Context-sensitive Team Formation: Towards Model-Based Context Reasoning and Update

Hong Qing Yu, Yi Hong, Reiko Heckel and Stephan Reiff-Marganiec
Department of Computer Science, University of Leicester, UK
{hqy1, yh37, reiko, srm13 @mcs.le.ac.uk}

Abstract. Selection problems tend to have two aspects: one that is structural and one that is quantitative in nature. Here we investigate a method that allows decisions on both aspects. The paper considers a typical example, that of selecting members for a team, where decisions are based on context information. We show that graph transformations are providing a solution to the structural selection, while logic scoring of preferences allows qualitative decision making. On an implementation level OWL and SPARQL are used to retrieve and update context data.

1 Introduction

Selection problems occur in many aspects of computer systems and every-day life. For example in a service oriented computing system, one finds the need to select a service to complete a task. Or in a collaborative work environment (virtual or not) one finds a need to assemble teams to complete specific projects which in turn requires selecting team members. In the former example traditionally the terms functional and non-functional requirements are used to describe the *structural* selection (a service with the right interface) and a *qualitative* selection (the most suitable service guaranteeing specific QoS requirements).

The latter problem, of team formation, is the one we wish to concentrate on in this paper, as it is easy to explain, has merits of its own and allows us to present the method to solve it – however that method is more generic and the specific problem is to be seen as a case study. To select team members one needs to make decisions based on structural criteria (to find a member who fits the profile required) and a qualitative decision, to find the most suitable (experienced, qualified, …) such member.

Of course these selection problems are influenced by many factors, with one of the most interesting being context: both suggested selection problems are not performed on static domains (services come and go, as do possible team members, e.g. through being unavailable in certain weeks). Context captures the dynamic nature of the problem environment in a way suitable for processing. In general “context is information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user

and an application, including the user and applications themselves"[1]. Context is also a very important source of information in our computing environments.

In this paper we will use the case of team formation to investigate techniques addressing both the structural and the qualitative aspect of selection. The structural selection and update of context will be described by graph transformation rules, visually expressed in a UML-like notation, while the qualitative concern will be handled using the Logic Scoring Preference (LSP).

The paper is organized as follows: in Section 2, a motivating example of team member selection for a Java software project is given. In Section 3, we present our sample context model as a UML class diagram, which can be translated into an OWL ontology. Moreover, the technologies for retrieving of and reasoning about context information are described and we explain how to model the selection problems using graph transformation rules which can be translated into the SPARQL query language for OWL. In Section 4, the modified LSP method for selection and ranking of potential team members is illustrated. Finally, conclusions and future work are discussed in Section 5.

2 Context-sensitive Team Formation

Selecting the right people for establishing a professional team to conduct a specific task is always combined with hard decision problems. These problems often appear in e-business and e-government applications. The problems fall into two categories: structural decision problems and qualitative/quantitative decision problems. The basis for the decision process are provided by a context model capturing amongst others, the people's characteristics. The structural decision problem is captured by the question: how we can choose people who satisfy the desired context structure? Moreover, the target team formation requirements can also be expressed in this structural way. The structures are a subset of the context model's properties. The "qualitative/quantitative" is concerned with selecting the best from amongst the people who satisfy the team formation requirements.

Let us consider the following concrete example: A software company needs a professional team of 4 to work on a software project. The people required are an experienced specification analyzer (more than 5 years experience of analysis, having a qualification and currently working in a project team as analyzer), a software architect (at least 3 years Java design experience), and two good Java developers (more than 3 years Java coding experience and interest in web technology). The problem structure is captured by Fig. 1. Please note that we ignore additional people such as testers, managers and integrators in order to keep the problem simple for presentation.

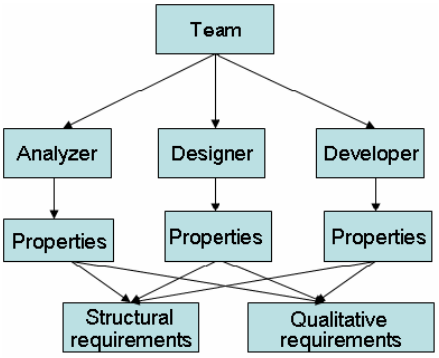


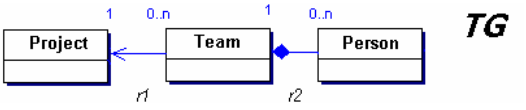
Fig. 1: Motivating example of a software development team

3 Model-based Context Reasoning by Graph Transformation

Since structural requirements can be presented as graphs, a graph-based approach is a natural candidate for addressing the problem. Graph transformations (GT) [2] provide such an approach, combining the modeling of data structures and configurations as graphs with the use of rules to describe the update of these structures. In our case, graphs represent snapshots of context data while rules model context selection, update and reasoning. Apart from this abstract model, which provides the semantic core of our approach, we suggest the use of UML as a human-oriented modeling notation, OWL (Web Ontology Language) as a machine-readable representation of context models, and SPARQL for pattern matching and reasoning on OWL. In this section we will describe these different levels and their relation through examples.

