Evaluating an Application Ontology for Recommending Technically Qualified Distributed Development Teams

Larissa Barbosa, Gledson Elias
Informatics Center
Federal University of Paraíba
João Pessoa, Brazil
e-mail: larissa@compose.ufpb.br, gledson@ci.ufpb.br

***Abstract*—As a reflection of globalization, Distributed Software Development (DSD) has become a mainstream approach, in which the cooperation among globally distributed software development teams has the potential to reduce cost and development time. However, in order to make such promises a reality, it is important to find teams with specific technical background, required for implementing software modules that constitute the software product under development. Thus, it is a key aspect to contrast technical background of development teams against specified technical requirements for implementing the software project, making possible to select the most skilled teams to develop each software module. In such a context, this paper presents the evaluation of an application ontology that supports selection processes of distributed development teams, which are technically skilled to implement software modules in distributed software projects. Experimental results show that the evaluated 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.**

Keywords-ontology; distributed software development; selection process.

I. INTRODUCTION

In software engineering, a great body of knowledge has been accumulated over the last decades regarding methods, techniques, processes and tools, improving productivity and software quality. As such, several software development approaches have been proposed by academia and industry. Nowadays, as a mainstream approach, Distributed Software Development (DSD) promotes the cooperation among globally distributed teams for implementing different software product modules, reducing the development cost and time, favored by the hiring of cheaper staff in different locations, the fast formation of development teams and the adoption of the follow-the-sun development strategy [1][2]. Besides, DSD also enables to find qualified workforces and domain experts in worldwide outsourced teams or even teams in global coverage companies [3][4][5].

In order to make DSD promises a reality, it is a key task to identify development teams with specific skills and technical knowledge required to develop software modules that compose the software product under development. In such a context, it is important to compare the skills and technical knowledge of the candidate development teams against the technical requirements specified to implement each software module, making possible to identify those that are more qualified to implement each one.

However, in DSD projects, geographic dispersion may cause difficulties for the project manager to assess the skills and technical knowledge of the candidate teams. In most cases, the project manager does not develop face-to-face activities with remote teams, having neither direct personal contact nor drinking fountain talks [6]. Hence, it is therefore hard to get precise and up-to-date information about members of such remote teams, given that the formal communication mechanisms based on documents or data repositories do not react as quickly as informal ones. Besides, even when the project manager has a bit of information about candidate teams, in large software projects, the task of selecting teams may still be quite complex and prone to evaluation errors, since different teams may adopt ambiguous vocabulary and incompatible methods to identify and evaluate their skills and technical knowledge.

As a consequence, we have proposed a layered recommendation framework [7] as a mean to help project managers in the selection and allocation of development teams in DSD projects. The framework is composed of three recommendation phases: *recommending software modules* – intends to cluster components into software modules, reducing dependencies among modules and hence, minimizing communication requirements; *recommending qualified teams* – aims to identify technically qualified teams to implement each software module; and *recommending teams allocation* – intends to suggest possible allocations of software modules to qualified development teams, concerning their non-technical attributes as a mean to reduce inter-team communication requirements.

In the context of the framework, this paper presents the experimental evaluation of an application ontology, called *OntoDSD* [8], whose main goal is to support the selection of distributed development teams that are technically skilled to implement software modules in DSD projects. Note that *OntoDSD* is part of the second phase of the recommendation framework which, as mentioned, is called *recommending qualified teams*. As the main contribution, experimental results show that *OntoDSD* 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.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-590-6

The rest of this paper is organized as follows. Section II introduces the main concepts and components related to an ontology. Section III presents an overview of the *OntoDSD* ontology, explaining its main 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 the experimental evaluation. Next, Section V presents final remarks, identifies limitations and indicates future work.

II. FUNDAMENTS

The literature contains many definitions of an ontology. For the purposes of this paper, as defined in [9], an ontology is a formal explicit description of concepts in a given domain of discourse, properties of each concept describing its features and attributes, and restrictions on such properties.

