maps, the concepts and relationships are represented as follows: ellipsis represent the classes, rectangles represent instances, blue arrows represent object properties, green arrows represent datatype properties and black arrows represent subtypes.

[1] In order to keep the ontology source code working as in its original, the class names and all OWL code will be kept in Portuguese.

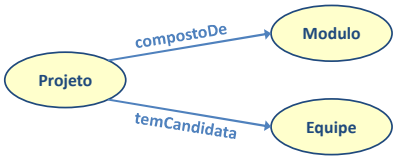

Figure 4 – Projects, Modules and Teams

A software project is characterized also by the adoption of team selection policies, which based on different criteria may recommend different teams for each one of the modules that need to be implemented.

The selection policies present cut points that establish a suitability level that is the minimum for the teams to be considered adequate to implement the software modules.

In order to represent the projects, their policies and cut points, as shown in Figure 5, the proposed ontology has the classes called *Projeto* (Project), *Politica* (Policy) and *PontoDeCorte* (CutPoint).

In Figure 5, the object property called *adotaPolitica* (adoptsPolicy) represents the relationship among projects and selection policies, according to the specific project needs. On the other hand, the object property *temPontoDeCorte* (hasCutPoint) represents the relationship between projects and their cut points, indicating that a software project has different cut points for each of its possible policies using the object property *naPolitica* (inPolicy).

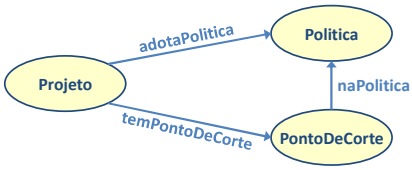

Figure 5 – Projects, Selection Policies and Cut Points

Considering that the goal of the *OntoDDS* ontology is the team selection process, two types of recommendations are offered inside the ontology for the selection of teams that are capable of implementing software modules.

The first type of recommendation is represented by the class called *Recomendacao* (Recommendation) and is characterized by the evaluation of candidate teams according to the software modules and the technologies that are required to implement them. Its goal is to identify the suitability of each team in relation to the technologies and skills required to implement the software modules at hand.

The second type of recommendation is represented by the class *RecomendacaoFinal* (FinalRecommendation) and is characterized by the selection of the teams that can implement each software module based on the cut point adopted by the software project.

In order to represent the projects and their recommendations, as shown in Figure 6, the proposed ontology adopts the classes called *Projeto* (Project), *Recomendacao* (Recommendation) and *RecomendacaoFinal* (FinalRecommendation).

In Figure 6, the object properties called *temRecomendacao* (hasRecommendation) and *temRecomendacaoFinal* (hasFinalRecommendation) represent the relationships between projects and their respective recommendations, showing that a software project may have several different recommendations.

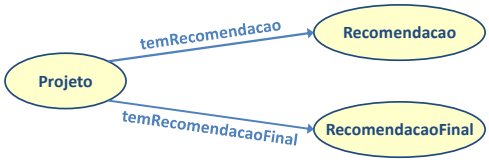

Figure 6 – Projects and Recommendations

Figure 7 presents the full ontology integrating and expanding the conceptual maps of the previous figures. It should be noted that the classes *Projeto*, *Modulo*, *Equipe*, *Politica*, *PontoDeCorte*, *Recomendacao* and *RecomendacaoFinal*, already introduced by Figures 4, 5, and 6 now can be seen in an integrated way in Figure 7. On the other hand, the classes *Requisito*, *Habilidade*, *Tecnologia*, *Pessoa* and *Regra* represent expansions related to the conceptual maps of Figures 4, 5, and 6, and will be presented in detail in the next subsections.

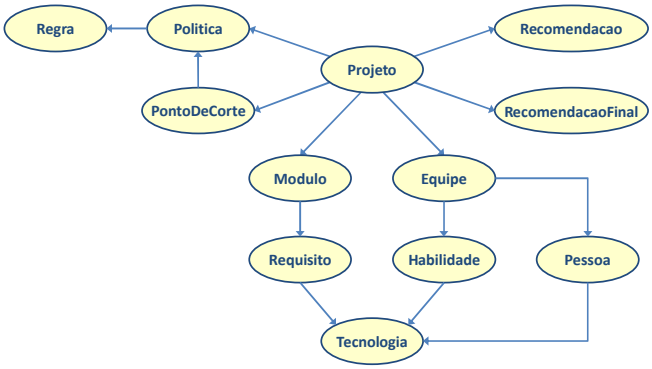

Figure 7 – Conceptual map of the *OntoDDS* ontology

It is important to point out that *OntoDDS* is described in OWL and hence, all the classes shown in Figure 7 are defined using the constructor *owl:Class*. For instance, Figure 8 shows the definition of the classes *Projeto*, *Modulo* and *Equipe*. Another point to highlight is that *OntoDDS* neither uses the constructor *owl:disjointWith* to create disjoint classes nor the constructor *rdfs:subClassOf* to create class hierarchies, except when necessary in cardinality restrictions of the object and datatype properties.

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

Figure 8 – Class definition

In the definition of object and datatype properties, we use the constructors *owl:ObjectProperty* and *owl:DatatypeProperty*, with its domain and range restrictions defined with the constructors *rdfs:domain* and *rdfs:range*.

Figure 9 shows the definition of the object property *temCandidata*, which has the class *Projeto* as domain and the class *Equipe* as range. Other types of constructors were also used in some object and datatype properties, and will be clearly indicated in the next subsections.

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

Figure 9 – Definition of an object property

The next subsections explain in detail the object and datatype properties of the classes, as well as their cardinality restrictions and inference axioms. It is important to point out that the subsection structure was defined considering the competency questions that must be answered by the ontology, as defined in Figure 3.

A. Characterization of Software Modules

The team skills and knowledge evaluation, required for the implementation of software modules, creates the need to represent information of the software modules, specially related to the technical requirements for their implementation. Therefore, the characterization of software modules must identify the technologies required to implement each of the software project modules, as well as the necessary skill levels. In order to perform this module characterization, the project manager must rely on the support of the software engineers and architect, the responsible parties for the specification of the software architecture.

In the ontology proposed here, the number of levels and the terms used to describe the knowledge levels may be redefined by the project manager, if he deems necessary. The initial proposal defines the knowledge levels using the terms *"Baixo"* (Low), *"Medio"* (Average) and *"Alto"* (High).

The characterization of the technical requirements must be performed for each software module that will be implemented in the software project. As we can see in Figure 10, in *OntoDDS* the characterization of the technical requirements for software modules is performed through the instances of the classes *Modulo* (Module), *Requisito* (Requirement) and *Tecnologia* (Technology), which are related through the object properties called *temRequisito* (hasRequirement) and *naTecnologia* (inTechnology).

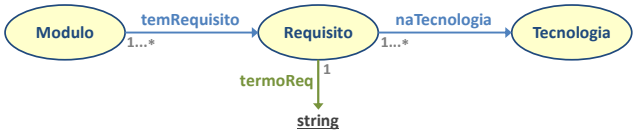

Figure 10 – Modules, Requirements and Technologies

The object property *temRequisito* associates a specific software module *m* with a requirement *r*, and through its datatype property *termoReq* (reqTerm), flags the required knowledge level *n*, whose initially proposed levels are *"Baixo"* (Low), *"Medio"* (Average) and *"Alto"* (High).