3.1 Graphs and graph transformation

In our approach, graphs occur at two levels. A static context model can be represented as a *type graph TG*, while a system snapshot and states are illustrated as *instance graphs G, H,…*. Dynamic changes regarding the update of structure or attribute values are modeled as graph transformation rules. A graph transformation rule $p: L \rightarrow R$ consists of a name p and a pair of instance graphs L, R over TG. The left-hand side graph L describes the pre-conditions of the rule while the right-hand side R shows the replacement of L after the transformation. The example below shows the type graph, rule, and transformation for transferring personnel between teams in a project.



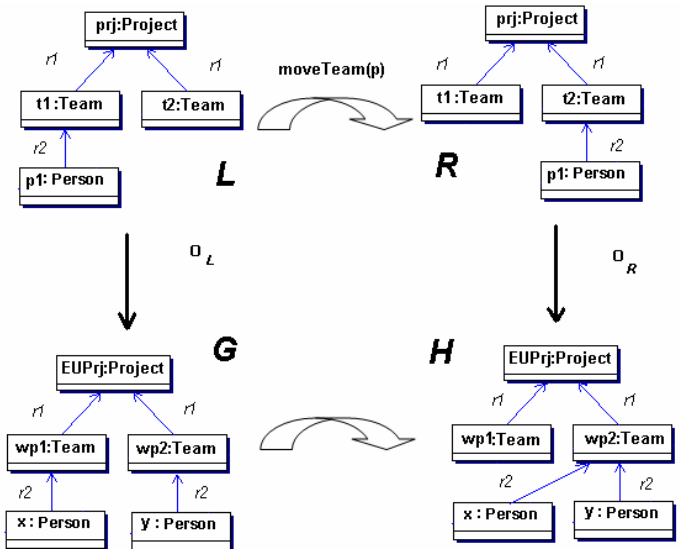


Fig 2: Type graph (previous page) and graph transformation step using rule *moveTeam.*

The rule *moveTeam(p)* specifies how a person *p1* moves from team *t1* to *t2*, both within a common project *prj*. Applying the rule, we are replacing an occurrence of *L* in *G* with a copy *R*, in three steps: (1) Find an occurrence o_L of *L* in *G*. (2) Delete all vertices and edges of G that are matched by *L*. (3) Paste to the result a copy of *R* to generate the new graph *H* [4].

Note that one of the most essential steps in this process is graph pattern matching; we will return our focus to this issue in due course by considering SPARQL queries on RDF/OWL documents to find patterns corresponding to the rule's left-hand side in an instance graph.

3.2 Modeling Technique and Context Representation

As a human-readable front-end, UML provides a visual mechanism for modeling both the static structure of a system and its snapshots at specific times. The UML class diagram is used to describe static concepts such as class, property and class-level relationships, including generalization and association, as well as cardinality constraints. In our approach, graph diagrams are seen as a visual representation of type graphs. Note that we are not using the full power of the language. For example, it is not necessary to define operations or visibility of properties since these cannot be represented in OWL. However, we make use of constraint. Assume that we would want to express two kinds of associations: "*is leader of*" and "*is colleague of*" both represented as self-association of class *Person.* But, as a matter of fact, they are quite different – "*is leader of*" is transitive while "*is colleague of*" is symmetric association. As we need to distinguish one from the other in terms of reasoning, the OCL (Object Constraint Language) can be used to enhance the class diagram as follows.

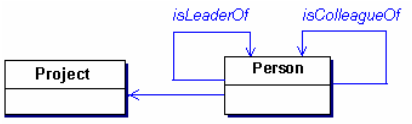


Fig.3: Class diagram (type graph)

```
Context Person inv:
 Person.allInstance->forAll(p1,p2.p2| p1.isLeaderOf=p2 & p2.isLeaderOf=p3
              implies p1.isLeaderOf=p3)
   Person.allInstance->forAll(p1,p2) |
p1.isColleagueOf=p2 implies p2.isColleagueOf=p1)
```

Table 1: OCL constraint for class diagram

This additional information is useful when instantiating the class diagrams. Also at implementation level, as an extension of RDF, OWL introduces the mechanism for describing property and association characteristics, e.g. the <owl:TransitiveProperty> in OWL Lite [5].

