Distributed Linked Data as a Framework for Human-Machine Collaboration

Paolo Pareti *

University of Edinburgh, Edinburgh, United Kingdom
Taiger, Madrid, Spain
paolo.pareti@taiger.com

Abstract. This paper presents a novel application of Linked Data as an indirect communication framework for human-machine collaboration. In a decentralised fashion, agents interact by publishing Linked Data resources without having access to a centralised knowledge base. This framework provides an initial set of solutions to the problems of dynamic Linked Data discovery, of querying frequently-updated distributed datasets and of guaranteeing consistency in the case of concurrent updates. As a motivation for this framework we take the use-case of human and machine agents collaborating to the execution of tasks. This use-case is based on existing real-world Linked Data representations of human instructions and research on their integration with machine functionalities.

1 Introduction

On the web, the amount of structured information available to machines is steeply increasing. This information can be used by computer systems to answer complex queries about factual knowledge, for example about the population of cities and the date of birth of notable people. However, machines still have little or no understanding of human activities. For example, a user who is in the process of performing a complex task might use the functionalities offered by multiple software tools. However, these systems might operate in isolation, not knowing what the user is trying to achieve and how. This lack of understanding is a limitation to human-machine collaboration as machines cannot predict when and how their functionalities might be needed.

A typical approach to describe activities is by writing instructions. Previous research has demonstrated how human instructions can be converted into Linked Data and how related tasks and entities can be automatically interlinked [5]. While certain steps of the instructions can only be executed by humans, others, such as sending emails or modifying files, can be automated. Such steps can be linked to machine functionalities and executed at the right time when a user is performing a related activity [4]. This paper generalises this approach

* This research has been funded by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 607062 ESSENCE: Evolution of Shared Semantics in Computational Environments.

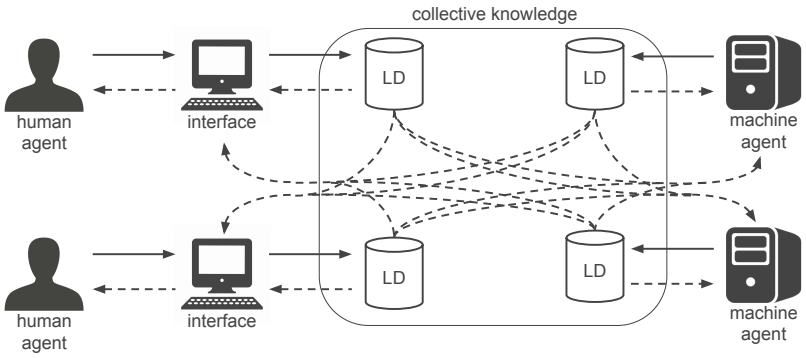

Fig. 1. A multi-agent environment with no shared editable resources. Dashed and solid lines represent, respectively, the ability to read and write Linked Data resources.

to automation to the case of multiple agents and decentralised resources. As its main contribution, this paper presents a novel application of Linked Data as an indirect collaboration framework for humans and machines. The main benefits and limitations of this framework are discussed and initial solutions are proposed to the problems of dynamic Linked Data discovery, of querying frequently-updated distributed datasets and of guaranteeing consistency in the case of concurrent updates by multiple agents.

2 Problem Description

The proposed framework addresses the problem of allowing a collaborative set of human and machine agents, who can publish and access web resources but that cannot directly interact with each other, to *communicate* and *coordinate* their actions to collaboratively achieve tasks in the absence of a centralised system. To achieve decentralisation, no shared resource which multiple agents can edit is assumed to be available. As depicted in Figure 1, each agent is able to modify resources in its own repository while it is able to read the resources in all of the repositories of the other agents. In this context, *communication* refers to the process by which agents can propagate information (i.e. RDF[1] triples) to the other agents by modifying the *collective knowledge*, namely the resources that all agents can access. *Coordination* instead refers to the ability to guarantee certain conditions across all datasets. For example, coordination might be required to ensure that no agent can declare its intention to execute a task which is already being executed by another agent.