Besides, the object property *naTecnologia* associates the requirement *r* with a specific technology *t*. Hence, together, these classes and properties represent the fact that the module *m* has the requirement *r* in the technology *t*, with required knowledge level *n*.

It is important to point out that in the definition of a class, we can also define cardinality restrictions for the object and datatype properties inside this class using the constructors *owl:minCardinality*, *owl:cardinality* and *owl:maxCardinality*.

Figure 10 illustrates instances of the class *Modulo* that has at least one associated requirement through the object property *temRequisito*. Similarly, instances of the class *Requisito* have at least one associated technology through the object property *naTecnologia*. Besides, each instance of the class *Requisito* has exactly one textual term associated through the datatype property *termoReq*. Figure 11 highlights the cardinality restriction of the object property *temRequisito* to the class *Modulo*, indicating that each module must have at least one associated requirement.

```
<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>
```

Figure 11 – Cardinality restriction

B. Characterization of Development Teams

After characterizing the technical requirements for software modules, it is necessary to gather information from the teams on the required technologies to implement them. In *OntoDDS* the skill and technical knowledge that each team possesses on each technology must be measured and then represented by a real number in the interval [0, 1].

For each technology, three pieces of data must be gathered: *years of experience*, *number of developed projects* and *number of degrees*. As can be seen in [15][16], in general, the years of experience in a specific technology, as well as the degrees in this technology (including certifications and courses) can be used to evaluate whether an individual is an expert in a specific technology. Weiss [17] explains that an important factor to determine if someone is an expert in a specific domain is his discriminating ability, which is taken as the ability to identify subtle differences in similar contexts. Nevertheless, this ability to discriminate can only be achieved

through observation of the events as the years go by, given that is typically an empirical data.

Nevertheless, it is possible to infer, to a certain degree, the discriminating ability of a person through the number of projects he/she has previously worked on. That is, given that the ability to discriminate is acquired through the participation in many different projects with similar contexts, the more someone participates in projects, the higher the probability that he/she will have analyzed domains that have small differences, which increases his ability to discriminate [18]. This way, we can say that the piece of data *number of developed projects* has a high correlation with the ability to discriminate and can be used as its replacement.

Hence, information pertaining years of experience, number of developed projects and number of degrees are used in the *OntoDDS* ontology to characterize the technical attributes of teams. It is important to point out that this information is widely discussed in the literature pertaining to certain areas, such as expert identification [16][17], team selection [18], expert recommendation [19][20], human resources allocation [21][22][23], task assignment [24] and employee recruitment [25][26].

Figure 12 shows that in the ontology proposed the teams are represented by individuals of the classes *Equipe* (Team), *Pessoa* (Person) and *Tecnologia* (Technology), which are related by the object properties *possuiPessoa* (hasPerson), *temExperiencia* (hasExperience) and *temProjeto* (hasProject), as well as through the datatype property called *temTitulo* (hasDegree).

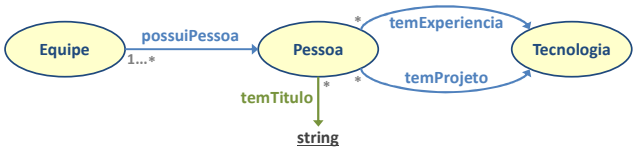

Figure 12 – Teams, Persons and Technologies

The object property *possuiPessoa* (hasPerson) associates a certain member *m* from a certain team *e*, which, through its datatype property *temTitulo* (hasDegree) and its object properties *temExperiencia* (hasExperience) and *temProjeto* (hasProject), associate the member *m* to a specific technology *t*. Hence, together, those classes and properties represent that the team has one or more members with degrees, projects and experiences in the many technologies that are required in the software project at hand.

Figure 13 shows the properties *temExperiencia* e *temProjeto*, which possess sub-properties representing respectively, the years of experience a member *m* has in a technology *t*, as well as the number of projects developed by member *m* in a technology *t*.

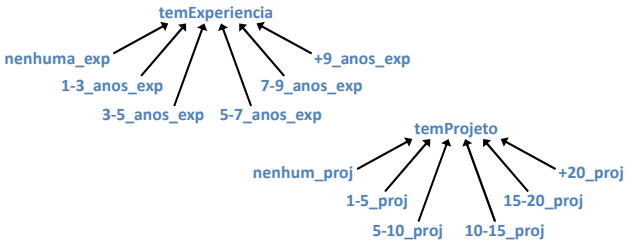

Figure 13 – Sub-properties for years of experience and number of projects

It is important to highlight that the sub-properties *temExperiencia* and *temProjeto* are defined with the constructors *owl:ObjectProperty* and *rdfs:subPropretyOf*, as shown in Figure 14 for the sub-property *nenhuma_exp* (no_experience).

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

Figure 14 – Definition of sub-property

At this point, from the characterization of the knowledge and skill set of the developers in all technologies, the project manager can derive and measure empirically or mathematically, the technical skills of each team in relation to technologies required by software modules. It must be pointed out, though, that in the case studies performed, the mathematical approach developed by Santos [15] was adopted to derive the technical skills of the teams. In this mathematical approach, based on the forms filled by each developer about the years of experience, number of degrees and projects in each technology, the answers are weighted in a set of equations that derives the level of knowledge from each developer in each technology. Next, based on the skill level for each member of each team in a specific technology, we can derive mathematically the knowledge level of the whole team in that technology.

Once derived and measured the technical skills for each team, now it is necessary to represent them in the ontology. Figure 15 shows in now we represent in the ontology using classes *Equipe* (Team), *Habilidade* (Skill) and *Tecnologia* (Technology), related by properties *temHabilidade* (hasSkill) and *naTecnologia* (inTechnology). Property *temHabilidade* associates a given team *e* to one or more skills *h*, which, through its datatype property *valorHab* (skillValue) signalizes the real numeric value within the interval [0, 1] that represents the team skill. On the other hand, the property *naTecnologia* associates a skill *h* to a specific technology *t*. Hence, together, these classes, object properties and datatype properties represent that a team *e* has skill *h* in technology *t*.

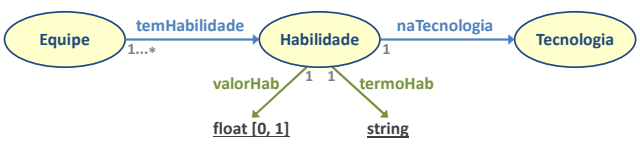

Figure 15 – Teams, Skills and Technologies

Even though the team skill level is calculated as a real number in the interval [0, 1], the ontology also represents technical skill as textual terms such as: *"Nenhuma"* (None), *"Baixa"* (Low), *"Media"* (Average) and *"Alta"* (High). These terms also represent the team technical skill in a specific technology. Figure 15 shows that the technical skills and their respective terms are represented as the datatype properties *valorHab* (skillValue) and *termoHab* (skillTerm), which represent respectively the numerical value and the textual term that characterize the technical skill a team *e* has in a specific technology *t*.

C. Characterization of Selection Policies

Once identified and represented the technologies required to implement the software modules, as well as the technical skills of each team in each of those technologies, it is necessary to define a policy for the selection of the teams that are technically qualified to implement the software modules.