An instance of a class diagram is visualized by a UML object diagram, representing a system state and corresponding to an instance graph. Object diagrams are not directly used for modeling, except where fragments of a state are to be investigated, but transformation rules are displayed as pairs of object diagrams as seen in Fig. 3.

Since diagrams are not very suitable for automated processing, we use semantic web languages designed for machine-readable representation of data and documents on the web. Among these, RDF (Resource Description Framework) [7] is a framework recommended by W3C that describes web resources using *subject-predicate-object* triples. For example, *rabbit* (subject) *is a subclass of* (predicate) *mammal* (object). Consequently, RDF triples can be easily represented as vertices *V* and edges *E*, so that each edge *e (predicate)* in E connects a source vertex *(subject)* and a target vertex *(object).* In particular, we can present certain RDF triples as *instance-property-value* in UML object diagram. The OWL (Web Ontology Language) is built on top of RDF but adds more complex properties, characteristics and restrictions. A mapping between class diagrams and OWL can be defined as shown in Table 2 below [6].

UML concept	RDF/OWL concept [1]
class diagram	RDF/OWL Schema
object diagram	RDF/OWL document (instant of schema)
Basic data structure	built-in XML Schema datatypes
Class	<rdfs:Class> <owl:Class>

[1] Some features such as Object property, Transitive property, Symmetric Property, InverseFunctionalProperty , and advanced cardinality restriction are only support in OWL, which is an extension of RDF

property (attribute)	<rdf:Property>
general association	<rdf:Property>
source and target of the association	<rdfs:domain>< rdfs:range>
specified association (transitive association, symmetric association, etc) between classes	<owl:ObjectProperty> <owl:TransitiveProperty> <owl:SymmetricProperty> <owl: InverseFunctionalProperty >
class Inheritance (generalization)	<rdfs:subClassOf>,
N/A [2]	< rdfs:subPropertyOf>
cardinality, OCL restriction (Size-related)	<owl:cardinality> <owl:maxCardinality> <owl:minCardinality >
instance x of class X	<x rdf:ID='X'>
links (instance of association) between objects	instance of <rdf:Property> etc.

Table 2: Concept mapping between UML and RDF

Hence, OWL schemata provide a machine-readable representation to type graphs, while OWL instances correspond to instance graphs.

3.3 Pattern Matching Using SPARQL

As mentioned before, pattern matching is on of the major ingredients of graph transformation. Since OWL instances can be seen as graphs, subgraphs satisfying a certain pattern can be retrieved by executing a query on OWL. There are several OWL query languages available such as SPARQL [8] (SPARQL Protocol and RDF Query Language)[3] and. OWL-QL[4]. Jena is an implementation of SPARQL in Java, which provides a framework for building semantic web applications.

Before we demonstrate pattern matching in SPARQL, we present our sample context model as a class diagram. This does not claim to be complete, but captures the essentials that allow us to present the example and explain the techniques.

[2] UML does not explicitly support property inheritance.
[3] SPARQL syntax and specification : http://www.w3.org/TR/rdf-sparql-query/
[4] OWL-QL is designed at Stanford Knowledge System Laboratory

