Generating Human-Computer Micro-Task Workflows from Domain Ontologies

Nuno Luz[1], Nuno Silva[1], Paulo Novais[2]

[1] GECAD (Knowledge Engineering and Decision Support Group), Polytechnic of Porto
{nmalu,nps}@isep.ipp.pt
[2] CCTC (Computer Science and Technology Center), University of Minho
pjon@di.uminho.pt

Abstract. With the growing popularity of micro-task crowdsourcing platforms, a renewed interest in the resolution of complex tasks that require the cooperation of human and machine participants has emerged. This interest has led to workflow approaches that present new challenges at different dimensions of the human-machine computation process, namely in micro-task specification and human-computer interaction due to the unstructured nature of micro-tasks in terms of domain representation. In this sense, a semi-automatic generation environment for human-computer micro-task workflows from domain ontologies is proposed. The structure and semantics of the domain ontology provides a common ground for understanding and enhances human-computer cooperation.

Keywords: Human-Machine Computation, Micro-Task Workflows, Ontologies

1 Introduction

With the emergence of micro-task crowdsourcing platforms such as CrowdFlower and Mechanical Turk, human computation has gained renewed interest in the resolution of complex tasks that require the cooperation of human and machine participants [1–6]. This interest has led to several approaches built upon workflows of micro-tasks.

Micro-task workflows present new challenges at different dimensions of the human-machine computation process, namely in micro-task specification and human-computer interaction [1, 2]. The unstructured nature of micro-tasks in terms of domain representation makes it difficult (i) for task requesters not familiar with the crowdsourcing platform to build complex micro-task workflows and (ii) to include machine workers in the workflow execution process [7]. Furthermore, it is seldom explicitly defined that while some of the micro-tasks in the workflow are better performed by humans, others are better performed by a machine.

Obrst et al. [8] state that ontologies "represent the best answer to the demand for intelligent systems that operate closer to the human conceptual level". Considering this, this paper presents a semi-automatic generation environment for human-computer micro-task workflows from domain ontologies. The inherent process relies

in the domain expertise of the requester to supervise the automatic interpretation of the domain ontology. The structure and semantics of the domain ontology enhances human-computer cooperation and allows the automatic generation (with included contextual information) of preliminary micro-task worker interfaces using a markup language.

This paper is organized as follows. Section 2 provides some background knowledge on micro-task workflow approaches and ontologies. It is followed by, in section 3, the presentation of the overall architecture behind the proposed micro-task workflow generation approach. Section 4 presents the generation process through a running example. Finally, the conclusions are given along with some remarks on the future directions of this work.

2 Background Knowledge

2.1 Human Computation and Micro-Task Workflows

Several experiments in different domains have shown that human computation (in particular micro-task crowdsourcing) has great potential for solving large scale problems that are often difficult for computers to solve automatically, on their own [9]. These problems usually require a degree of creativity or just common sense plus some background knowledge [10, 11]. The interpretation and recognition of images and natural language are two examples of these kinds of problems.

Crowdsourcing platforms like Mechanical Turk, CloudCrowd, ShortTask and CrowdFlower are widely used for tasks such as (i) categorization and classification, (ii) data collection (e.g., finding a website address), (iii) moderation and tagging of images, (iv) surveys, (v) transcription from multimedia content (e.g., audio, video and images), and (vi) text translation.

In particular, micro-task workflow approaches like CrowdForge, Jabberwocky and Turkomatic employ divide and conquer and map reduce strategies to build workflows. This usually involves workflows that include tasks for (i) the partitioning of the complex task (partition tasks), (ii) the execution of the partitioned tasks (map tasks), and (iii) the aggregation of results (reduce tasks).

However, the input given by workers, in several cases, is unstructured and in natural language. Furthermore, micro-task interfaces are built using markup languages that contain little if no meta-data, making it difficult for machine micro-tasks to be included in the workflow.