In general, the concepts in the modeled domain are represented by elements called *classes*, which can adopt inheritance abstraction to formulate a class hierarchy, in which each class inherits properties from one or more superclasses. Classes can have *instances*, which correspond to individual objects in the modeled domain. A class has several 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 can have a set of allowed values, ranging from simple types to instances of classes. Properties can be divided into *object properties* and *datatype properties*. Object properties associate instances of one or two classes. Datatype properties create a relationship between a class instance and values of a certain simple type, such as strings and numbers. Each instance can have concrete values for the properties of its respective class.

In relation to development methodologies, there are several proposals in the literature to systematize the construction and evolution of ontologies [10]. However, despite the valuable contributions, none of them can be considered the correct one. Indeed, none of them has enough maturity, and therefore, there is no consensus on the best, most complete or most appropriate methodology that can be widely applicable in varied domains and application needs.

As a result, due to its simplicity of documentation, ease of application, extensive tooling support and focus on the construction of ontologies, we opted for the methodology *Ontology Development 101* [9], which defines a very simple guide based on an iterative approach that assists ontology designers, even non-experts, in the creation of ontologies using a support tool, such as Protégé [11].

In the *OntoDSD* development, we decided to adopt a *top-down* approach because it favors better control of the level of details, avoiding excessive details present in a *bottom-up* approach, which may lead to greater rework, effort and inconsistencies, and moreover, may hinder the identification of relationships and similarities among concepts [12].

It is important to highlight that the development of the ontology was specified using the Protégé tool [11], which provides support for the constructors of the Web Ontology Language (OWL) [13], recommended by the World Wide Web Consortium (W3C).

III. ONTODSD

OntoDSD is an application ontology, which has the main goal of supporting the selection of distributed development teams, technically skilled to implement software modules in DSD projects. Thus, the modeled domain is *DSD projects*, and, in a more specific way, the scope is the *selection of technically qualified teams to implement software modules*. Figure 1 presents the *OntoDSD* conceptual map.

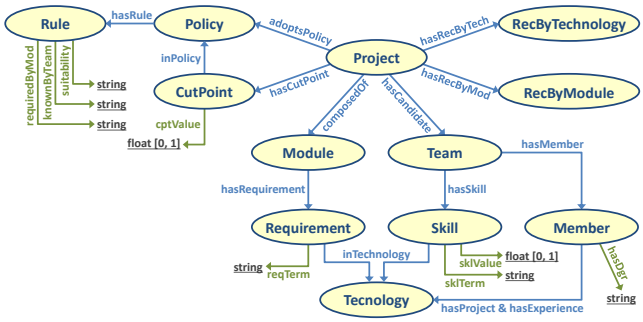

Figure 1. OntoDSD conceptual map.

In *OntoDSD*, a DSD project (*Project*) is composed of a set of software modules (*Module*) that can be developed by a set of globally distributed development teams (*Team*). In Figure 1, the object property *composedOf* represents the relationship between a project and its constituting software modules. The object property *hasCandidate* represents the relationship between a project and distributed development teams, which are candidates to implement software modules.

A software project (*Project*) adopts selection policies (*Policy*) for recommending development teams to implement software modules based on different criteria (*Rule*) and cut points (*CutPoint*), which establish a minimum suitability level for considering a team adequate to implement software modules. In Figure 1, the object property *adoptsPolicy* represents the relationship among a project and possible selection policies, according to specific project needs. The object property *hasCutPoint* represents the relationship between a project and defined cut points, each on related to each possible policy using the object property *inPolicy*.

OntoDSD provides two types of recommendations. The first, called *RecByTechnology*, represents the suitability level of candidate teams in relation to each technology required by software modules. The second, called *RecByModule*, represents the suitability level of candidate teams for implementing each software module. In Figure 1, the object properties *hasRecByTech* and *hasRecByMod* represent the relationships between a project and their recommendations.