This framework is intended to address *dynamic partially-automatable* problems. A partially-automatable problem requires some human intervention, as it cannot be entirely automated. At the same time, it cannot be optimally solved by humans alone, as some part of it could be automated. Dynamic problems, instead, are those that cannot be predicted in advance, making the development

[1] http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/

of dedicated software solutions impractical. To simplify the problem at hand, it is assumed that the agents involved already know and trust each other. Issues such as agent discovery, coalition formation and trust are considered outside the scope of this project.

3 Motivational Use-Case

As a running example we take the use-case of John, an employee of a company working on a project alongside with Ann. Every week, John and Ann write a report about the status of the project which then gets sent to the other members of the company. To do so, they agree on the following procedure:

Weekly report procedure:
Step 1: John writes a draft of the report
Step 2: The report is uploaded to an online repository
Step 3: Ann corrects the report
Step 4: The report is sent to all the colleagues

While this procedure can be completed manually, it can also be made more efficient with automation. For example, as soon as John finishes writing the draft of the report, a machine agent could upload it to the correct repository. Ann could then be automatically notified that she can start correcting the report as step 3 is ready to be executed. Step 4 might also be a good candidate for automation. Different colleagues might have different preferences on how to be contacted. For example, some might prefer to receive an email, others a message on an instant messaging system or a social networking platform. A machine agent could deal with these diverse preferences automatically, provided its given the necessary contact details.

4 Methodology

Lacking any central system, agents are the only components of the proposed framework. At a functional level of abstraction, all agents can be seen as entities capable of reading and writing Linked Data resources. Agents also have individual repositories where they can publish and modify Linked Data. More specifically, agents routinely perform three phases. During the first phase, agents sense the environment by accessing the Linked Data resources of the other agents. This phase is described in Section 4.1. During the second phase, agents can perform actions and decide what to communicate the the other agents by reasoning over the state of the environment and their own capabilities. Human agents perform this phase by interpreting the state of the environment using intuitive interfaces while machine agents use a logical formalism. This phase is described in Section 4.2. During the third phase, agents update their own repository with the information they intend to communicate to the other agents, such as the outcome of their actions. This phase is described in Section 4.3.

Table 1. The RDF namespaces used in this document. All other namespace prefixes can be considered to be associated with generic URIs such as `http://example.org/`

Prefix	Namespace URI
`prohow`	`http://w3id.org/prohow#`
`rdfs`	`http://www.w3.org/2000/01/rdf-schema#`

4.1 Knowledge Retrieval

While direct communication requires knowledge of the recipients of the messages, indirect communication requires a common environment that all agents can observe [2]. An agent wanting to use direct communication must know how to transmit information to specific recipients. This would be a impractical in our scenario as agents might be humans or machines and might need to agree on different communication protocols. On the other hand, agents using indirect communication mechanisms only need to know how to store and retrieve information from the shared environment. Using indirect communication, agents can be oblivious to the characteristics or even the very existence of the other agents.

In this framework the shared environment that enables indirect communication is a set of Linked Data resources available online, here called *collective knowledge*. Therefore agents, to communicate, only need the ability to read and write Linked Data resources. It is important to notice that while all agents can access and modify the overall collective knowledge, they cannot modify all parts of it, but only the resources that they have direct control of.

Although agents do not need to have explicit knowledge of each other, they need to know where to access the resources that make up the collective knowledge. Implicitly, this involves either knowing or being able to retrieve the addresses of the resources of the other agents. One possible approach is to agree on a URI for the collaboration. This URI should be dereferencable to an RDF resource containing the addresses of the other repositories in the collective knowledge. For example, we can imagine agent a joining coalition `x:coalition`. By dereferencing this URI, an agent might retrieve the following triples:

```
x:coalition prohow:includes a:resource, b:resource, c:resource .
```

RDF triples are here represented in TURTLE[2] format following the namespace prefixes defined in Table 1. The relation `prohow:includes` links the identifier of a coalition with the resources included in the collective knowledge of the coalition. These resources are here called *seed* URIs. If none of the seed URIs points to a resource editable by an agent a, then it can be inferred that agent a does not belong to this coalition, as it would not be possible for this agent to communicate information to the other agents. We will assume here that resource `a:resource` is editable by agent a. Agents do not necessarily know how many other agents are involved in the coalition, if any, as Linked Data resources might not belong to any agent, or multiple resources might belong to the same agent.