The terminology employed in the crowdsourcing domain often varies from platform to platform. In the context of this paper, the following terms and entities are considered:

- Worker – a person that solves tasks;
- Community – a set of workers;
- Job – a complex task or workflow of tasks;
- Task (or micro-task) – a definition of a concrete computation or operation that may be performed by workers;

- Requester – an entity (typically a person) that submits jobs;
- Unit – an input of a task;
- Reference Unit – an input of a task for which the output is already known;
- Assignment – an assignment of a unit to a single worker;
- Answer – the given solution of a worker to a specific assignment;
- Workflow – the continuity of work by passing the output of one task as the input of another.

2.2 From Domain Ontologies to Micro-Task Workflows

In this work, Description Logics (DL) knowledge bases and ontologies are considered. A DL knowledge base is defined as containing both a TBox (terminological box) and an ABox (assertion box), where the TBox contains all the concepts and relationships that define a specific domain, and the ABox contains the instances or individuals defined according to the elements in the TBox [12]. It is assumed that ontology is synonym of TBox.

In the TBox, a set of concepts (or classes) and properties exist. Each concept has associated property restrictions that define the necessary, and necessary and sufficient conditions for an individual to be an instance of the concept. These conditions may be enforced according to two main types of properties: object properties and data-type properties. While object properties relate instances (or individuals) with other instances, data-type properties relate instances with "primitive" type values (e.g., string, integer, double, date, time).

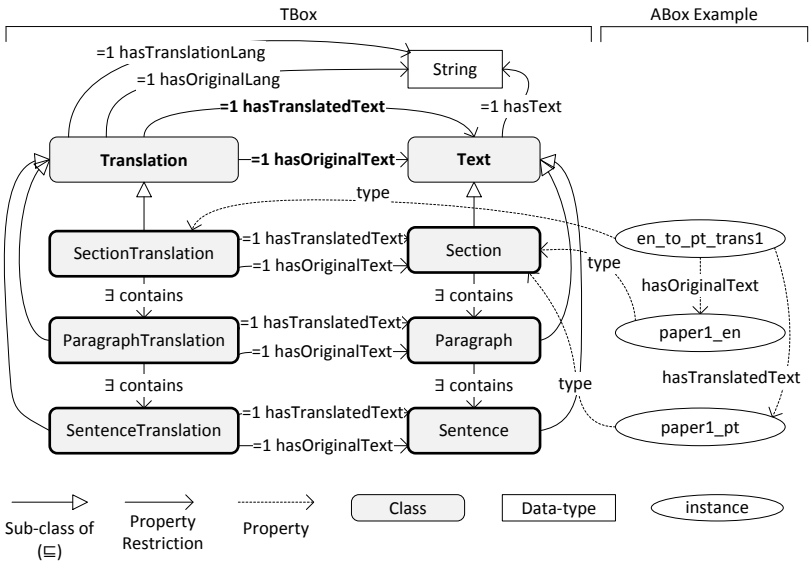

Fig. 1. The article translation ontology (TBox only) with ABox example.

Consider the translation ontology presented in **fig. 1**, where each rectangle represents an instance and each ellipse represents a class. Arrows in the TBox represent property restrictions and dashed arrows are actual property relationships in the ABox.

3 Three-Layered Workflow Generation

Micro-tasks, whether they involve physical actions or not, can be seen as a process that, in a specific context, results in the emergence of new data (answers) from the presentation of other particular pieces of data (units) to a worker. Analogously, a workflow of micro-tasks is the continuous ordered increment of new (different types of) data, in a specific context or domain.

The context (or domain) can be defined and delimited through domain ontologies, modelling all input and output data. Thereafter, a micro-task can be considered to be *the instantiation of classes and specification of new relationships between instances* according to the domain ontology. A workflow of micro-tasks is then considered as *the incremental instantiation of the domain ontology according to its structure and semantics*.