According to the needs of the software project, different policies may be adopted, changing the way the teams can be selected. For instance, if a project is late, selecting the teams more qualified to implement the modules may be the best option, making it easier for them to perform their task in a shorter time. Nevertheless, choosing the most qualified teams is not always the ideal choice, given that selecting them may cause them to waste their knowledge and also lead to higher costs, in case their technical knowledge level is much higher than the one required to implement the software modules. Hence, it is important to select a policy that tries to choose teams with technical knowledge levels as close as possible to those required to implement the software modules, in order to avoid knowledge waste and minimize project costs.

Hence, the definition of selection policies is determined by the specific project needs and by the organizational context in which the software project is immersed. Consequently, it is of the utmost importance to allow project managers to adjust the adopted policies or create new ones according to the needs of different software projects.

A selection policy can be understood as a table of rules of the type **IF-THEN**, which correlate the terms in the rows with the ones in the column, defining rules that generate the desired results, represented by the cells in their intersections. Table I shows a possible example of selection policy. Notice that the number of rules in a policy is equivalent to the product of the number of rows with the number of columns.

.

Table I – Selection policy

			Module		
			Required Knowledge Level		
			Low	**Average**	**High**
Team Technical Skill Level		**None**	Average	Low	None
		Low	High	Average	Low
		Average	Average	High	Average
		High	Low	Average	High

At Table I, we can understand the rule composition with the following example: **IF** *Team Skill Level* is *"None"* **AND** *Required Knowledge Level* is *"Average"* **THEN** *Suitability Level of this team to this module* is *"Low"*.

The proposed ontology represents the policies as individuals of the classes *Politica* (Policy) and *Regra* (Rule), which are related by the object property *temRegra* (hasRule), as can be seen at Figure 16. Please observe that a certain policy *p* must be associated with a set of rules $\{r_1, r_2, ..., r_n\}$, modeling each of the cells that make up the table that represents the selection policy at hand.

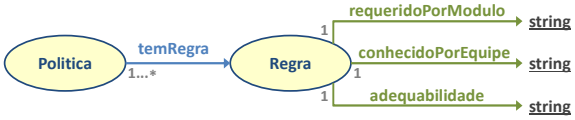

Figure 16 – Policies and Rules

In turn, rules are modeled using the datatype properties called *requeridoPorModulo* (requiredByModule), *conhecidoPorEquipe* (knownByTeam) and *adequabilidade* (suitability), which represent, respectively, the knowledge level required in a specific technology *t* by the module *m*, the technical proficiency of team *e* in this technology *t*, and, consequently, the technical suitability of team *e* for the implementation of module *m* concerning the technology *t*.

D. Characterization of the Technically Proficient Teams

Once we know the information concerning the technologies that are required to implement the software modules, as well as the team skill levels on these technologies, it is important to apply the selection policy to discover the technical suitability of each team to implement each module.

The proposed ontology represents the technical suitability of the teams as recommendations. As discussed above, this ontology has two kinds of recommendations, represented by the classes *Recomendacao* (Recommendation) and *RecomendacaoFinal* (FinalRecommendation).

Recommendation of Teams to Required Technologies

The conceptual map illustrated in Figure 17 shows in detail the characterization of the class *Recomendacao*. Considering a team *e*, a software module *m* and a technology *t* required to implement it, the purpose of a recommendation is to identify which rule *r* of the selection policy *p* must be chosen.

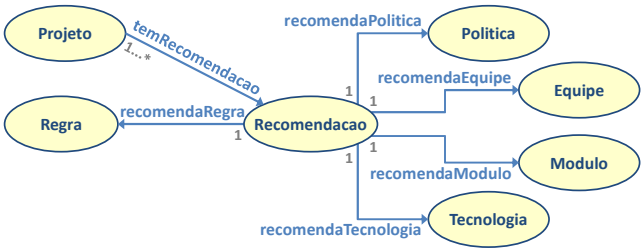

Figure 17 – Recommendation of teams to technologies

In order to represent this relationship between the recommendation, the adopted selection policy, the evaluated team, the software module under scrutiny, the considered technology and the instantiated selection rule, each instance of the class *Recomendacao* has a set of object properties, which is: *recomendaPolitica* (recommendsPolicy), *recomendaEquipe* (recommendsTeam), *recomendaModulo* (recommendsModule), *recomendaTecnologia* (recommends Technology) and *recomendaRegra* (recommendsRule). Together, these object properties represent, respectively, the selection policy adopted in the software project, the development team under evaluation, the software module under scrutiny, the technology required to implement it and, finally, the rule of the selection policy that must be considered.

It is important to point out that the object properties whose domain is the class *Recomendacao* are represented as functional properties using the constructor *owl:FunctionalProperty*, as illustrated in Figure 18 for the object property *recomendaRegra*. Together, these properties signalize that each recommendation must be associated to a single policy, team, module, technology and rule.

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

Figure 18 – Functional property

It should be noted that the object properties *recomendaPolitica*, *recomendaEquipe*, *recomendaModulo* and *recomendaTecnologia* can be derived from information already stored in the ontology, that is, the characterization of the modules and the required technologies; the characterization of teams and their knowledge level on each required technology; the characterization of the selection policy adopted in this specific software project. For instance, to infer the property *recomendaEquipe* (recommendsTeam), we only need a query to *OntoDDS* to identify all the candidate teams associated with this project and, next, instantiate a new recommendation and associate it to each candidate team through the property *recomendaEquipe*. The other properties are inferred in a similar way and, at the end, we have a recommendation for each combination of policy, team, module and technology.