Agents wanting to agree on a coalition URI and on the list of seed URIs might have to resort to direct communication or to centralised systems, such

[2] `http://www.w3.org/TR/turtle/`

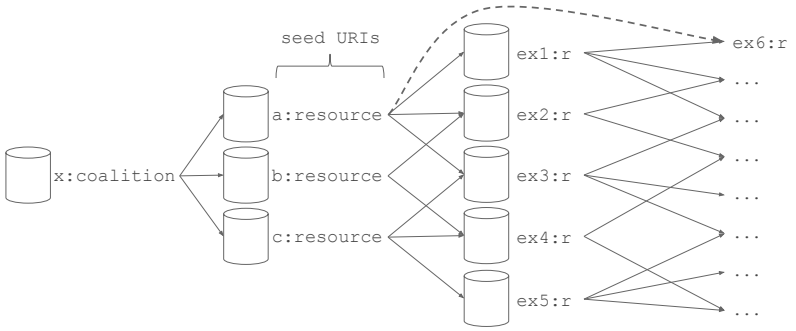

Fig. 2. Dereferencing URIs might lead to the discovery of a large amount of resources. Adding the dashed link reduces the link distance from the seed URIs to `ex6:r` to 1.

as matchmaking systems. This process, however, is considered part of coalition formation and therefore outside of the scope of this work. In this paper we assume that agents already have access to the list of seed URIs.

Having obtained the list of seed URIs, agents can now discover the collective knowledge of the coalition. This is done by dereferencing the URIs of the entities they are interested in, starting from the seed URIs. Agents should attempt to request machine-readable versions of the resources first, using content negotiation. The retrieved resources should then be correctly interpreted if encoded in one of the standard formats, such as Turtle, RDF/XML[3] or RDFa.[4] After retrieving all the data in the collective knowledge agents can proceed to reason about it by querying it. This process of querying distributed Linked Data resources by retrieving them locally is called a *materialisation-based* approach.

Dereferencing URIs allows agents to dynamically discover Linked Data resources at runtime. However, as depicted in Figure 2, the discovered resources might contain more URIs which might lead to the discovery of even more resources, leading to a virtually unconstrained set of resources. Similarly to web crawling, if no limit is imposed on the depth of the exploration, an excessively large number of resources could be discovered and included in the collective knowledge. Considering that agents need to frequently access all the resources in the collective knowledge, unconstrained exploration is impractical.

One possible solution to this problem is to limit the exploration of URIs only to one level from the seed URIs. In other words, only URIs directly retrieved from the seed URIs would be considered as potential resources in the collective knowledge. Agents discovering relevant resources by exploring beyond this limit can include them in the collective knowledge by including their URIs in one of the seed URI resources. In Figure 2, for example, agent a can add `ex6:r` to the collective knowledge by adding a link to it in resource `a:resource`.

An alternative to dynamic Linked Data discovery is the addition of relevant data into a single repository. For example, agent a could copy the RDF data from resource `ex6:r` into its resource `a:resource`. Doing this would give more

[3] `http://www.w3.org/TR/rdf-syntax-grammar/`
[4] `http://www.w3.org/TR/rdfa-syntax/`

flexibility to agent a to decide which triples to include in the collective knowledge. By linking to resource `ex6:r` instead, agent a must commit to include all data from this resource in the collective knowledge. Moreover, copying data in the agent's repository could be considered a more stable approach in case the external resource has a high risk of becoming unavailable.

If the external resource is sufficiently stable and does not contain a significant amount of superfluous information, however, the dynamic Linked Data discovery approach could be considered more advantageous. Linking to external resources and allowing their discovery avoids the problem of data duplication. Not only data duplication increases the amount of repository space required by the agents, but it also makes updates more difficult. In fact, if the data duplication approach is followed, an update to repository `ex6:r` might take a long time to propagate to all the resources that contain its copy, or even not propagate at all. For these reasons, dynamic Linked Data discovery at runtime can be a practical solution if the discovered resources are stable and self-contained.

4.2 Knowledge Representation and Reasoning

In order communicate meaningfully, a shared knowledge representation format needs to be established. Following the Linked Data principles, agents represent their knowledge according to the RDF data model. Shared semantics is achieved by the use of URIs as global identifiers to unambiguously identify entities and concepts, and by the adoption of a shared vocabulary. In the human-machine collaboration scenario the PROHOW[5] vocabulary is chosen.