In order to harness the power of ontologies in micro-task workflows, an iterative semi-automatic workflow generation process is proposed. This process is based in a layered architecture that defines the set of operations that can be performed by micro-tasks on top of the ontology data (see **fig. 2**).

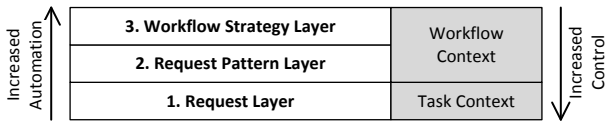

Fig. 2. The layered architecture of the generation process.

The request layer defines the set of possible atomic operations that can be performed over the ontology and workflow data. In this layer, a low-level structural analysis of the ontology is performed.

A request pattern, in the request pattern layer, is a set of requests associated to a specific ontological pattern [13, 14]. Request patterns often depend on the employed ontology construction methodology [15]. In some cases, though, they may also depend on the domain of the ontology. In the request pattern layer, mostly high-level structural and low-level semantic analyses of the ontology are performed.

The workflow strategy layer is an abstraction over the previous base layers. Each workflow strategy represents a subset of requests and request patterns, often found in specific workflow domains. They automate the process by restricting the set of possible choices presented to the requester during the workflow extraction process. For instance, a workflow strategy for recommendation workflows could restrict the possible choices of the requester through its set of possible request patterns and requests.

Through the workflow strategy layer, a high-level semantic analysis of the ontology is performed.

3.1 The Request Layer

A request is always associated with a workflow step and defines the operation to be performed. Multiple types of requests can be performed. They can be classified according to their operation (see fig. 3) and structure. A request can be of multiple non-disjoint types.

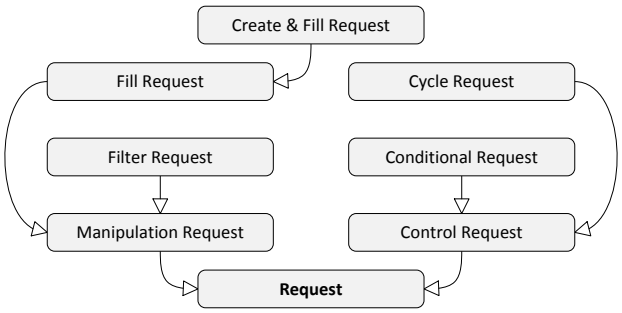

Fig. 3. Classification of requests according to their operation.

In terms of operation, there are two main request types: (i) manipulation requests and (ii) control requests. On the one hand, manipulation requests (i) represent explicit machine or human micro-tasks in the workflow that perform some kind of operation over ontological instances in the ABox, according to the TBox. These can be creating instances and specifying their properties (Create & Fill Request), just specifying or completing the properties of existing instances (Fill Request), or filtering existing instances (Filter Request). On the other hand, control requests (ii) (e.g., conditional blocks, cycles) establish flow control components in the workflow.

Table 1. Structure of a request and corresponding classification.

Symbol	Component (Structure)	Classification
C	An ontology class (or concept)	Concept Request
$Rest$	A set with at least one restriction	Restricted Request
P	A path of relations (property restrictions)	Path Request
$NPRel$	A non-built relation in the path	Relation Request
LP	At least one link path	Link Path Request
I	A cycle	Cycle Request
$Cond$	A condition	Conditional Request

A request is a tuple $RQ(C, Rest, P, NPRel, LP, I, Cond)$ as presented in table 1. The existence of each of the presented components is directly related to the classification in **table 1**. The ontology class defines the object of the operation. For example, new instances of a specific class can be requested through a Create & Fill Concept Request.

In some situations, the ontology class is not expressive enough to restrict the set of instances to be affected or created by the operation. A Restricted Request specifies property values that must be present for instances of the ontology class. If an instantiation is performed, these property values will automatically be set and enforced. Otherwise, if instances are already present, those that do not contain these property values will be filtered from the operation.