On the other hand, the object property *recomendaRegra* must be inferred based on the selection policy adopted, considering the development team under evaluation, the software module that needs to be implemented and the associated technology. At this point, the rule inference indicated for each recommendation is performed by the axiom represented in Figure 19. In order to infer the selection policy rule, we need to identify: (i) the selection policy *po* adopted by project *pr*; (ii) the knowledge level *vreq* required by module *m* at technology *t*; and (iii) the skill level *vh* of team *e* at technology *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)
```

Figure 19 – Axiom for the recommendation of a selection rule

At the axiom in Figure 19, it should be noted that the policy *po* adopted at project *pr* is inferred in a direct way, evaluating the object property *adotaPolitica(?pr, ?po)*, modeled in the conceptual map previously shown in Figure 5.

In order to identify the knowledge level *vreq* required by module *m* at technology *t*, the axiom evaluates some object and datatype properties. Initially, the properties *temRequisito(?m, ?req)* and *naTecnologia(?req, ?t)* identify a specific requirement *req*, which represents the fact that module *m* requires technology *t*.

Next, considering the requirement *req*, the datatype property *termoReq(?req, ?vreq)* identifies the knowledge level *vreq*, required by module *m* at technology *t*.

Now, to identify the skill level *vh* of team *e* at technology *t*, the axiom also considers some object and datatype properties, modelled in the conceptual map previously shown at Figure 15.

First of all, the object properties *temHabilidade(?e, ?h)* and *naTecnologia(?h, ?t)* identify a specific skill *h*, which represent the fact that team *e* has knowledge on technology *t*. Hence, considering skill *h*, the datatype property *termoHab(?h, ?vh)* identifies the skill level *vh* of team *e* at technology *t*.

At this point, knowing the adopted policy *po*, the knowledge level *vreq* required by the module at the evaluated technology and the skill level *vh* of the team at this technology, the axiom can infer the adopted rule *r*, evaluating the object property *temRegra(?po, ?r)* and the datatype properties *requeridoPorModulo(?r, ?vreq)* and *conhecidoPorEquipe(?r, ?vh)*, both of which were modeled in the conceptual map shown at Figure 16.

Finally, once we identified the rule *r* to be adopted, the axiom infers the object property *recomendaRegra(?re, ?r)*, representing the fact that the recommendation *re* must adopt

rule *r*, which in turn represents in its datatype property *adequabilidade* (suitability), the suitability level related to the knowledge of the team at the technology under discussion required by the software module.

Recommendation of Teams to Software Modules

Based on the development team recommendation for each required technology it is possible to measure the suitability of the teams for each software module. For this, the project manager must adopt an empirical or mathematical formula to calculate the team suitability to the modules, based on the team suitability for each technology required by each module.

This team suitability for each software module is represented in the ontology by the class *RecomendacaoFinal* (FinalRecommendation), whose conceptual map is shown in Figure 20. Each final recommendation is characterized by having the object properties *recomendaPolitica* (recommendsPolicy), *recomendaEquipe* (recommendsTeam), *recomendaModulo* (recommendsModule) and the datatype properties *valorAdeq* (suitabilityValue), *termoAdeq* (suitabilityTerm) and *adequada (*adequate).

The object properties represent, respectively, the policy that was selected in the software project, the evaluated candidate team and the soon to be implemented module. Observe that the object properties whose domain is the class *RecomendacaoFinal* are represented as functional properties using the constructor *owl:FunctionalProperty*, being specified in a way that is similar to the example of Figure 18. Together, these properties signalize that each final recommendation must be associated to a single policy, team and module.

The datatype properties *valorAdeq* and *termoAdeq* represent, respectively, the numeric value in the interval [0, 1] and the textual term for the team suitability to the module that will be implemented. This information will consolidate the possible candidate teams that can implement the modules of a specific software project.

Figure 20 – Recommendation of Teams to Software Modules

Please observe that we did not include the datatype property *adequada* (suitable) on purpose, because it is related to the application of the cut point which will be seen later in this section.

Once we calculated the individual technical skill for each development team member and the team technical ability, these numerical suitability values are converted to fuzzy textual terms through the application of axioms, so that it is possible to determine the final team suitability, which can be seen in Figures 21, 22, 23 and 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")
```

Figure 21 – Axiom for Final Suitability “Nenhuma” (None)

```
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")
```

Figure 22 – Axiom for Final Suitability “Baixa” (Low)

```
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")
```

Figure 23 – Axiom for Final Suitability “Media” (Average)

```
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")
```

Figure 24 – Axiom for Final Suitability “Alta” (High)

Observe that in Figure 20, the object properties called *recomendaPolitica* (recommendsPolicy), *recomendaEquipe* (recommendsTeam) and *recomendaModulo* (recommends Module) can be derived from information already stored in the ontology, related to the characterization of modules, teams and selection policies. At this point, the inference of textual suitability term referring to the numerical value can be automatically performed by the axioms. In order to infer the suitability term, we need to identify: (i) the policy *po* adopted by the project *pr*; (ii) the numerical value of the suitability *rf* of team *e* to module *m*.

For example, in the axiom of Figure 21, please observe that the textual term for suitability is inferred in a direct way evaluating the property *valorAdeq(?rf, ?v)*, modeled in the conceptual map illustrated in Figure 20.

In order to identify the textual term for suitability of team *e* to module *m* based on policy *po*, the axiom evaluates some object and datatype properties modeled in the conceptual map of Figure 20. Initially, the object properties called *recomendaPolitica(?rf, ?po)*, *recomendaEquipe(?rf, ?e)* and *recomendaModulo(?rf, ?m)* identify a final recommendation *rf* associated to a specific policy *po*, a specific team *e*, and a specific module *m*. Next, the datatype property *valorAdeq(?rf, ?v)* identifies the numerical value of suitability *v* of team *e* to implement module *m* using policy *po*. Finally,

considering the numerical suitability value *v*, the axiom infers the textual term for suitability of the final recommendation *rf*. For that, the axiom performs a comparative analysis among the numerical value to suitability *v* and the intervals defined for each of the textual terms. For example, in the axiom of Figure 21, the textual term is *"Nenhuma"* (None). For a team to have this textual term, it is necessary that the suitability is higher than or equal to 0 and less than 0,15.

The axioms for the next suitability terms (*"Baixa"*, *"Media"* and *"Alta"*), can be seen in Figures 22, 23 and 24, respectively, according to the limits defined for the numeric intervals of the textual terms.

Application of the Cut Point

With the goal of filtering out the teams that might have a low suitability, a cut point defined by the project manager must be used. This step consists simply in eliminating those teams that do not reach the cut point. For that, we must update the instances of the class *RecomendacaoFinal* (FinalRecommendation), setting the value of its datatype property called *adequada* (suitable) as illustrated in Figure 20. It is important to point out that the update of the property *adequada* is made automatically through an ontology axiom, as will be detailed below in this section.

Figure 5 shows that in the *OntoDDS* ontology, each project has its own selection policy and its own cut point, represented by the relationship among classes *Projeto*, *Politica* and *PontoDeCorte*, through the object properties *adotaPolitica* (adoptsPolicy), *temPontoDeCorte* (hasCutPoint) and *naPolitica* (inPolicy). Please notice that in order to be possible to adopt different cut points for different selection policies, it was necessary to define a relationship between individuals of the classes *PontoDeCorte* and *Politica*, through the object property *naPolitica*, as is better illustrated by Figure 25.

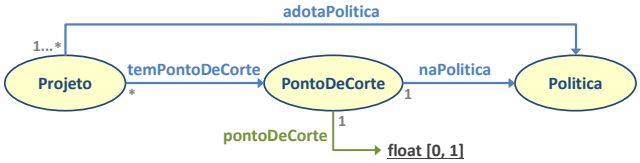

Figure 25 – Detailed view of *Ponto de Corte* (Cut Point)

The object property *temPontodeCorte* (hasCutPoint) associates a project *p* to a specific cut point *pc*, which through its datatype property *pontoDeCorte* (cutPoint) stores a real numeric value *n* in the interval [0, 1], stipulated by the project manager to determine if a specific team is able to implement a specific software module. On the other hand, the object property *naPolitica* (inPolicy) associates the cut point *pc* to a specific policy *po*. Hence, together, those classes and properties represent the fact that the project *p* has the cut point *pc* with value *n* in the policy *po*.

It is important to point out that the object property *naPolitica* is represented as a functional property using the constructor *owl:FunctionalProperty*, in a way that is similar to the one shown in Figure 18.