For the reminder of this paper we will assume that data is represented in RDF and the agents agree on a common vocabulary. However, several approaches could be used in scenarios where these assumptions do not hold. For example, ontology alignment techniques could be used to create semantic interoperability between different conceptualisations. Also, existing systems have been developed to translate non-RDF data into RDF, or mine RDF models from unstructured resources, such as natural language text. One such system has been developed to automatically convert human-generated instructions into Linked Data [5] using the PROHOW vocabulary. The feasibility of this process was demonstrated by the creation of an RDF dataset[6] of over 200.000 interlinked procedures extracted from the wikiHow[7] and Snapguide[8] websites. The ability to integrate a framework with existing resources is particularly important to mitigate the cold start problem. This problem occurs when the low value offered by a system discourages its adoption, while at the same time adoption is needed to increase its value. The value of Linked Data applications, which typically focus on integration and standardisation, often depends on the number of people and organisations adopting them. Applications which can be integrated with existing systems and resources can provide a higher initial value to the early adopters.

[5] `http://w3id.org/prohow#`
[6] `http://w3id.org/knowhow/dataset/`
[7] `http://www.wikihow.com/`
[8] `http://snapguide.com/`

The PROHOW vocabulary has been used to represent both real-world human instructions and machine functionalities [4]. This vocabulary represents tasks in terms of instructions and of their execution. For instance, the procedure in the example introduced in Section 3 can be translated in the following PROHOW representation using an existing parser:[9]

```
:t rdfs:label "Weekly report procedure" .
:t prohow:has_step :t1 , :t2 , :t3 , :t4 .
:t1 rdfs:label "John writes a draft of the report" .
:t2 prohow:requires :t1 .
:t2 rdfs:label "The report is uploaded to an online repository".
:t3 prohow:requires :t2 .
:t3 rdfs:label "Ann corrects the report" .
:t4 prohow:requires :t3 .
:t4 rdfs:label "The report is sent to all the colleagues" .
```

In this example, the main task `:t` is connected to the four steps `:t1` to `:t4` using the `prohow:has_step` relation. The order between the steps is given by the `prohow:requires` relation. Step `:t2` could be linked to a machine functionality `:t5` as follows:

```
:t2 prohow:has_method :t5 .
:t5 rdfs:label "Upload a document to a repository" .
:t5 prohow:requires :t6 , :t7 .
:t6 rdfs:label "The document to upload" .
:t7 rdfs:label "The repository where to upload it to" .
```

The `prohow:has_method` relation indicates that task `:t5` is one possible way to achieve task `:t2`. A machine agent might then be able to complete this task automatically given the required inputs `:t6`, "The document to upload" and `:t7`, "The repository where to upload it to".

One of the main challenges to enable communication between humans and machine agents is the representation of knowledge in a format which is both human and machine understandable. It is therefore important to map such representation both to a logical formalism to allow machine reasoning, and to an intuitive representation, such as natural language, which can be understood by humans. Data modelled with the PROHOW vocabulary can be translated both into a natural language representation and into logical statements [4]. Logical statements allow a machine to infer, for example, that a task is ready to be executed because if all of its requirements are complete, or that it has been accomplished if one of its methods has been completed.

While machines rely on logic, human reasoning and actions are mediated through *interfaces*. In this context, an interface is defined as a software agent capable of translating Linked Data into a human readable representation and of translating human interactions into Linked Data. PROHOW processes can be represented in different ways, including as a list of steps, methods and requirements, a common format in human-written instructions. Interfaces also allow human users to translate their actions and decisions into Linked Data. An existing JavaScript implementation,[9] for example, presents PROHOW tasks in an HTML document and displays buttons to provide additional functionalities. A

[9] `http://w3id.org/prohow/r1/interfaces`

user interested in starting task :t can click on a button next to description of the task to publish the following information on the web:

```
:en prohow:has_goal :t .
```

This triple declares a new attempt (or *environment*) :en to accomplish task :t. This environment :en can be dereferenced to another HTML document where the user can tick off the steps that have been completed, similarly to a to-do check-list. If the user checks the first step :t1, the interface will publish the following triples online:

```
:ex prohow:has_environment :en .
:ex prohow:has_task :t1 .
:ex prohow:has_result prohow:complete .
```

This triples state that, within environment :en, the execution :ex of task :t1 was successful. This information could then be used by a machine agent to infer that step :t2 is ready to be executed, and consequently attempt to accomplish its method :t5.