A. Representing Software Modules

Considering a given software module (*Module*), it is important to characterize the knowledge requirements (*Requirement*) imposed in relation to technologies (*Technology*) adopted to implement the module. In *OntoDSD*, the knowledge requirement indicates the knowledge level required in each technology.

As illustrated in Figure 1, the object property *hasRequirement* associates a specific software module with a

knowledge requirement, and through its datatype property *reqTerm*, flags the required knowledge level, whose initially proposed levels are *low*, *medium* and *high*. It is important to note that the number and the value of the terms used to label the knowledge level may be redefined by the project manager. Now, regarding a given knowledge requirement, the object property *inTechnology* associates the knowledge requirement with a specific technology. Hence, together, these classes and properties represent the fact that a given module has a certain knowledge requirement in relation to a particular technology, demanding a given specified knowledge level.

B. Representing Development Teams

In *OntoDSD*, a development team (*Team*) is composed of a set of members (*Member*). In Figure 1, the object property *hasMember* represents the relationship between a team and its constituting members. Regarding a given development team, it is vital to gather information about each member in relation to technologies (*Technology*) required by software modules. To do that, for each required technology, three pieces of data must be gathered: *years of experience*, *number of developed projects* and *number of degrees*. As can be seen in [14][15][16], in general, such information can indicate whether an individual is an expert in a specific technology.

In *OntoDSD*, via the datatype property *hasDgr* and the object properties *hasExperience* and *hasProject*, each member (*Member*) is associated to a specific technology (*Technology*). Note in Figure 2 that properties *hasExperience* and *hasProject* have sub-properties for representing respectively, the years of experience a given member has in a specific technology, as well as the number of projects developed by the member in such a technology. Hence, together, such classes and properties represent that a given team has members with degrees, projects and experiences in many technologies.

Figure 2. Sub-properties for years of experience and number of projects.

Now, such gathered information about members of a given team (*Team*) allows to infer the skill and technical knowledge (*Skill*) possessed by the whole team in relation to each technology (*Technology*). In Figure 1, the object property *hasSkill* associates a given team to one or more skills, which in turn are associated to their respective technologies using the object property *inTechnology*. For each skill, the datatype properties *sklValue* and *sklTerm* signalize the real numeric value within the interval [0, 1] and the correspondent textual term, such as *none*, *low*, *medium* and *high*, representing the skill level of the team. Hence, together, such classes, object and datatype properties represent that a given team has a specified technical skill level in a certain technology. Again, the number and the value of the terms used to label the skill level may be redefined by the project manager.

C. Representing Selection Policies

In order to evaluate the technical suitability of candidate teams, it is necessary to define a selection policy. According to the needs of the software project, different policies may be adopted, changing the way the teams can be selected. A selection policy is a table of rules (Table I), stated by *if-then* expressions, which correlate terms in rows and columns, defining rules that generate desired results, represented by cells in their intersections. We can realize the rule rationale with an example: *IF Skill Level is "none" AND Knowledge Level is "medium" THEN Suitability Level is "low"*.

TABLE I. SELECTION POLICY

		Technical Requirements		
		Knowledge Level		
		low	**medium**	**high**
Teams Skill Level	**none**	*medium*	*low*	*none*
	low	*high*	*medium*	*low*
	medium	*medium*	*high*	*medium*
	high	*low*	*medium*	*high*

OntoDSD represents policies as individuals of the classes *Policy* and *Rule*, which are related by the object property *hasRule*, as shown in Figure 1. Observe that a certain policy must be associated with a set of rules, modeling each cell of the selection policy table. In turn, rules are modeled using the datatype properties *requiredByMod*, *knownByTeam* and *suitability*, representing, respectively: the knowledge level required in a given technology, the technical skill possessed by a certain team in that technology and accordingly, the suitability level owned by that team in that technology.

D. Representing Technically Skilled Teams

Now, it is time to apply the selection policy in order to discover the technical suitability owned by each team to implement each software module. *OntoDSD* represents the technical suitability possessed by teams as recommendations. As discussed before, there can be two kinds of recommendations, *RecByTechnology* and *RecByModule*, which are characterized in the conceptual map in Figure 3.