Often, the target of a request is not the whole set of instances of an ontology class, but only those related to instances of other classes and so on. These relations form a path of relationships between named classes that represent the context in which the operation will be performed. Requests that include a path from the ontology graph, typically ending in the requested ontology class, are Path Requests.

Path Requests contain relations that are either missing (one, at most) or present in the built graph. If a new relation (not in the built graph) is requested, the request is classified as a Relation Request (see **fig. 4**).

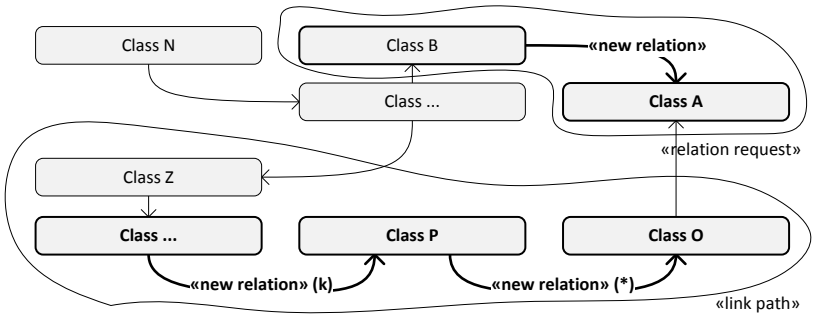

Fig. 4. Structure of a Relation Link Path Request (k and * are fixed and multiple cardinality values, respectively).

While Path Requests and Restriction Requests capture operations upon all related entities surrounding the ontology class (or its context), it may also be useful to instantiate secondary paths in order to establish relationships that, otherwise, would be lost. These secondary paths are called link paths, as depicted by the link path area in **fig. 4**.

Link paths may contain the instantiation of at most one class, and the specification of at most two new relations. If two new relations are required, at least one must have a known fixed exact cardinality. If this is not verified, a combinatorial explosion of relationships problem, which falls outside the context of an atomic operation, occurs.

Without link paths, additional secondary paths to instances generated by a specific request would not be possible, since it is not possible to select only instances generated by a previous request.

3.2 The Request Pattern Layer

The request pattern layer establishes a correspondence between an ontology pattern and a set of requests. These correspondences are called request patterns and provide automation over the direct usage of requests.

Ontology patterns can be found in several different ontologies describing a variety of domains. In the case of micro-task workflows, several approaches exist that try to employ divide and conquer or map-reduce strategies. These strategies focus on dividing the task at hand, executing the resulting units, and assembling the results. Such strategies can also be employed through the definition of partition and assembly request patterns.

Both partition and assembly request patterns can be defined through meronymic (part-of relation) ontology patterns. While the partition request pattern picks requests using a top-down search following meronymic relationships, the assembly request pattern picks requests using a bottom-up search.

3.3 The Workflow Strategy Layer

The workflow strategy layer aggregates multiple request patterns, further automating the workflow construction process.

Using both the partition and assembly requests patterns previously described a divide and conquer workflow strategy can be specified. Considering that the input instances supplied by the requester have meronyms, for which an output that can be assembled exists, both request patterns can be employed to form a full workflow. **Fig. 5** depicts the application of the divide and conquer workflow strategy for a translation ontology.

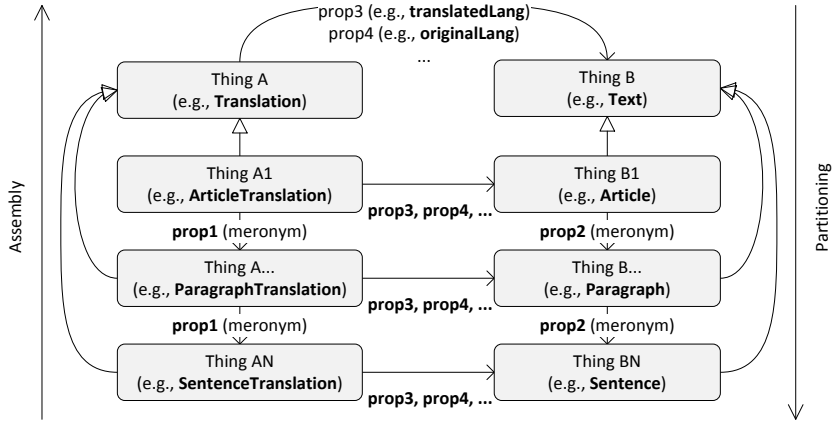

Fig. 5. Application of the divide and conquer workflow strategy with both partition and assembly request patterns.