4.3 Knowledge Update and Coordination

An agent wanting to communicate a set of triples to the other agents does so by publishing them in a repository which belongs to the collective knowledge. It will be the other agents responsibility to retrieve these triples and to consider them in their reasoning process. This approach is straightforward in case the decision to upload certain triples is independent on the rest of the collective knowledge. However, coordination typically requires inferences based on the whole collective knowledge, and changes to the collective knowledge could potentially invalidate such inferences. For example, an agent's decision to complete a task might be overridden by the knowledge that the task has already been completed.

In Linked Data, changes to a knowledge base could be seen in terms of additions and deletions of triples. The problem of unexpected deletions of triples can be avoided by choosing a collaboration model that does not require triple deletion. The chosen collaboration model, PROHOW, is based on a monotonic increase of knowledge and no facts need to be retracted. To avoid the problem of unexpected triples additions, agents should write potentially conflicting statements as "candidate" additions which are confirmed on a first-come first-served basis only after verifying that no other conflicting statement exists.

Candidate sets of triples can be written using RDF reification.[10] Reification provides a unique identifier for each triple, which can be used to attach meta-information about the triple such as the timestamp at which the reified triple was created. Moreover, a reified triple does not entail the triple. This gives agents the flexibility to limit their reasoning only to confirmed facts or to consider also the candidate statements that other agents intend to assert.

Going back to the example presented in Section 3, we can imagine both John and a machine agent a to be capable of completing step :t2, "The report is

[10] http://www.w3.org/TR/2004/REC-rdf-mt-20040210/#Reif

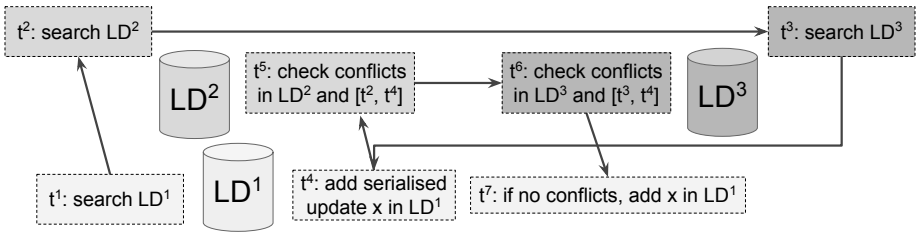

Fig. 3. A simple coordination mechanism for distributed datasets.

uploaded to an online repository". In this scenario, it is desirable that only one of the agents accomplishes task `:t2`. This problem can be solved by coordination, and a simple way of implementing it is on a first-come-first-served basis as depicted in Figure 3. After retrieving the collective knowledge (timestamps t^1 to t^3) and discovering that task `:t2` needs to be completed, John publishes on his individual repository LD^1 the intention to complete it using timestamped reified statements at time t^4. However, agent a might have also published its intention to complete task `:t2` on repository LD^3 at an earlier time t^x ($t^3 < t^x < t^4$). Before considering his intention as final, John's interface will access the other agents repositories to verify whether new conflicting statement have been added (timestamps t^5 and t^6), whether reified or not. When accessing resource LD^3 at time t^6, John's interface will discover a's intention to accomplish task `:t2`. Since a's statement has been created before John's one, the interface will warn John that task `:t2` is already in the process of being automated and he will be asked not to complete it. Agent a, instead, not having found any conflicting statements published before its own, will publish its intention to complete `:t2` in a non-reified format and proceed in accomplishing the task.

5 Complexity Analysis

Agents behaviour described in Section 4 can be seen as a loop where iterations are repeated with a certain frequency. We define frequency f as the number of times the iteration is repeated within a fixed time interval. During each iteration, agents need to retrieve the collective knowledge locally to reason over it. This process of retrieving data locally might need to be repeated a certain number of times k. The coordination algorithm described in Section 4.3 requires retrieving the data twice ($k = 2$) per iteration.