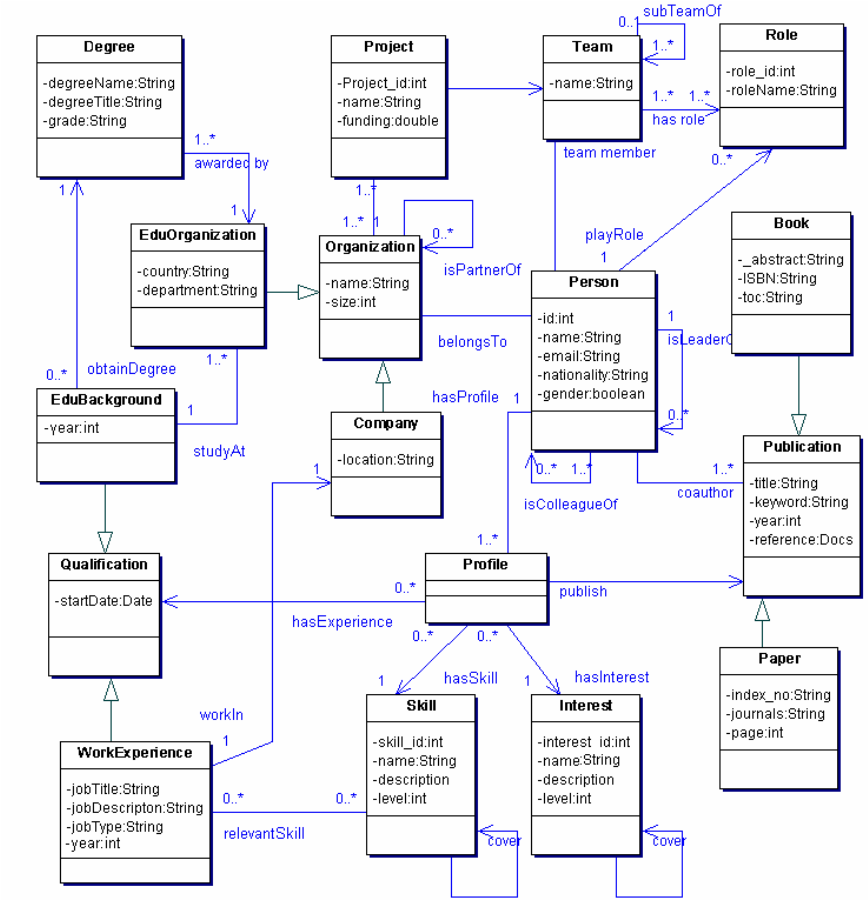


Fig. 4: Context model in class diagram (type graph)

Returning to our example, the software company needs to select a young person to join a new developing group. She should be under 30 and currently working as a programmer with more than 2 years experience in Java coding. We can define a SPARQL query on the OWL document representing an instance of our context model, to match this pattern in the instance graph below.

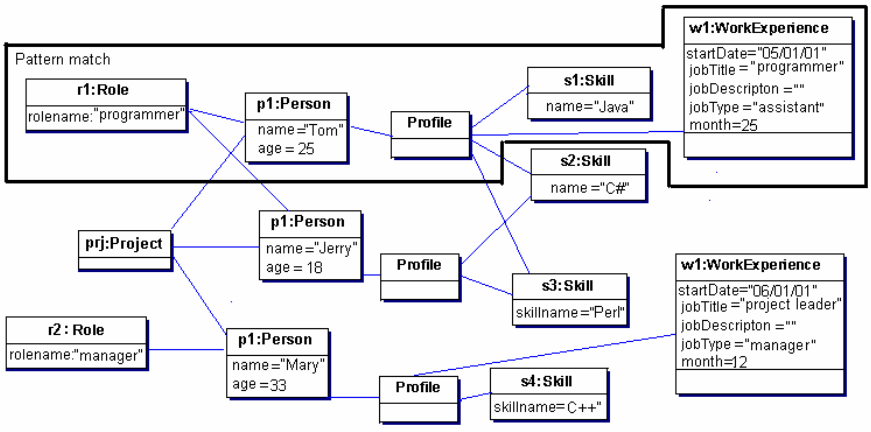


Fig. 5: Pattern matching in partial object diagram (instance graph)

```
PREFIX ns:<http://somewhere.owl#>

SELECT ?p
WHERE {
 ?p ns:playRole ?r.
 ?r ns:rolename "programmer".
 ?p ns:age ?z.
 ?p ns:hasProfile ?profile.
 ?profile ns:hasExperience ?experience.
 ?experience ns:month ?month.
 ?profile ns:hasSkill ?skill.
 ?skill ns:skillname "Java".
 FILTER(?z <=30 && ?month >=24)
 }
```

The SPARQL syntax is similar to other structured query languages such as SQL, the `WHERE` statement indicates a set of triple patterns. The result set will match when the triple pattern all match at the same time. By executing this SPARQL query on OWL instance we are able to detect the relevant subgraphs in an instance diagram that are candidates for context updates or reasoning by means of graph transformation. This is specified more formally in the next section.

3.4 Context Reasoning

It is noticeable that some information may not explicitly be presented in the defined context model. Thus, the problem is that how to derive additional information from given context data. In this paper, we concentrate on the “Rule-based deduction” reasoning method.

The context data is implied by that explicitly present. The rules can be given at metadata or the application level. At the metadata level they are based on the

metamodel (the metadata definition) as a type graph. Example includes the transitivity of subclass or subobject (composite) relations, as shown in the following example.