The object properties *temPontoDeCorte* and *naPolitica* and the datatype property *pontoDeCorte* may be derived directly from information already stored in the ontology on the project characterization and its adopted selection policy. At this point the suitability value inference as a function of the cut point is made by the axiom represented in Figure 26. In order to infer whether the team suitability is acceptable in relation to the cut point, we need to identify the following: (i) the policy *po* adopted by the project *pr*; (ii) the numeric value of the suitability *va* of team *e* in module *m*; and (iii) the numeric value of the cut point *vpc* adopted by the policy *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)
```

Figure 26 – Axiom for *Ponto de Corte* (Cut Point)

In order to identify if the numeric value of the suitability *va* of a team *e* to implement module *m* is acceptable considering the cut point *pc*, the axiom evaluates some object and datatype properties modeled in the conceptual map shown in Figure 25. Initially, the object properties *temPontoDeCorte(?pr, ?pc)* and *naPolitica(?pc, ?po)* identify the specific cut point *pc* adopted by the policy *po*. Next, considering the policy *po*, the datatype property *pontoDeCorte(?pc, ?vpc)* identifies the numeric value of the cut point *vpc*, required by the policy *po*.

At this moment, knowing the numeric value of the cut point *vpc* required by the adopted policy, the axiom can evaluate whether the team suitability numeric value *va* is greater than or equal to the cut point *vpc*. For that, the axiom evaluates the datatype property *valorAdeq(?rf, ?va)*, and, finally, infers the datatype property *adequada(?rf, true)*, which represents that the recommendation *rf* is considered adequate according to the cut point.

IV. USE CASE

In order to evaluate the usability and applicability of the proposed ontology, we developed three use cases based on the project of two different software product lines. Details of the case studies can be found in [27].

The two first cases were developed using a hypothetical software product line in the area of electronic commerce (e-commerce) documented in [28]. These two first use cases were organized in two development iterations, contemplating the phases of domain engineering and application engineering of the product line. Next, another use case was developed based on a real project of a middleware product line for mobile devices called *Multi-MOM* [29] whose instantiation will be briefly illustrated next in this section.

When conducting the use cases, first the *OntoDDS* ontology

was completely specified and validated in the Protégé tool [11], contemplating the classes, object and datatype properties, restrictions and axioms. Next, each use case was also instantiated and validated in the Protégé tool, including individuals of the several elements of the *OntoDDS* ontology.

The Protégé tool supports the OWL specification language [13], recommended by W3C. Using this tool, it was possible to create and model classes, object and datatype properties, axioms and restrictions, as well as to create class instances. Besides, the Protégé tool allows for queries and visualization of the results that are automatically generated by the several axioms in the ontology.

A. Characterization of Software Modules

Multi-MOM [29] is a middleware product line for mobile computing that is essentially focused in the communication functionality. Considering its component-based architecture presented in Figure 27, we defined five software modules, according to the phase *recommending software modules* [30] of the team selection and allocation framework [7], briefly explained in section I of this paper.

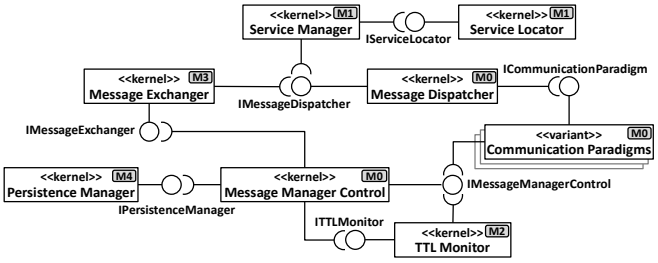

Figure 27 – Multi-MOM Architecture

Figure 27 shows that we identified five different modules indicated in the small rectangles labeled with the terms *M0*, *M1*, *M2*, *M3* and *M4*. The characterization of the technologies required by the modules was performed by the software architect that created and designed *Multi-MOM*. As an example, Figure 28 illustrates the instantiation of the *OntoDDS* ontology to characterize the technologies required to implement module *M1*.

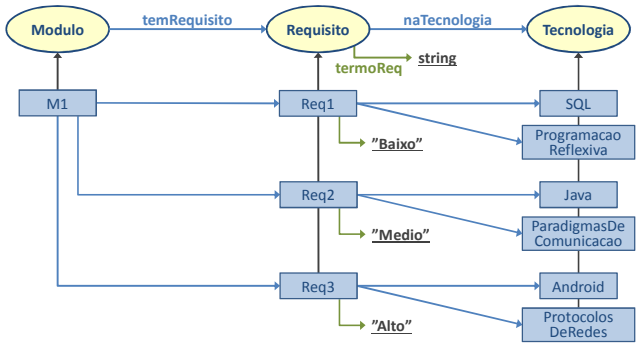

Figure 28 – Characterization of Module M1

Figure 28 shows that the module M1 requires the technologies *SQL* and *ProgramacaoReflexiva* (Reflective Programming) with knowledge level "Baixo" (low). On the other hand, it requires a level "Medio" (average) of knowledge on *Java* and *ParadigmasDeComunicacao* (Communication Paradigms). Finally, it requires a level "Alto" (high) of knowledge on the technologies *Android* and *ProtocolosDeRedes* (Network Protocols).

B. Characterization of Development Teams

Considering the difficulty of finding real development teams for use cases, the development team definition was performed base on the local market and students from the Computer Science course that answered a questionnaire contemplating all the technologies required for the use case, according to the modules to be implemented in their respective product lines. This questionnaire was performed online, resulting in a set of 179 participant developers. The adopted forms and the respective answers of developers can be found in [18].

Next, the answered questionnaires were used to characterize the skills and technical knowledge of the 179 developers in each technology required by the modules. Figure 29 shows an example instantiation of the proposed ontology for the characterization of the skills and technical knowledge in *Java* of developer *D1* that belongs to team *E1*. As can be seen, developer *D1* has from five to seven years of experience in *Java*, has participated in up to five projects that adopt *Java* and has the SCJA and SCJP certificates.

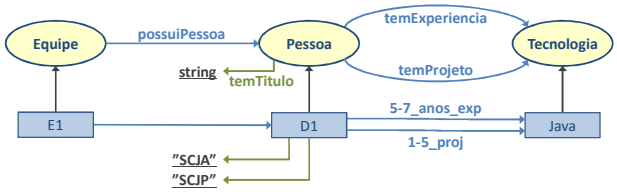

Figure 29 – Characterization of Developer *D1* in *Java* technology

Based on a set of 179 developers, we created 22 teams with different sizes (from 2 to 18), dividing the members randomly until we completed all teams. The final composition of the teams was: 1 team with 2 members, 3 teams with 3 members, 5 teams with 5 members, 4 teams with 8 members, 2 teams with 9 members, 3 teams with 10 members, 3 teams with 15 members and 1 team with 18 members.

Next, based on the skills and technical knowledge of each developer, it is possible to characterize the skills and technical knowledge of the respective teams for each technology that was required by the software modules. Figure 30 shows an example of an instantiation in *OntoDDS* of the characterization of team *E1* in the *Java* technology. As can be seen, considering the skills and technical knowledge of its developers, team *E1* has a technical skill level with value *0,61* in the *Java* technology, which, according to the ranges of levels adopted, characterizes an average skill, represented by

the textual term *"Medio"*.

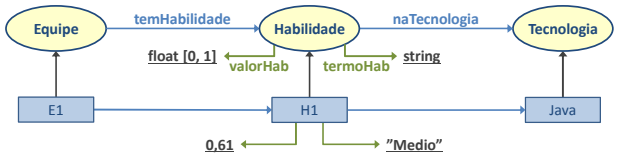

Figure 30 – Characterization of Team *E1* in *Java* technology.

C. Characterization of Selection Policies

In the instantiation of the proposed ontology, we initially specified four different selection policies, created based on the observations and analysis presented in other works in the literature [21][22][23][25]. The four proposed policies are:

a) Policy of equivalent qualification: selects teams that have the technical skills close to the required to implement the software modules;

b) Policy of most skilled teams: selects teams that have the highest technical skills, independently of the knowledge level required by the software modules;

c) Policy of minimum qualification: selects teams that possess the minimum technical skills required to implement the software modules;

d) **Policy of training provision:** selects teams that have technical skills bellow the required to implement the software modules;

For instance, considering the selection policy of equivalent qualification, previously defined in Table I, the rule instantiation represented by the intersection of the third row with the second column of Table I, here called *R8*, is presented in Figure 31. According to this policy, the instantiated rule is interpreted as follows: **IF** *Required Technical Skill is "Medio" (Average)* **AND** *Technical Skill Level is "Medio" (Average)* **THEN** *Suitability Level is "Alto" (High).* It is important to point out that in this use case the 12 rules of Table I were numbered from *R1* to *R12*, going from the left to the right and the top to the bottom.