Within a set of agents A, the average amount of data that an agent a needs to retrieve to access the collective knowledge is $d_A(|A| - 1)$, where d_A is the average number of triples in the repositories of the agents in A, and $|A|$ is the number of agents in A. Therefore, within a fixed time interval, the average amount of triples an agent has to retrieve from the web is $fkd_A(|A| - 1)$. An agent collaborating with multiple sets of agents $\bar{A}$ only needs to retrieve the resources of each different agent once. Agents can merge data from each distinct agent it collaborates with ($\dot{A} = \bigcup_{A \in \bar{A}} A$) and use it to collaborate with all the other sets of agents in $\bar{A}$. Using this approach, the average number of triples an

agent would need to retrieve is:

$$fkd_{\dot{A}}(|\dot{A}| - 1) \tag{1}$$

From this formula it can be observed that the required web traffic increases linearly with the frequency at which the agent checks and computes updates f and the number of times the same resources need to be accessed per iteration k. In a concrete implementation of the system, both k and f can be considered constant. The remaining variables determining the complexity are therefore the average number of triples in the knowledge bases of all the other collaborative agents ($d_{\dot{A}}$) and the absolute number of those agents $|\dot{A}|$. The web traffic generated by a single agent can then be considered as having complexity $O(d_{\dot{A}} * |\dot{A}|)$.

The space and computational complexity of the implementation of an agent strongly depends on the number of triples that needs to be stored and processed at each iteration. This number equals to all the triples stored by all the agents that are being interacted with: $d_{\dot{A}}|\dot{A}|$, and it is therefore comparable to the web traffic complexity. The space and time complexity is also affected by other factors which depend on the specific implementation of an agent. These factors are the number and type of SPARQL queries that needs to be executed, and the efficiency of the particular RDF storage and SPARQL engine used.

From this analysis we can observe that the complexity of each agent can be reduced by having agents collaborate with fewer other agents at a time and by having them reason over smaller datasets. It should be noted that, thanks to decentralisation, this complexity does not depend on the total number of agents in the system nor on the total number of triples involved. This complexity can then be kept constant as the overall number of agents and knowledge in the system increases. This analysis suggests that this framework could be applied to a large number of lightweight agents, namely agents that interact with a small number of other agents and that do not publish a large volume of data.

6 Related Work

With respect to human-machine collaboration, the main characteristics that distinguish the proposed framework from existing approaches are two. Firstly, the control given to the users on the whole computation allows them to define how tasks should be accomplished. Secondly, automation does not rely on any central system and it can happen opportunistically. In other words, in different environments, or depending on agent availability, the same task could be automated to a lesser or greater degree. In the field of Human Computation (HC) [7], human and machine efforts are combined to solve tasks that neither humans nor machines alone could solve efficiently. However in typical HC systems, such as Galaxy Zoo,[11] humans play a subordinate role, and their collaboration relies on an existing centralised software system. In Human-Provided Services [9], users can define and advertise the services they want to offer, although they are not

[11] http://www.galaxyzoo.org/

in charge of the overall computation. Related to HC are the fields of Social Machines (SM) and Social Computation (SC) [10]. While HC systems might not have a social dimension, SM and SC focus on how to support social interactions between users. LSCitter [3], for example, is a system supporting collaborative tasks, such as organising a meal or sharing a taxi. While these collaborations are initiated by the users, they must currently follow pre-defined protocols.

The proposed approach also shares similarities with blackboard systems [1], where a central knowledge base is used by multiple agents to collaborate, but it differs for two main reasons. First of all, in the proposed approach the shared environment, or collective knowledge, is fragmented into multiple resources which cannot be accessed simultaneously. A perfectly updated view of the collective knowledge is therefore impossible, as during the time required by an agent to retrieve a single resource, all the other resources might have potentially changed. Secondly, agents can only modify their own resources and no resource can be modified by more than one agent. This means that agents cannot agree on a single resource to be used for coordination. The indirect collaboration mechanism of the proposed framework can also be seen as a form of stigmergy. In the context of multi-agent systems the term stigmergy traditionally refers to complex collaborations emerging from simple agents by indirect interactions mediated through an environment. In the case of rational agents, the concept of *cognitive stigmergy* has been proposed [8].