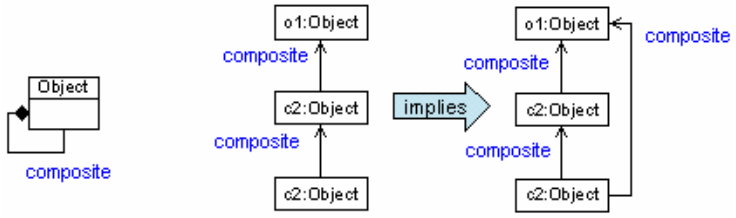


Fig. 6 Deduction-based example – Transitivity

At the domain or application level, where reasoning rules are specific to the problem at hand, they may be probabilistic or based on data obtained through data mining or statistical methods. The following rule (see Fig.7), however, is deterministic, stating that persons who are members of the same team are coworkers. (We use symbol ① *teamMmber*, ② *SubTeamOf*, ③ *isColleagueOf* stands for these associations)

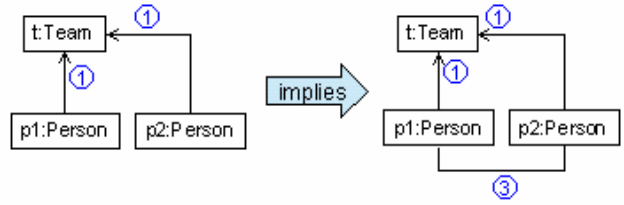


Fig. 7 Deduction-based example - Inference

Applying these two rules in sequence, the following deduction can be made as Fig. 8

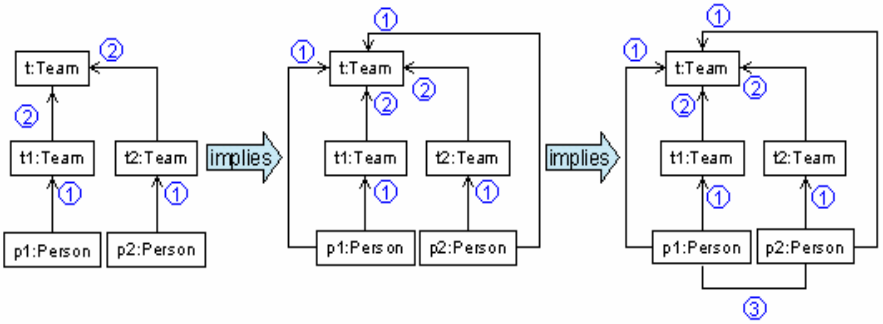


Fig.8 Deduction-based example

How to apply deductive rule in conjunction with SPARQL? One possible approach is to combine Jena inference engine with Jena ARQ. Assume we want to say if “Person a” and “Person b” are working in the same team, then we can apply the example deduction rule to get “a” and “b” are coworker. This can be defined by Jena rule syntax as follow:

[is_colleague_Imp: (?a belongsTo ?t),(?b belongsTo ?t)->(a? isColleagueOf ?b)]

The inference engine will apply the rules and subsequently add implicit triples to a new generated Inferred graph based on the original model. Then when we execute a SPARQL query to list all isColleagueOf triples, it should be able to return A and B even the links do not exist in the original model. So far, a prototype of reasoning component has been made and accessible via SOAP.

3.5 Team Formation as Graph Transformation

The team formation problem can be separated into two parts, the selection of team members and their actual assignment to the team. The first step corresponds to the graph matching inherent in the application of a transformation rule, while the second is represented by the actual application of the rule. Based on the context model in Fig. 4, seen as a type graph, the team members' context requirements can be given by an instance graph forming the left hand side *L* of the rule in Fig. 6. The graph R presents the intended replacement of this structure by the newly formed team.