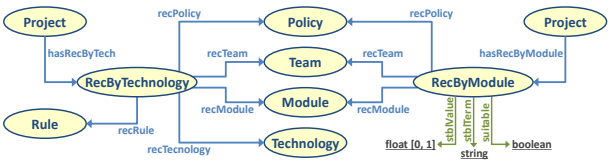

Figure 3. Recommendations in OntoDSD.

1) Recommendation of Teams to Required Technologies

A recommendation *RecByTechnology* represents the suitability level possessed by a certain team (*Team*) in relation to a particular technology (*Technology*) required by a specific module (*Module*) according an adopted policy (*Policy*). Indeed, the suitability level is signalized by an instance of the rule (*Rule*) trigged by the adopted selection policy, in which the datatype property *suitability* (Figure 1) indicates the textual term representing the suitability level.

As can be seen in Figure 3, the relationship between such concepts is represented using a set of object properties:

recPolicy, *recTeam*, *recModule*, *recTechnology* and *recRule*. It should be noted that such object properties can be derived through inference from information already stored in the ontology. In order to infer such properties, *OntoDSD* has a set of specified axioms, which are not discussed herein for simplicity, but interested readers can found details in [8].

2) Recommendation of Teams to Software Modules

Based on the suitability level possessed by a given team for each required technology, it is possible to estimate the suitability level owned by that team in each software module, which in *OntoDSD* is represented by the recommendation *RecByModule*. To do that, the project manager ought to adopt an empirical or mathematical method, like the one proposed in [14].

A recommendation *RecByModule* represents the suitability level possessed by a given team (*Team*) in relation to a particular module (*Module*) according an adopted policy (*Policy*). Indeed, the suitability level is signalized by the datatype properties *stblValue* and *stblTerm* associated to the respective recommendation (Figure 3), which indicate its numerical value and textual term, respectively.

As shown in Figure 3, the relationship between such concepts is represented via the object properties *recPolicy*, *recTeam* and *recModule*. Note that such object properties can also be derived by inference from information already present in the ontology. However, for simplicity, the set of related axioms are not discussed herein, but detailed in [8].

3) Application of the Cut Point

With the goal of filtering out the teams that might have a low suitability level, 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. As depicted in Figure 1, the object property *hasCutPoint* associates a project (*Project*) to a specific cut point (*CutPoint*), which through its datatype property *cptValue* stores a real numeric value in the interval [0, 1], stipulated by the project manager to determine the suitability possessed by a given team in relation to a certain software module.

To do that, we must update the instances of the recommendation *RecByModule*, setting the value of its datatype property *suitable*, illustrated in Figure 3. It is important to point out that the update of the property *suitable* is also inferred automatically through an ontology axiom.

IV. CASE STUDY

In order to evaluate the usability and applicability of the *OntoDSD* ontology, we developed three use cases based on the project of two different software product lines. The two first cases were developed using a hypothetical software product line in the e-commerce area, documented in [17]. 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* [18], whose instantiation will be illustrated next in this section.

When conducting the use cases, first the *OntoDSD* ontology was completely specified and validated in the Protégé tool [11], which supports the OWL specification language [13]. Using Protégé, it was possible to create and model classes, object and datatype properties, restrictions and axioms. Next, each use case was also instantiated and validated in Protégé, including individuals of the several *OntoDSD* ontological elements. Besides, Protégé allows for queries and visualization of the results that are automatically generated by several *OntoDSD* axioms.

A. Representing Software Modules

Multi-MOM [18] is a middleware product line for mobile computing. As shown in Figure 4, its component-based architecture has five software modules, indicated in the small rectangles labeled from *M0* to *M4*, according to the first phase of the proposed framework [7], explained in Section I.