Given the frequent updates to the collective knowledge of all the agents, we choose to query the distributed Linked Data resources using a materialisation-based approach. Given no assumptions on which resources are likely to be updated, and how frequently, this approach guarantees that any update to the collective knowledge will propagate to all agents after at most one iteration. Under different assumptions, other distributed query strategies could be more efficient. One class of approaches assumes that query processing capabilities, such as SPARQL endpoints, are available remotely [6]. Under this assumption, queries could be split into several sub-queries that will be evaluated remotely against the remote sources. The other class of approaches, to which materialisation-based approaches belong to, does not make this assumption and requires remote resources to be retrieved locally before they can be queried. This process can be optimised by creating local indexes of the remote resources based on their schema [11], based on the URIs they contain, or based on both of those properties [12]. This information is used to decide which resources to access when computing a query, potentially avoiding the retrieval of irrelevant resources.

7 Conclusion

This paper proposed a novel framework for decentralised human-machine collaboration. To avoid the complexity of direct interactions between distributed human and machine agents, Linked Data is used as an indirect communication mechanism. The problem of coordination is therefore translated into a knowledge-sharing problem, where the only requirement for agent participation

is the ability to retrieve and publish Linked Data. Linked Data used for collaboration is frequently updated and therefore agents need to constantly retrieve the most updated version. The necessity to regularly retrieve distributed resources imposes practical constraints on their size which results in Linked Data being divided into small and self-contained resources. This framework can make efficient use of URI dereferencing to discover Linked Data resources at runtime thanks to the small size of those resources and the necessity to retrieve them frequently. To query distributed Linked Data, materialisation-based approaches are chosen over index-based ones due to the high frequency of updates and the need to compute exact answers. An algorithm is proposed to coordinate the potentially concurrent updates of distributed resources by multiple agents. A complexity analysis of this system shows that this framework can scale to a large number of agents as long as the resources shared by the agents are kept small in size, and agents collaborate only with a limited number of other agents at a time.

References

1. D. D. Corkill. Blackboard Systems. *AI Expert*, 6(9):40–47, 1991.
2. D. Keil and D. Goldin. *Indirect Interaction in Environments for Multi-agent Systems*, pages 68–87. 2006.
3. D. Murray-Rust and D. Robertson. LSCitter: Building Social Machines by Augmenting Existing Social Networks with Interaction Models. In *Proc. of the 23rd Int. Conf. on World Wide Web*, WWW '14 Companion, pages 875–880, 2014.
4. P. Pareti, E. Klein, and A. Barker. Linking Data, Services and Human Know-How. In *The Semantic Web. Latest Advances and New Domains*, volume 9678 of *LNCS*, pages 505–520. 2016.
5. P. Pareti, B. Testu, R. Ichise, E. Klein, and A. Barker. Integrating Know-How into the Linked Data Cloud. In *Knowledge Engineering and Knowledge Management*, volume 8876 of *LNCS*, pages 385–396. 2014.
6. B. Quilitz and U. Leser. Querying Distributed RDF Data Sources with SPARQL. In *The Semantic Web: Research and Applications*, volume 5021 of *LNCS*, pages 524–538. 2008.
7. A. J. Quinn and B. B. Bederson. Human Computation: A Survey and Taxonomy of a Growing Field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1403–1412, 2011.
8. A. Ricci, A. Omicini, M. Viroli, L. Gardelli, and E. Oliva. Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts. In *Environments for Multi-Agent Systems III*, volume 4389 of *LNCS*, pages 124–140. 2007.
9. D. Schall. *Service-Oriented Crowdsourcing: Architecture, Protocols and Algorithms*, chapter Human-Provided Services, pages 31–58. 2012.
10. N. R. Shadbolt, D. A. Smith, E. Simperl, M. Van Kleek, Y. Yang, and W. Hall. Towards a Classification Framework for Social Machines. In *Proc. of the 22nd Int. Conf. on World Wide Web*, WWW '13 Companion, pages 905–912, 2013.
11. H. Stuckenschmidt, R. Vdovjak, J. Broekstra, and G. Houben. Towards Distributed Processing of RDF Path Queries. *International Journal of Web Engineering and Technology*, 2(2/3):207–230, 2005.
12. J. Umbrich, K. Hose, M. Karnstedt, A. Harth, and A. Polleres. Comparing data summaries for processing live queries over Linked Data. *World Wide Web*, 14(5):495–544, 2011.