4 The Generation Process

The workflow generation process consists in four steps: (i) ontology and input specification, (ii) input pre-processing, (iii) iterative construction and (iv) post-processing. Each of these steps will be described through a running example, which consists in the construction of a workflow for translating articles or papers.

During both the iterative construction step (iii), a task pattern detection algorithm, which acts according to the defined strategy, is triggered. If a pattern (or sequence of patterns) is detected, the requester may choose to apply the pattern, automatically adding several micro-tasks to the workflow.

4.1 Input Specification and Pre-processing

The process starts with the ontology and input specification (i) by the requester. The input includes the initial ABox (instances and relationships) fed to the workflow and the ontology describing the domain (e.g., the translation ontology in **fig. 1**). Typically, input instances contain data describing the initial task that will be supplied to workers.

From these data, two graphs are extracted in the input pre-processing step (ii): the ontology graph and the built graph. The ontology graph is fully instantiated during this step and contains nodes that represent named classes in the ontology. Each edge represents a property restriction that relates two named classes and contains relevant information about the relationship such as the type of restriction (e.g., existential quantification, universal quantification, cardinality) and corresponding cardinality. The built graph will contain all classes and relations requested during the iterative construction step (iii). The built graph is partially instantiated during this step with the classes and properties found in the input data.

For instance, the input in table **table 2** will result in a built graph containing the classes Article and ArticleTranslation, and the object property restriction originalText relating ArticleTranslation with Article.

Table 2. Input statements/triples for the translation ontology use case.

Instance	Property	Value
onto:paperTranslation1	rdf:type	onto:ArticleTranslation
onto:paperTranslation1	onto:originalLang	“Portuguese”
onto:paperTranslation1	onto:translatedLang	“English”
onto:paperTranslation1	onto:originalText	onto:paper1
onto:paper1	rdf:type	onto:Article
onto:paper1	onto:lang	“Portuguese”

4.2 Iterative Construction

Using both built and ontology graphs, the iterative construction step (iii) presents the requester with a set of possible choices (requests, request patterns or workflow strategies). As the requester iteratively picks one of these choices, new classes and relations (present in the chosen requests) are added to the built graph. Ultimately, the built graph becomes a clone of the ontology graph.

As requests are picked by the requester, they are added to the request graph. Edges in this graph define the dependencies between requests.

During the first iteration of the article translation example, the requester can opt to either request the entire translation of the article (using only the top-level classes Translation and Text, or ArticleTranslation and Article), or to partition the article before requesting translations. This partitioning of the task may be useful for articles with many paragraphs.

For the entire translation of the articles given as input, one request would suffice, that is, a Create & Fill request for Article instances related to ArticleTranslation through the onto:translatedText property. Notice that, during the workflow construction, the requester deals only with the TBox (ontology classes and properties). The ABox (instances) will only be handled during the execution of the workflow. The initial input ABox is an exception to this rule, with the purpose of facilitating the specification of the input TBox.

If the requester opts to partition the article into paragraphs, two possible request choices are presented (see **table 3**).

Table 3. Possible requests for the ontology class Paragraph in the translation example - first iteration of the iterative construction step (new relation in path in bold).

#	Operation	Path
1	Create & Fill	-
2	Create & Fill	ArticleTranslation - originalText - **Article - part - Paragraph**

Request 1 is always presented for any ontology class. It results in instances of Paragraph, completely unrelated to already existent instances. Request 2 is only possible because the path ArticleTranslation - originalText - Article already exists in the built graph (added during the input specification step).