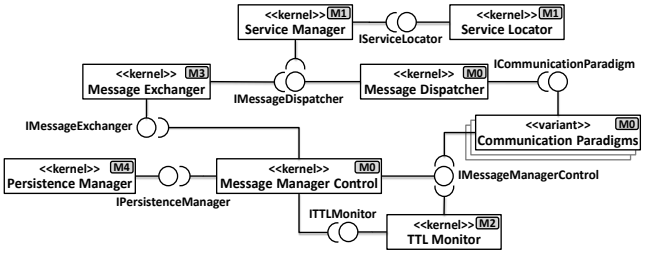

Figure 4. Multi-MOM architecture.

The characterization of the technologies required by those modules was performed by the software architecture that created and designed *Multi-MOM*. As an example, Figure 5 illustrates the *OntoDSD* instantiation to characterize the technologies required to implement module *MD0*. As illustrated in Figure 5, module *M0* requires technologies *Communication Paradigms*, *Reflexive Programming*, *Android* and *Java* with *"high"* knowledge level. Besides, it requires *"medium"* knowledge level on *SQL*.

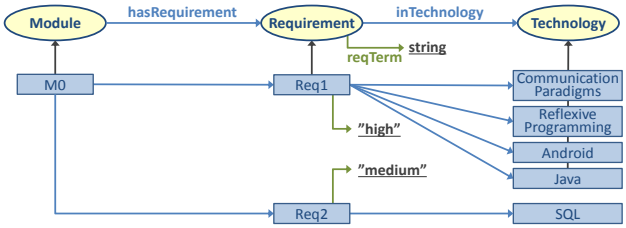

Figure 5. Characterization of module M0.

B. Representing Development Teams

Considering the difficulty of finding real development teams for use cases, the development team definition was performed based on the local market and computer science students, resulting in a set of 179 participant developers, which answered online forms covering all technologies required by modules of the use cases. The adopted forms and the respective answers can be found in [14].

Next, based on the answered forms, the skills and technical knowledge of the 179 developers were characterized in each technology required by modules.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-590-6

Figure 6 shows an example instantiation for characterizing the skills and technical knowledge in *Android* possessed by member *MB1* that belongs to team *T20*. As can be seen, *MB1* has between three and five years of experience in *Android* and has participated in up to five projects that adopt *Android*.

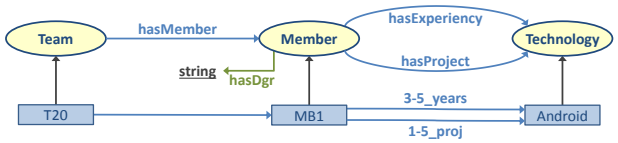

Figure 6. Characterization of member MB1 in Android Technology.

Regarding 179 developers, we created 22 teams with different sizes, varying from 2 to 18, dividing the members randomly until complete 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 required by modules. Figure 7 shows the instantiation for characterizing team *T20* in *Android* technology. As can be seen, considering the skills and technical knowledge of its developers, team *T20* has a technical skill level with value *0,88* in *Android* technology, which, according to the ranges of levels adopted, characterizes a "*high*" skill.

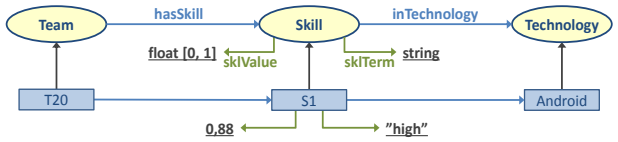

Figure 7. Characterization of team T20 in Android technology.

It is important to stress that each candidate team can consist of colocalized members only. Thus, if one needs to consider a candidate team consisting of members from different locations, it is suggested to model different teams for each location. Besides, *OntoDSD* does not represent non-technical aspects related to DSD projects in geographical, temporal, cultural and economic dimensions. Such a design decision is a consequence of the layered architecture of the proposed framework [7], introduced in Section I, which deals with such non-technical aspects in its third phase called *recommending teams allocation*.