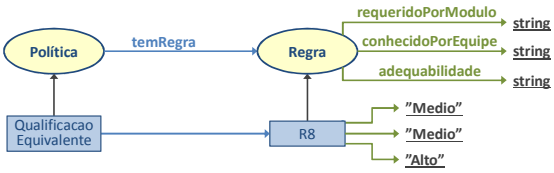

Figura 31 – Characterization of Rule R8 in the Selection Policy

Table II shows that different cut points were used for each selection policy adopted. Based on the use cases performed, we realized that the suitability values for the teams varied according to the adopted selection policy, which was expected due to the fact that different policies attribute different suitability to teams. Nevertheless, in an experiment analysis where each use case was evaluated according to each selection policy, we say a trend of the training provision policy to present suitability values higher than all the other ones. On the other hand, the minimum qualification policy tends to present higher values than the equivalent qualification and more skilled team policies. Finally, we also realized that the equivalent qualification policy tends to generate higher values than the more skilled team policy. Given this empirical evidence, we decided to use different cut points for each selection policy under consideration in the use cases.

Table II – Cut Points

Selection Policiy	Cut Point
Equivalent Qualification	0,60
Most skilled teams	0,55
Minimum Qualification	0,70
Training Provision	0,75

Figure 32 exemplifies the instantiation of the cut points in the ontology, showing the representation of the cut point of value 0,60 adopted in the selection policy *QualificacaoEquivalente* (Equivalent Qualification) used in the *Multi-MOM* project.

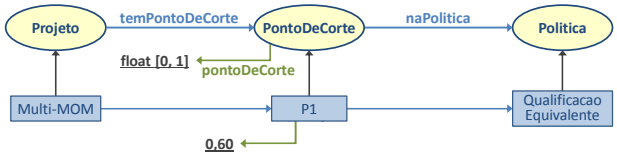

Figure 32 – Cut point used in the *QualificacaoEquivalent* Policy

D. Evaluation of Team Suitability

At this point, considering the technologies required by the modules, the team technical skills in each technology and the selection policy adopted in the project, we can infer the technical suitability for each team in each technology required by each module, according to the selection policy. Figure 33 shows an example of technical suitability inference, that of team *E1* in *Java* technology required by module *M1*, according to the selection policy *QualificacaoEquivalente*.

As we can see in Figure 33, the referred suitability is defined by the application of rule *R8*, whose instantiation in the proposed ontology was shown in Figure 31. It is relevant to point out that the selection rule inference adopted is performed by the axiom in Figure 19.

At this point, it is possible to measure empirically or mathematically the suitability of the teams to the software modules. For that, in these use cases, we adopted the mathematical approach proposed in [15] to derive the team suitability to the modules, based on the team suitability to each technology required by the software modules.

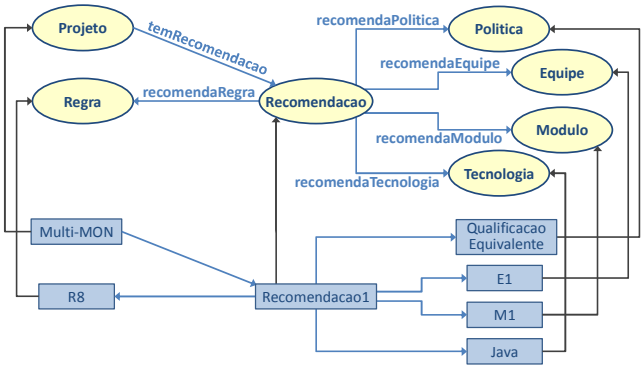

Figure 33 – Technical Suitability of Team *E1* to *Java* in Module *M1*

Figure 34 shows an example of the final recommendation of team E1 to module M1, whose numeric suitability value is 0,64. If we apply the axioms of Figures 21, 22, 23 and 24, it is possible to infer the textual terms that represent the suitability. In Figure 34, the suitability textual term is *"Medio"* (Average).

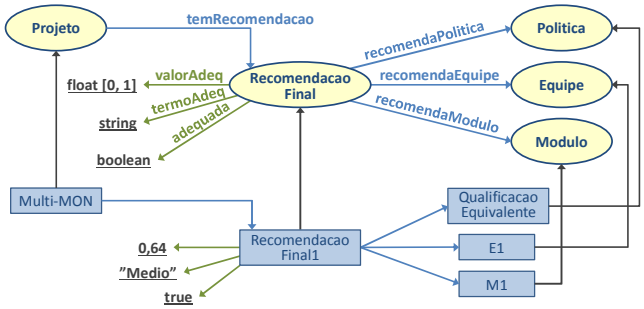

Figure 34 – Recommendation of Team E1 to Module M1

Finally, based on the axiom of Figure 26, we can infer the technically suitable teams for each software module from the evaluation of the cut point defined in the software project to the selection policy at hand, defining hence the possible candidate teams for the implementation of the software project modules. Please observe that in the datatype property *adequada* (suitable), Figure 34 already includes the result of the suitability inference of team *E1* to module *M1* in the policy *QualificacaoEquivalente*.

In the use case of the project of the *Multi-MOM* product line, after applying the cut point, among the 22 candidate teams, we received recommendations for 5, 11, 12, 21 and 19 teams to implement modules *M0*, *M1*, *M2*, *M3* and *M4*, respectively.

Considering four selection policies defined and three use cases developed to evaluate the usability and applicability of the proposed ontology, each use case resulted in four recommendations of suitability of the teams to the modules, generating one recommendation for each selection policy. Hence, considering all use cases, we generated 12 different recommendations, whose details can be found in [27].

V. Related Work

In this section we present and discuss three approaches identified in the literature which are related to our work to some extent. The three approaches considered are: *(i) OntoDiSEN* [31] – an ontology to share information on DSD projects; *(ii) Burbeck's proposal* [32] – an ontology to establish electronic contracts; and *(iii) ICARE* [19] – an expert recommendation system that uses an ontology to characterize users and specialists.

It is important to mention the difficulty to find proposals in the literature that are directly related to the selection of technically qualified distributed teams. Hence, in spite of the fact that the identified approaches do not share the specific purpose of supporting team selection in distributed software projects, the discussed works present some aspects related to *OntoDDS* because they adopt ontologies to represent information associated with DSD environments, to support the definition of criteria to hire electronic services and to characterize users and experts.

In order to guide the comparison of approaches evaluated with the *OntoDDS* approach here proposed, we synthesize the main characteristics in Table III. Next, we present a brief description of each related work, together with a comparative discussion in relation to *OntoDDS*.

OntoDiSEN [31] is an application domain with the purpose of describing concepts and contextual elements, which are represented, stored and shared by an information dissemination tool, allowing the communication and cooperation among members of the geographically distributed teams, and so, increasing their perception about actions related to produced artifacts. In such a colaborative scenario, *OntoDiSEN* is the element responsible for representing contextual information, promoting the dissemination of the context in a uniform and standardized way between distributed teams.

In *OntoDiSEN*, the information of the skills and required knowledge are associated with the phases of the process. Hence, *OntoDiSEN* adopts a target entity with thicker granularity in relation to *OntoDDS*, which associates this information to software models, whose granularity is thinner.

In terms of the characterization of skills and knowledge required by the project phases, *OntoDiSEN* allows for the instantiation of multiple non-valued attributes. For instance, they may represent the requirements in different technologies, tools or processes, without quantifying those needs. In a similar way, *OntoDDS* also allows the instantiation of multiple attributes to characterize required skills and knowledge but, differently, those needs are quantified in different levels.

Following a similar approach, in *OntoDiSEN*, the users' knowledge and skills are also characterized by the instantiation of multiple non-valued attributes, while *OntoDDS* represents such information by instantiating multiple attributes to

characterize the developers' skill and knowledge, but as valued attributes, that is, quantified in different levels. Besides, instead of characterizing only individuals (users, in the case of *OntoDiSEN*), *OntoDDS* also characterizes the skills and knowledge of development teams based on the knowledge of their respective developers.

OntoDiSEN does not define any method or mechanism to gather knowledge or skills that the users possess and is demanded by the process phases, letting the project manager decide on how to get them. *OntoDDS*, on the other hand, establishes the adoption of implementation tables to represent the technologies required to implement the software modules and, in the case of developers, is based on forms to gather the information related to years of experience, number of developed projects and number of degrees.