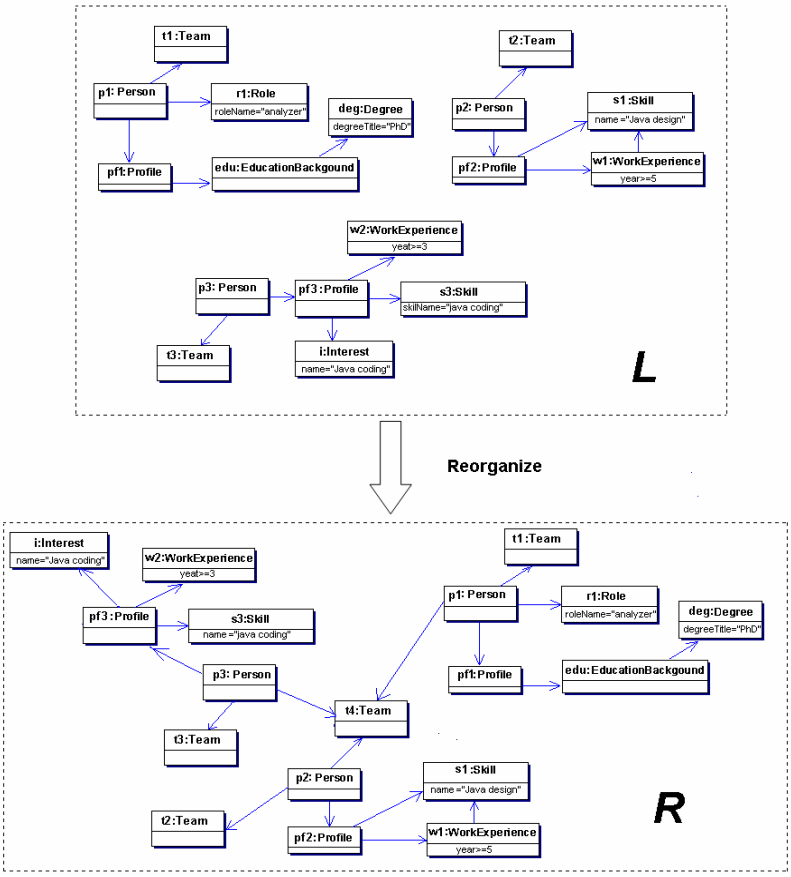


Fig. 6: Specifying the team formation problem by graph transformation rule *Reorganize*

Because the context model is based on an OWL ontology, we can use SPARQL for implementing the three patterns for the candidates' profiles as discussed in section 3.3. The patterns are treated as preferences for the selection. For example, the criteria for searching the correct analyzer are generated by the following query code.

```
   PREFIX ns:<http://somewhere.owl#>

   SELECT ?p
   WHERE {
    ?p ns:playRole ?r.
    ?r ns:rolename "analyzer".
 ?p ns:hasProfile ?profile.
 ?profile ns:hasExperience ?experience.
 ?experience ns:month ?month.
 ?profile ns:hasWorkExperience ?w
 ?w ns:jobTitle ?jt.
 FILTER(regex (?jt,"developer","i") || regex (?jt,"analyzer","i") || regex
(?jt," architecture","i"))
 FILTER(?month >=60)

    }
```

As result, we obtain values for the criteria "rolename", "workExperience.month", and "workExperience.jobTitle". SPARQL returns all candidates satisfying the requirements as given by the structural criteria. In the case of only one candidate for every position we could directly apply the rule and create the only team possible in the present state. However, most of the time we will have a number of suitable potential team members among which we must select the most suitable ones. This corresponds to selecting one of a number of possible occurrences for the application of a graph transformation rule, and this selection will be based on qualitative criteria. The next section realizes this based on a modified LSP method, guiding the graph transformation process by selecting the "best" occurrences for every rule application.

4 LSP Method for Ranking and Selection

LSP is a quantitative method based on scoring techniques and a continuous preference logic [9]. The method allows establishment of an evaluation criterion by specifying the expected properties of a system. To each one of these properties a criterion function is assigned. These functions transform specific domain values to a normalized scale indicating the degree of satisfaction of the corresponding preference. Then, all preference values can be properly grouped using a stepwise aggregation structure to yield a global preference. This can be achieved by means of a preference aggregation function, called *generalized conjunction/disjunction* or *andor*, combining weighted power means to obtain the global preference e_0 as in:

$$e_0 = \left(W_1 e_1^r + \ldots + W_k e_k^r\right)^{1/r}, W_1 + \ldots + W_k = 1 \quad (1)$$

Where the power r can be suitably selected to obtain desired logical properties (see [9, 10] for further details). However, the disadvantages of the method are that they require the input of a human expert, which is not suitable for working in a dynamic

environment [11]. We have defined a modified LSP method which is applicable to finding solutions for dynamic problems. In the rest of this section, we will introduce how to use this method to solve the quantitative aspect of our team formation problem.

4.1 Type-based Unified Evaluation Methods

In order to address dynamic problems such as ranking of structural matches, we have modified the original LSP method. The first change is in defining a unified evaluation method. With regard to the context model defined in section 3, we find that the context information is formatted as four types: Boolean ("gender"), String (such as "name"), Level (such as "skill.level"), and Number (such as "workexperience.year"). Thus, we identified four different evaluation functions to capture these four information types. The four functions are: "*exact match*" (equation 3), "*set overlap*" (equation 4), "*level match*" (equation 5) and "*specific value*" (equation 6).