For a specific Path request, a set of possible link paths is usually presented. In the case of request 2, the following possible link paths exist (notice how the start and end in classes present in the request path):

1. ArticleTranslation - translatedText - Article - part - Paragraph
2. ArticleTranslation - part - ParagraphTranslation - originalText - Paragraph
3. ArticleTranslation - part - ParagraphTranslation - translatedText - Paragraph

Picking one of these link paths will result in a connection from onto:paperTranslation1 to all instantiated Paragraphs through an intermediary “empty” (no other properties will be specified) instance. As instances of ParagraphTranslation

are required in order to request the actual translation of paragraphs in future iterations, all link paths will be excluded except for link path 2.

Up to this point, the requester has built a workflow (with only one micro-task) that partitions articles into paragraphs. Further partitioning of paragraphs into sentences is also possible by repeating the previous choice pattern.

In the second iteration, the requester can finally request translations of paragraphs. The possible request choices for this iteration are presented in **table 4**.

Table 4. Possible requests for the ontology class Paragraph in the translation example - second iteration of the iterative construction step (new relation in path in bold).

#	Operation	Path
1	Create & Fill	-
2	Create & Fill	ArticleTranslation - part - **ParagraphTranslation - translatedText - Paragraph**
3	Filter	ArticleTranslation - part - ParagraphTranslation - originalText - Paragraph

As the requester picks the request 2, only one possible link path is presented: ArticleTranslation - translatedText - Article - part - Paragraph. This link path must be selected, otherwise no relation between the translated Article and its corresponding Paragraphs will be set.

After requesting paragraph translations, the requester can finally request the translation of the article (third iteration). **Table 5** contains the possible requests of this iteration.

Table 5. Possible requests for the ontology class Article in the translation example - third and final iteration of the iterative construction step.

#	Operation	Path
1	Create & Fill	-
2	Fill	ArticleTranslation - translatedText - Article
3	Filter	ArticleTranslation - translatedText - Article
4	Fill	ArticleTranslation - originalText - Article
5	Filter	ArticleTranslation - originalText - Article

Picking request 2 will add a micro-task to the workflow, requesting workers to fill the translated article data-type property data, which includes the actual translated text. All related instances are given as contextual information, meaning that workers will have access to all related translated paragraphs and properties of the Translation instance.

4.3 Post-processing

The post-processing step (iv) is executed after the requester decides to conclude the iterative construction step. It outputs the workflow structure after applying a transitive reduction algorithm over the edges of the request graph.

For the given running example, it results in a sequential workflow with three micro-tasks (see **fig. 6**).

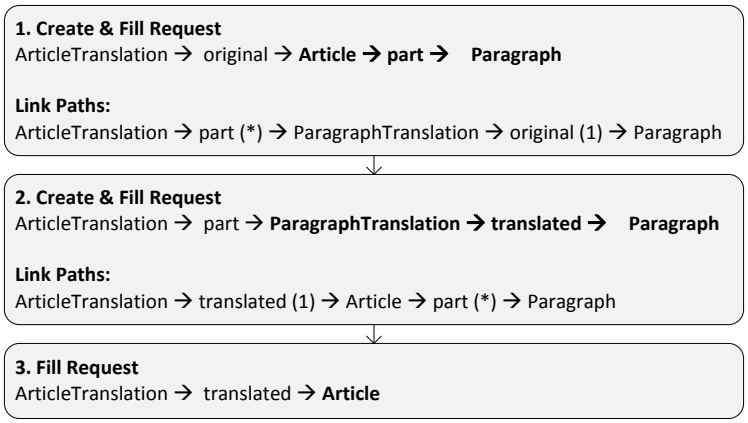

Fig. 6. Request workflow generated from the translation ontology.