C. Characterization of Selection Policies

In the *OntoDSD* instantiation, we initially specified four different selection policies, created based on the observations and analysis presented in other related proposals in the literature [19][20][21][22]. The four proposed policies are:

a) **Equivalent qualification:** selects teams with technical skills close to knowledge level required by modules.
b) **Most skilled teams:** selects teams that have the highest technical skills, independently of the knowledge level required by modules.
c) **Minimum qualification:** selects teams that possess minimum technical skills required by modules.
d) **Training provision:** selects teams that have technical skills bellow the required by modules.

For instance, considering equivalent qualification policy, defined in Table I, the rule instantiation represented by the intersection of the fourth row with the third column, here called *R12*, is presented in Figure 8. The instantiated rule is interpreted as follows: *IF Skill Level is "high" AND Knowledge Level is "high" THEN Suitability Level is "high"*. It is important to point out that the 12 rules in Table I were numbered from *R1* to *R12*, going from the left to the right and the top to the bottom.

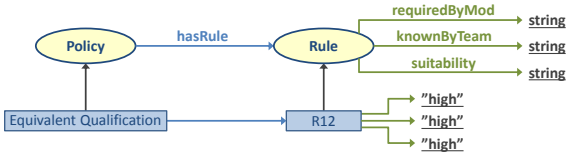

Figure 8. Characterization of rule R12 in selection policy.

Table II shows that different cut points were used for each selection policy. Based on the use cases, it was perceived that the suitability values for the teams varied in relation to adopted selection policies, reinforcing that different policies assign different suitability to teams. Nevertheless, in an experiment analysis where each use case was evaluated according to each selection policy, we saw a trend of the training provision policy to present suitability values higher than all 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.

TABLE II. ADOPTED CUT POINTS

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

Figure 9 exemplifies the instantiation of the cut points in *OntoDSD*, showing the representation of the cut point of value *0,60* adopted in the selection policy *Equivalent Qualification* used in the *Multi-MOM* project.

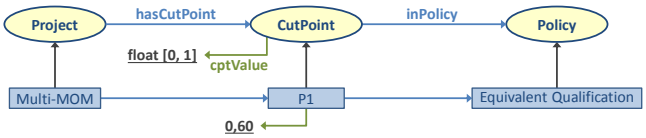

Figure 9. Cut point for policy Equivalent Qualification.

D. Evaluation of Team Suitability

At this point, considering technologies required by 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 10 shows the technical suitability inference for team *T20* in *Android*, which is required by module *M0*, according to the selection policy *Equivalent Qualification*.

As we can see in Figure 10, the referred suitability is defined by the application of rule *R12*, whose instantiation in *OntoDSD* was shown in Figure 8.

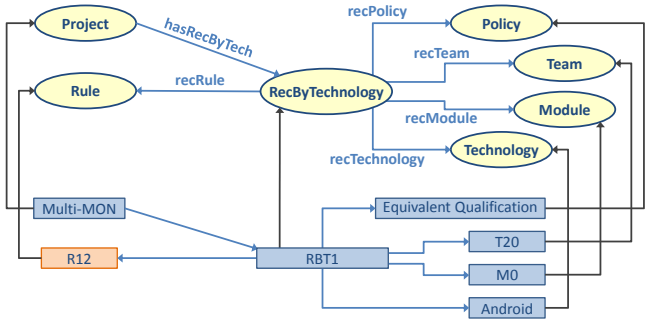

Figure 10. Technical suitability of team T20 to Android in module M0.

It is relevant to point out that the adopted rule *R12* is inferred by an *OntoDSD* axiom, illustrated in Figure 11, as specified in Protégé. As already indicated, *OntoDSD* has six axioms for inferring six ontological elements: selection rules, suitability terms and technically suitable teams. Herein, figures show ontological elements inferred by axioms in orange color. However, for simplicity, the other axioms are not presented, but interested readers can found them in [8].

```
RecByTechnology(?re), hasRecByTech(?pr, ?re),
recPolicy(?re, ?po), recTeam(?re, ?e), recModule(?re, ?m), recTechnology(?re, ?t),
knownByTeam(?r, ?vh), requiredByModule(?r, ?vreq) -> recRule(?re, ?r)
```

Figure 11. Axiom for recommending a selection rule.