Typical usage is linked to the data type of the context aspect: if the context aspect can be expressed by a Boolean or a simple string[5], then the exact match would be used; considering sets of information (complex string type), set overlap is useful; level match is useful for ordered discrete values (such as low, medium and high); and finally specific value allows for complex functions that calculate a numerical value (e.g. "workexperience.year"). Because of the link to the data type – which is apparent from the model – it can be automatically determined which function should be used. In addition, the weight $\omega < 0$ expresses that a lower value is desirable, while $\omega \geq 0$ means that a higher value is desired. Thus the global preferences evaluation function is changed from equation (1) to (2).

$$e_0 = (|\omega_1|E_1^r + |\omega_2|E_2^r + \ldots + |\omega_n|E_n^r)^{1/r} \text{ with } 0 \leq E \leq 1, \sum_{i=1}^{n} |\omega_i| = 1 \tag{2}$$

The respective formulas to compute values for these functions (E1 to En) are as follows:

$$E = \begin{cases} 1 & \textit{if criterion is met} \\ 0 & \textit{otherwise} \end{cases} \tag{3}$$

$$E = (e_1 + e_2 + \ldots + e_n)/n \text{ with } e_i \text{ being a score for each element of the set} \tag{4}$$

$$E = {}^{i_c}\!/_{i} \text{ where } i \text{ is the number of levels and } i_c \text{ is the current level match} \tag{5}$$

$$E = \begin{cases} 1 - \left(\dfrac{v_{\max} - v}{v_{\max} - v_{\min}} \right) & \textit{iff } \omega \geq 0 \\ \left(\dfrac{v_{\max} - v}{v_{\max} - v_{\min}} \right) & \textit{otherwise} \end{cases} \tag{6}$$

[5] Simple string means that only one kinds of string is filed in context attribute, for example the "name" of the people in the context model has to be one. Contrast to that the complex string is multi values for one attributes such as "qualifications".

$v_{\min}$ being the minimum value for all people, $v_{\max}$ the maximum value and v the value for the current people in (6).

4.2 Dynamic Logic Calculation

The second significant change is that we design an automated calculation method (ACM) to find a single logic GCD (Generalized Conjunction/disjunction) function based on Continuous Logic [12] for all preferences. For this to work, all weights of preferences sum up to 1. (The logic meaning can be reflected by a meaningful weight, details on deciding on which is out scope of this paper.) We consider the value of the weight ω_i belonging to a set $A, A \in (0,1)$. Then we have an ordered set W = $(\omega_{1,\ldots}\omega_n)$, and $\omega_1 \geq \ldots \geq \omega_n$. Based on the meaning of or-ness in the OWA decision making method [13], we can get the following function:

$$\lambda_{orness} = \frac{1}{n-1}\sum_{i=1}^{n-1}(n-i)|\omega_i|, \omega_i \text{ is the } i_{th} \text{ place in set V} \qquad (7)$$

Here V is the ordered set obtained from W be reordering according to the following algorithm: First, find weights $\omega_{1+1,\ldots,}\omega_{1+i}$ equal in value to ω_1 and put them to the tail of the set. Second, taking all $\omega_{2+1}\ldots\omega_{2+i}$ which have the same value as ω_2 in the new set in front of the $\omega_{n-i}\ldots\omega_n$. Repeat the second step until the last element that has not been reordered before. For example, if W = {0.2, 0.2, 0.15, 0.15, 0.1, 0.1, 0.1}, then, V = {0.2, 0.15, 0.1, 0.15, 0.1, 0.1, 0.2} the value λ presents the degree of the "or-ness" as computed by equation (7). The relation between the value of r and the value of λ is shown in Table 3.

Value of λ	GCD Operator symbols		Operation
$\lambda < 0.3333$	GEO	$r = 0$	Geometric mean
$0.3333 \leq \lambda < 0.3750$	C-	$r = 0.2$	Weak QC
$0.3750 \leq \lambda < 0.4375$	C--	$r = 0.5$	Weak QC (-)
$0.4375 \leq \lambda < 0.5000$	A	$r = 1$	Arithmetic mean
$0.5000 \leq \lambda < 0.5625$	D--	$r = 1.5$	Weak QD (-)
$0.5625 \leq \lambda < 0.6232$	SQU	$r = 2$	Square mean
$0.6232 \leq \lambda < 0.6250$	D-	$r = 2.3$	Weak QD
$0.6250 \leq \lambda$	D-+	$r = 3$	Weak QD (+)