5 Conclusions and Future Work

The proposed process tackles the challenge of assisting requesters in building complex micro-task workflows while promoting human-machine cooperation through high-level, declarative and semantically explicit domain ontology models. This is achieved through a semi-automatic micro-task workflow generation process that filters and proposes the following steps in a workflow according to pattern analysis and the context given by previous tasks.

The current prototype of this process possesses a simple command line interface and allows the implementation of different adapters that interact with different crowdsourcing platforms such as CrowdFlower.

Future work includes the continuous analysis of several ontologies, which may result in the identification of new relevant task patterns and strategies. Also, a focus to the automatic generation of micro-task worker interfaces from the domain ontology and context must be given. Evolving the prototype implementation and providing a friendly and efficient graphical user interface implementation, instead of the current command line interface, is also part of the future work.

Acknowledgements. This work is partially funded by FEDER Funds and by the ERDF (European Regional Development Fund) through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT

(Portuguese Foundation for Science and Technology) under the projects AAL4ALL (QREN13852) and FCOMP-01-0124-FEDER-028980 (PTDC/EEI-SII/1386/2012).

References

1. Ahmad S, Battle A, Malkani Z, Kamvar S (2011) The jabberwocky programming environment for structured social computing. Proc. 24th Annu. ACM Symp. User Interface Softw. Technol. pp 53–64
2. Kittur A, Smus B, Khamkar S, Kraut RE (2011) Crowdforge: Crowdsourcing complex work. Proc. 24th Annu. ACM Symp. User Interface Softw. Technol. pp 43–52
3. Kulkarni AP, Can M, Hartmann B (2011) Turkomatic: automatic recursive task and workflow design for mechanical turk. Proc. 2011 Annu. Conf. Ext. Abstr. Hum. Factors Comput. Syst. pp 2053–2058
4. Little G, Chilton LB, Goldman M, Miller RC (2010) Turkit: human computation algorithms on mechanical turk. Proc. 23nd Annu. ACM Symp. User Interface Softw. Technol. pp 57–66
5. Luz N, Silva N, Maio P, Novais P (2012) Ontology Alignment through Argumentation. 2012 AAAI Spring Symp. Ser.
6. Sarasua C, Simperl E, Noy NF (2012) CrowdMap: crowdsourcing ontology alignment with microtasks. Springer, pp 525–541
7. Quinn AJ, Bederson BB (2011) Human computation: a survey and taxonomy of a growing field. Proc. 2011 Annu. Conf. Hum. Factors Comput. Syst. pp 1403–1412
8. Obrst L, Liu H, Wray R (2003) Ontologies for corporate web applications. AI Mag 24:49.
9. Von Ahn L (2009) Human computation. 46th ACMIEEE Des. Autom. Conf. pp 418–419
10. Chklovski T (2003) Learner: A System for Acquiring Commonsense Knowledge by Analogy. Proc. 2nd ACM Int. Conf. Knowl. Capture. Sanibel Island, FL, USA, pp 4–12
11. Singh P, Lin T, Mueller ET, et al. (2002) Open Mind Common Sense: Knowledge Acquisition from the General Public. Move Meaningful Internet Syst. 2002 CoopIS DOA ODBASE. Springer, pp 1223–1237
12. Baader F (2003) The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press
13. Hammar K, Sandkuhl K (2010) The State of Ontology Pattern Research: A Systematic Review of ISWC, ESWC and ASWC 2005-2009. Workshop Ontol. Patterns Pap. Patterns ISWC Workshop. Shangai, China, pp 5–17
14. Gangemi A (2005) Ontology Design Patterns for Semantic Web Content. Semantic Web – ISWC 2005. Springer, pp 262–276
15. Blomqvist E (2007) OntoCase - A Pattern-Based Ontology Construction Approach. In: Meersman R, Tari Z (eds) Move Meaningful Internet Syst. 2007 CoopIS DOA ODBASE GADA IS. Springer, pp 971–988