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 by Santos [14] to derive suitability level possessed by teams in software modules, based on suitability possessed by those teams in each technology required by software modules. In this mathematical approach, based on forms filled by each developer about years of experience, number of degrees and projects in each technology, the answers are weighted in a set of equations that derive the knowledge level owned by each developer in each technology. Next, based on the skill level owned by each member of each team in a specific technology, we can derive mathematically the knowledge level of the whole team in that technology.

Figure 12 shows an example of the final recommendation of team *T20* to module *M0*, whose numeric suitability value is *0,71*. Note that, applying *OntoDSD* axioms, it is possible to infer the textual terms that represent the suitability. In Figure 12, the suitability textual term is *"medium"*.

Finally, based on *OntoDSD* axioms, 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 software modules. Please note that in the datatype property *suitable*, Figure 12 already includes the result of the suitability inference of team *T20* to module *M0* in policy *Equivalent Qualification*.

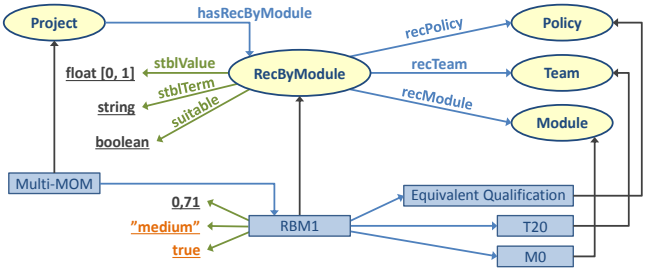

Figure 12. Recommendation of team T20 to module M0.

In the *Multi-MOM* use case, after applying the cut point, among the 22 candidate teams, *OntoDSD* recommended 5, 11, 12, 21 and 19 teams to implement modules *M0*, *M1*, *M2*, *M3* and *M4*, respectively. For instance, analyzing the recommendation for module *M0*, in sequence, teams *T20*, *T11*, *T16*, *T18* and *T19* are recommended as suitable considering the *Equivalent Qualification* policy.

Considering the high to medium knowledge levels required by module *M0* in all related technologies (Figure 5), an inspection by hand, in relation to skill levels possessed by all teams in such technologies, reveals that the recommended teams are better suited because their technical skills are closer to knowledge levels required by module *M0* in such technologies. Following such a rationale, it is possible to conclude that recommended teams are the most appropriate with respect to all adopted policies, but due space limitation, it is not possible to present and discuss in detail such manual inspection and assessment rationale.

In summary, regarding four selection policies defined and three use cases developed to evaluate the usability and applicability of the *OntoDSD* 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 set of recommendations.

V. Conclusion and Future Work

In this paper, we presented the evaluation of an application ontology, called *OntoDSD*, which supports the selection of technically qualified distributed teams for the implementation of software modules in DSD projects. As the main contribution, adopting the strategy divide and conquer, *OntoDSD* 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.

The general structure of *OntoDSD* is shown in the conceptual map in Figure 1, where the whole 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 viewpoint of the high complexity of the problem, which is

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-590-6

clear when this problem is handled in ad-hoc ways. Besides, *OntoDSD* facilitates the communication between the project manager and team members, establishing a common vocabulary between all stakeholders in the selection process.

The *OntoDSD* instantiation may require a considerable effort for creating instances and their object and datatype properties, and consequently is prone to error and may cause a waste of time. For instance, considering *Multi-MOM*, which includes 5 software modules, 7 technologies, 22 teams, and 4 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 in Protégé. In such a case, it was required around 500 man-hours to represent gathered information as instances and properties in Protégé.