Table III – Comparative of the related works

Proposal	Target Entity	Characterization of the Target Entity	Target Resource	Characterization of the Target Resource	Input Data Capture	Selection Policy
OntoDiSEN	Process phases	Multiple non-valued attributes	Users	Multiple non-valued user attributes	–	–
Burbeck	Services	Multiple valued attributes	Suppliers	Multiple valued supplier atributes	–	Implicit; Unchangeable
ICARE	Users	Multiple non-valued keywords	Experts	Multiple non-valued expert attributes	–	Implicit; Unchangeable
OntoDDS	Software modules	Multiple valued attributes	Teams	Multiple valued attributes from developers and teams	Tables and forms	Explicit; Configurable

The second approach evaluated is the Burbeck's proposal [32], which presents an ontology to support hiring services whose goal is to represent information to be used during the establishment of electronic contracts, both in a generic way, as well as specifically in the context of DSD. The ontology proposed by Burbeck allows the definition of non-functional requirements of QoS (quality of service) related to electronic services, as well as information necessary to clients and suppliers so that it is possible to evaluate if the requirements are satisfied. Hence, the ontology can be applied to support the establishment of electronic contracts, and be used to represent the concepts and relationships involved in a negotiation between the companies participating in a possible hire during the software development process.

Even though the purposes of *OntoDDS* and Buberck's proposal are different, we can correlate them indirectly. While *OntoDDS* is used to select development teams to implement software modules, Burbeck's proposal is used to hire suppliers for the execution of electronic services. Consequently, we can consider that the target entities have similar granularity, since software modules and electronic services can be considered as correlates.

Considering the characterization of QoS attributes for electronic services, Burbeck's proposal allows for the instantiation of multiple valued attributes which are quantified in different levels. Hence, in this aspect, it can be considered similar to *OntoDDS*, which also allows the instantiation of multiple valued attributes to characterize different levels of skill and technical knowledge required to implement software modules.

In an equally comparable way, in Burbeck's proposal the QoS attributes assured by the suppliers are characterized by the instantiation of multiple valued attributes which are quantified in different levels. From the point of view of *OntoDDS*, there is the analogous fact that the knowledge and skills of developers and the teams to which they belong are also instantiated in multiple valued attributes, representing their skill levels in the respective technologies, methods, processes or application domains.

In a similar way to *OntoDiSEN*, Burbeck's proposal does not define any mechanism or method to gather QoS attributes that are required by services and assured by suppliers, leaving it to the project manager the task of defining the way to obtain them. As previous discussed, *OntoDDS* behaves differently, adopting implementation tables and forms to represent the required technologies and the knowledge and skills of developers and teams.

The third approach evaluated is *ICARE* (*Intelligent Context Awareness for Recommending Experts)* [19], an expert recommendation system for specific domains, characterized by keywords supplied by users and taking into consideration the current context of users and experts. Notice that to characterize users and experts, as well as the contextual information and relationships among keywords and subjects of interest, *ICARE* adopts a domain ontology that allows for the

inference of personalized recommendations of different experts for different users in several subjects of interest.

It can be seen that in *ICARE* the purpose is not team selection. Nevertheless, we can indirectly correlate the skills and knowledge required by the software modules in *OntoDDS* with the subjects and domains of interest of the users in *ICARE*. Hence, in both cases, the goal is to model the needs of their target entities, which are software modules in *OntoDDS* and users in *ICARE*.

When dealing with the needs of users in terms of subjects and domains of interest, *ICARE* allows the users to inform multiple non-valued keywords, hence not adopting any type of quantification of the importance of each informed keyword. Thus, in relation to *OntoDDS*, *ICARE* can be considered less sophisticated or realistic, given that *OntoDDS* allows for the instantiation of multiple valued attributes which characterize the different skill and knowledge levels required by software modules.

In an analogous way, *ICARE* also characterizes the technical knowledge of the experts through the instantiation of multiple non-valued attributes and hence, does not consider the difference in the knowledge level of those experts. On the other hand, in *OntoDDS*, the knowledge and skill of the developers and their teams are considered by the instantiation of multiple valued attributes, which allows for the consideration of differences in the knowledge levels of these developers and their respective teams.

ICARE shares the same deficiencies with *OntoDiSEN* and Burbeck's proposal in the sense that it does not define any mechanism or method to capture the technical knowledge of experts, leaving to the project manager the responsibility of defining a way to gather them. *OntoDDS* is different, for it establishes implementation tables and forms to represent the required technologies and the knowledge and skills for the developers.

Finally, we realize that *OntoDiSEN* does not require and hence does not represent the concept of selection policy, for it has not the purpose of selecting any kind of target resource, but the sharing of contextual information in DSD projects. In a different way, both in the hiring of suppliers in Burbeck's proposal as well as in the recommendation of experts in *ICARE*, a selection policy concept is necessary. Nevertheless, in both proposals the selection policy is implicit and unchangeable in the ontological model, probably represented as rules to the inference model. Differently from both, we have the more explicit and configurable model of *OntoDDS*, in which different selection policies can be defined in the ontology by the project manager and their selection rules will be treated automatically and transparently by the ontology axioms in the inference engine.

VI. Final Considerations

In this paper we presented an application ontology to support the selection of technically qualified distributed teams for the implementation of software modules in software projects.

The proposed ontology is part of a recommendation framework [7] whose main goal is to support project managers in the process of allocating distributed teams to implementation tasks of software modules in software product line projects.

The *OntoDDS* ontology has four concept blocks that are related and that allow to perform the characterization of the following elements in a software project: *(i)* required technologies to implement software modules; *(ii)* skills and technical knowledge of the development teams in the technologies required by the software modules; *(iii)* selection policies; and *(iv)* technical suitability of the development teams to the software modules.

Please observe that the four concept blocks represent the concretization of each of the competency questions for the proposed ontology, which were mentioned in Figure 3 of Section II. Hence, it can be noticed that the *OntoDDS* ontology performs all the goals it is proposed to.

The main contribution of this work, adopting the strategy *divide and conquer*, is the model and formalization in a systematic and structured way of an extremely complex problem, which is the selection of technically qualified distributed teams for the implementation of software modules in distributed software projects.