Table 3: Relation between the value of r and the value of λ

4.3 Example for Applying the Modified LSP Method

Assume that there are two analyzers satisfying all criteria, but one of them (analyzer_1) has 8 years of work experience and a qualification of "analyzer". The other one (analyzer_2) has 5 years experience and two qualifications of "developer" and "analyzer". Additionally, the weight for experience is 0.7 while that for qualification is 0.3. The or-ness can be calculated as $\frac{1}{2-1}((2-1)\times 0.7+(1-1)\times 0.3)=0.7$. Therefore the logical power of r should be 3 (as per Table 3). Then, the scoring algorithm provides the following results:

analyzer_1= $(0.7\times(1-\frac{8-8}{8-5})^3+0.3\times(\frac{1}{3})^3)^{1/3}=0.2370$ and

analyzer_2= $(0.7\times(1-\frac{8-5}{8-5})^3+0.3\times(\frac{2}{3})^3)^{1/3}=0.0296$.

These results show that analyzer_1 is clearly preferable to analyzer_2.

6 Conclusion and Future Work

In this paper, we have proposed a process for solving the problem of selecting people to form teams based on context information. The key technologies used are OWL and SPARQL retrieving context at the implementation level. However, more interestingly methods at the abstract level have been defined, like the use of graph transformation to define reasoning rules, structural matching and updates and its enhancement by the LSP to select suitable occurrences of application according to qualitative criteria.

While we have concentrated on a rather narrow example here, we would like to point out that this highlights many aspects that are common to selection problems and the method is applicable in other domains such as service selection in SoC, an area where context is also very important due to the possibility of late binding of services.

This paper lays a proof-of-concept study for an area that we will investigate further, which is the enhancement of graph transformation techniques with methods to select "best" candidate matches.

More practically, we are going to work on the methodology for specifying the weight for each criteria and separating crucial preferences and desirable preferences by defining more context information. Furthermore, we will also continue to work on the theory of verifying the result by using graph transformation.

7 Acknowledgment

This work is supported by inContext **(**Interaction and Context Based Technologies for Collaborative Teams) project: *IST IST-2006-034718.*

References

[1] Anind K. Dey, Understanding and Using Context, Future Computing Environments Group, College of Computing & GVU Center, Georgia Institute of Technology, USA, 2000.

[2] Heckel R., http://www.gratra.org/

[3] Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages and Tools H. Ehrig, G. Engels, H-J Kreowski, Grzegorz Rozenberg, 1997, World Scientific Publishing Co. Ptc. Ltd, ISBN: 981-02-4020-1.

[4] R. Heckel (2006), Graph Transformation in a Nutshell, Electronic Notes in Theoretical Computer Science, pp. 187–198

[5] M. Deborah L., H. Frank van (2006) "OWL Web Ontology Language Overview" W3C Recommendation, http://www.w3.org/TR/owl-features/

[6] C Walter W. (1998) "A Discussion of the Relationship Between RDF-Schema and UML",W3C Note, http://www.w3.org/TR/NOTE-rdf-uml/

[7] Prud'hommeaux, E., Seaborne, A.(2006), "SPARQL query language for RDF". W3C Working Draft, http://www.w3.org/TR/rdf-sparql-query/

[8] B. Dan, G. R.V., M. Brian(2004), RDF Vocabulary Description Language 1.0: RDF Schema, W3C Recommendation,http://www.w3.org/TR/rdf-schema/

[9] Dujmovic J.J., Continuous Preference Logic for System Evaluation. In Proceedings of Eurofuse 2005, edited by B. De Baets, J. Fodor, and D. Radojevic, ISBN 86-7172-022-5, Institute “MihajloPupin”, Belgrade, 2005, pp. 56-80.

[10] Dujmovic J.J., A Method for Evaluation and Selection of Complex Hardware and Software Systems. The 22nd International Conference for the Resource Management and Performance Evaluation of Enterprise Computing Systems. CMG 96 Proceedings, Vol. 1, 1996, pp. 368-378.

[11] Levin V. I., Generalizations of the Continuous Logic. Automation and Remote Control, Vol. 62, No. 10, 2001, pp. 1743-1755.

[12] Yu H.Q. and Molina H., A Modified LSP method for services evaluation and selection, under editing, The 2nd European Young Researchers Workshop on Service Oriented Computing, 11-12 June 2008.

[13] Robert F., OWA operators in Decision Making. In C. Carlsson ed., Exploring the limits of Support Systems, TUCS General Publications No. 3, Turku Centre for Computer Science, 1996, pp. 85-104.