Nevertheless, *OntoDSD* offers as an additional facility a set of axioms, allowing the automatic inference of object and datatype properties. In *Multi-MOM*, such axioms infer 2.376 object properties and 880 datatype properties, representing a coverage around 12.5% and 44.4%, in relation to object and datatype properties, respectively.

OntoDSD 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 instances and object/datatype properties related to 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 adopted selection policies, the evaluation of a new software project may also reuse all instances and object/datatype properties related to technologies, teams and selection policies.

Even considering the reuse potential of the *OntoDSD* 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, as a future work, its instantiation could be performed programmatically, exploring the Protégé API, 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: software modules, required technologies, 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/datatype properties.

REFERENCES

[1] R. Martignoni, “Global Sourcing of Software Development: A Review of Tools and Services”, 4th IEEE International Conference on Global Software Engineering (ICGSE 2009), IEEE, 2009, pp. 303-308.

[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´09), IEEE, 2009, pp. 1-9.

[3] J. Herbsleb and D. Moitra, “Global Software Development”, IEEE Software, issue 2, pp. 16-20, 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, issue 4, pp. 233-247, 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, issue 4, pp. 267-281, 2003.

[6] A. Mockus and J. Herbsleb, “Challenges of Global Software Development”, 7th International Symposium on Software Metrics (METRICS 2001), IEEE, 2001, pp. 182-184.

[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 (RSSE´10), ACM, 2010, pp. 36-40.

[8] L. Barbosa, “An Ontological Approach for Recommending Technically Qualified Teams in Distributed Software Projects”, Master Dissertation, UFPB, Brazil, 2013.

[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, 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, 2005.

[11] Protégé. Available in: http://protege.stanford.edu 2017.08.16.

[12] M. Uschold and M. Gruninger, “Ontologies: Principles, Methods and Applications”, The Knowledge Engineering Review, vol. 11, issue 2, pp. 93-136, 1996.

[13] OWL Web Ontology Language Guide. Available in: http://www.w3.org/TR/owl-guide 2017.08.16.

[14] V. S. Santos, “An Approach for Selecting Technically Qualified Teams in Software Projects”, Master Dissertation, UFPB, Brazil, 2014.

[15] J. Shanteau, D. J. Weiss, R. P. Thomas, and J. C. Pounds, “Performance-Based Assessment of Expertise: How to Decide if Someone is an Expert or not”, European Journal of Operational Research, vol. 136, issue 2, pp. 253-263, 2002.

[16] D. J. Weiss, J. Shanteau, and P. Harries, “People Who Judge People”, Journal of Behavioral Decision Making, vol. 19, issue 5, pp. 441-454, 2006.

[17] H. Gomaa, “Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures”, Addison-Wesley, 2004.

[18] Y. M. Bezerra, “Multi-MOM: A Multi-Paradigm, Extensible and Message-Oriented Mobile Middleware”, Master Dissertation, UFPB, Brazil, 2010.

[19] A. S. Barreto, “Staffing a Software Project: A Constraint Satisfaction Based Approach”, Master Dissertation, Federal University of Rio de Janeiro, Brazil, 2005.

[20] M. A. Silva, “WebAPSEE-Planner: Supporting People Instantiation in Software Projects through Policies, Master Dissertation, Federal University of Pará, Brazil, 2007.

[21] D. A. Callegari, L. Foliatti, and R. M. Bastos, “MRES: A Tool for Resource Selection in Software Projects through a Fuzzy, Multi-Criteria Approach, Brazilian Symposium on Software Engineering (SBES 2009), Tools Session, 2009, pp. 61-66.

[22] J. Collofello, D. Houston, I. Rus, A. Chauhan, D. M. Sycamore, and D. S. Daniels, “A System Dynamics Software Process Simulator for Staffing Policies Decision Support”, 31st Annual Hawaii International Conference on System Sciences (HICSS´98), IEEE, 1998, vol. 6, pp. 103-111.

Copyright (c) IARIA, 2017. ISBN: 978-1-61208-590-6