The general structure of *OntoDDS* is shown in the conceptual map of Figure 7, where all the problem is modeled using only 12 classes, related by 23 object properties and 11 datatype properties, which, when instantiated, can systematize the decision making process of the project manager, especially when observed through the point of view of the high complexity of the problem, which is clear when this problem is dealt with in an *ad hoc* way. Besides, the proposed ontology facilitates the communication between the project manager and the team members, because it establishes a common vocabulary between all the stakeholders in the selection process.

An instantiation of *OntoDDS* for a distributed software project may require a considerable effort for the creation of the instances and their datatype and object properties, and consequently is prone to error which may cause a waste of time. For instance, considering the use case of *Multi-MOM*, presented in Section IV, whose architectural project was grouped into 5 software modules with requirements in 7 different technologies, and was evaluated to the suitability of 22 teams with 4 different selection policies, the number of class instances (3,267), object properties (19,150) and datatype properties (1,982) is staggering, requiring a remarkable effort to manipulate them inside the Protégé tool.

Nevertheless, *OntoDDS* offer an additional tool, its six axioms that allow for the automatic inference of an object property and two datatype properties. In the use case of *Multi-MOM*, the axioms infer 2,376 object properties and 880 datatype properties, representing a coverage of about 12.5% of the object properties and 44.4% of the datatype properties.

It is also important to point out that in spite of the high number of instances and their respective object and datatype properties, the proposed ontology has potential to be reused in many different scenarios. For instance, once a given software project is instantiated, with its software modules, required technologies, candidate teams and adopted selection policy, the evaluation of another selection policy may easily reuse all the instances and object and datatype properties related to the software modules, required technologies and candidate teams. In a most significant way, if we devise a data base of previous software projects, including most technologies usually required to implement software modules, a large number of candidate teams and the main selection policies adopted, the evaluation of a new software project may also reuse all the instances and datatype and object properties related to the technologies, teams and selection policies.

Even considering the reuse potential of the proposed ontology, it is still required a considerable effort during the manual instantiation to identify and manipulate the instances and their object and datatype properties that may be reused and those that need to be created.

In order to decrease this effort, the instantiation of the ontology could be performed programmatically, exploring the API of the Protégé tool, avoiding errors and saving time. Just as an illustration to the extremely positive impact of the programmatic approach, consider an application where the user signalizes in a specific set of tables: the software modules, required technologies to implement them, the candidate teams and their members. In such an application, it could almost all be created in an automatic and transparent way, including all instances and object and datatype properties.

Considering the discussed points, we can synthesize the following direct benefits or additional contributions of the adoption of *OntoDDS* to the problem of team selection:

i. Better understanding of the problem domain;
ii. Easier communication among the stakeholders in the team selection process, given that a common vocabulary is defined;
iii. Formalization of the concepts and relationships associated with the team selection process;
iv. Possibility of performing inferences on the domain when backed by tools with support to inference engines;
v. Reuse of the information on modules, teams, technologies and selection policies in different scenarios and software projects.

In spite of the relevant benefits and contributions, some limitations were observed in the use cases and mentioned in the previous discussion. The limitations are the following:

i. Not adopting the Protégé API tool to manipulate the ontology database in a programmatic way with Java;
ii. Use cases not performed with real developer teams from the software industry;
iii. Adoption of fuzzy terms without using fuzzy logic in the decision making process.

First, without using the API of the Protégé tool, the creation of classes, instances, and object and datatype properties was performed in a completely manual way, being prone to errors and causing waste of time.

Second, the development teams considered in the use case are fictitious teams based on local software developers and students from Computer Science courses, which do not provide for a real validation of the proposed ontology, even though they may make possible to evaluate its usability and applicability.

Finally, even though the requirements, skills and suitability are represented by fuzzy terms, the decision making process modeled in the selection policies does not fully contemplate the fuzzyfication, inference and defuzzyfication steps of the fuzzy logic, which are based on fuzzy sets and pertinence functions.

Given those limitations, we identified some future works, among them we include the following:

i. Adoption of the Protégé API tool with the goal of manipulating the ontology database in a programmatic way, using, for instance, the framework Jena [33];
ii. Validation of the ontology in a real project with globally distributed development teams;
iii. Evaluation of the logical complexity of the ontology;
iv. Full modeling of the decision making process using fuzzy logic in selection policies.

ACKNOWLEDGEMENT

This work was supported by the Nation Institute of Science and Technology for Software Engineering (INES – www.ines.org.br) and funded by the CNPq, process number 573964/2008-4.

REFERENCES

[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, available in http://protege.stanford.edu, last access 09/08/2015.

[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, available in http://www.w3.org/TR/owl-guide, last access 09/08/2015.

[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, An approach for recommending modules in distributed development projects of software product lines (in portuguese). V Workshop on Distributed Software Development (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, An approach for selection of technically qualified teams during the implementation of software projects (in portuguese), Master Dissertation, Federal University of Paraíba, 2014.

[19] H. Petry, ICARE: a context-aware expert recommendation system (in portuguese), Master Dissertation, Federal University of 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, Staffing a software project support: a constraint satisfaction based approach (in portuguese), Master Dissertation, Federal University of Rio de Janeiro, COPPE, 2005.

[22] M. A. Silva, WebAPSEE-Planner: support to people instantiation in software projects through policies (in portuguese), Master Dissertation, Federal University of Pará, 2007.

[23] D. A. Callegari, L. Foliatti, R. M. Bastos, MRES – a tool for resource selection in software projects through a fuzzy, multi-criteria approach (in portuguese), Brazilian Symposium on Software Engineering (SBES), Tools Session, 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, An ontological approach for recommending qualified teams in software projects (in portuguese), Master Dissertation, Federal University of 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: a multi-paradigm, extensible and message-oriented middleware for mobile computing (in portuguese), Master Dissertation, Federal University of Paraíba, 2010.

[30] T. Burity, An approach for recommending modules in distributed development projects of software product lines (in portuguese), Master Dissertation, Federal University of Paraíba, 2011.

[31] A. P. Chaves, DiSEN-CSE: a context-awareness model for disseminating information in a distributed software development environment (in portuguese), Master Dissertation, State University of 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, available in http://jena.apache.org, last access 04/06/2015.