# Integrating Matching Services into the Web of Needs

Heiko Friedrich, Florian Kleedorfer, Soheil Human
Studio Smart Agent Technologies
Research Studios Austria FG
Thurngasse 8/2/16, 1090 Vienna
heiko.friedrich, florian.kleedorfer, soheil.human
@researchstudio.at

Christian Huemer
Institute for Software Technology and Interactive Systems
Vienna University of Technology
Favoritenstrasse 9-11, 1040 Vienna
huemer@big.tuwien.ac.at

## ABSTRACT

Many platforms or marketplaces on the web offer services that describe supply in certain domains. Users have to search actively and often multiple times in different closed platforms to find what they want. This approach produces a lot of overhead for users which can only to some extend be reduced by recommendations and filtering for relevant information. The Web of Needs is an open infrastructure that aims to address this situation. Instead of users searching actively, they express and publish their needs (supply and demand) as linked data documents and get notified when suitable matches have been found. For this approach to work, need documents must be matched with each other by so-called matching services running in the background. The system allows multiple matching services to run in parallel that may be tuned for certain domains of needs. In this paper we describe a prototypical matching service architecture and explain how it can be extended to integrate custom matching algorithms into the Web of Needs.

## CCS Concepts

•**Information systems** → **World Wide Web;** *E-commerce infrastructure;* •**Human-centered computing** → **Collaborative and social computing systems and tools;** •**Computer systems organization** → *Distributed architectures;*

## Keywords

linked data, semantic web, electronic marketplaces, matching, instance matching, matchmaking

## 1. INTRODUCTION

The Web of Needs [7] is an open source system [5] which can be used to create distributed and decentralized marketplace-like applications that are based on common protocols and linked data descriptions for basic entities.

The central notion of the system, in contrast to many current services on the web, is not only to represent supply, but also demand of agents taking part in any kind of marketplace. This explicit representation of supply and demand as well as other forms of intended interaction, like engaging in activities with others or taking action based on shared criticism, is subsumed under the very general concept of *need* objects.

Needs are represented as RDF [8] documents that describe their semantic content in a standardized but flexible way since arbitrary ontologies depending on the domain of application can be used. To create and manage needs, domain-specific applications are required that interact with the users or other agents. When needs are created, they are published at so called *nodes* on the Web where they are stored and are publicly (but anonymously) available to be matched with other needs.

Matching of needs is done in order to connect agents with compatible needs. Therefore, *matching services* crawl multiple nodes to collect the needs, analyze the data, and propose matches between them. This process happens asynchronously with respect to need creation, so agents do not have to search actively. Rather, they are notified if fitting counterpart needs are found. Due to the distributed and open nature of the system, needs can be matched if they do not reside on the same node. Likewise, the architecture supports the integration of multiple mutually independent matching services that may specialize in different content domains or geographical regions.

Different people, companies or other parties can take part in this open setting by providing their own user interface applications (e.g. websites or apps) to describe needs in suitable domains. Other parties could provide node services to host these needs or matching services to connect them.

For instance, a user describes a need to organize a business trip with certain constraints (e.g. price, date, location) to another city. Matching services provide hints to offers from different domains like transportation (e.g. plane, train, car renting or sharing), accommodations (e.g. hotels or private flat offers) and events happening in the evening, for example. This way the user could organize a trip with only one tool, but still have the flexibility to choose from a variety of offers. We provide components to set up a basic implementation of a Web of Needs application [5, 4] including a user interface, nodes and matching services that can be customized and deployed on the Internet.

One of the main benefits of this solution is that all supply and demand, even from different domains, is explicitly represented and accessible in a standardized and open way. Thus, spontaneous and flexible collaboration between all agents in the market could become much easier. Furthermore, matching services could be applied to search for compatible needs automatically taking away a lot of effort from human users who mostly have to search every closed marketplace actively to fulfill their needs.

## 2. MATCHING DEFINITION

As described in an earlier publication, needs are represented as RDF documents that can be linked to each other and exchange messages [6]. General attributes used to describe the content of needs are title, description, tags, date, price or location for instance. However, depending on the domain of interest the content of needs can be described in a flexible way using any kind of ontology.

In this context, matching is defined as the problem of finding meaningful matches or links between each pair of two needs in the system. Such a function is sometimes referred to as matchmaking [10] and it is also linked to the term ontology matching and instance matching [11]. This finding of matches is a dynamic process since needs can be created at any time. The matches are ranked per need based on their score that describes the fit between the two matched needs on a scale from 0 (bad) to 1 (good). Users can give feedback about the quality of matches so that the system can improve matching.

We have defined four need types so far: SUPPLY, DEMAND, DOTOGETHER, CRITIQUE.[1] Need types indicate which needs are suitable for a match. Needs of type SUPPLY represent an offer of goods or services and should only be matched with needs of type DEMAND which represent a need for goods or services. Needs of type DOTOGETHER represent the wish for shared activities with other people while CRITIQUE represents the need to change something (in the environment, in politics, etc). Both should only be matched with needs of the same type.

Whether a match is meaningful depends on the description of needs in a specific domain. Therefore domain-specific matching services are needed for different application domains. But that does not mean that needs should only be matched within one domain. As described earlier for the business trip planning example, a complex need can be matched with other needs from many different domains. Since the Web of Needs is designed as an open system, these domain-specific matching services can be easily integrated as described in the following sections.

## 3. INTERFACES

Matching services gather information about needs and publish hints about possible matches between these needs. A custom matching service can be created by implementing three kinds of interfaces to communicate with nodes:

- Subscribing to need life cycle events
- Crawling needs
- Publishing hints

A matching service can subscribe to different messaging topics in order to be informed about need life cycle events and newly created needs. A node describes the topics it offers (e.g. need creation, need activation, need deactivation) for need information updates as an RDF document that can be downloaded and used by matching services. After a matching service is subscribed to these topics it will be informed about all relevant need changes (including need creation) on a node that happen from now on.

A matching service must be able to learn about needs that were created before it subscribed to a node's messaging topics to do a comprehensive matching of all active needs of a node. Therefore a node can always be crawled for all the needs it manages. This crawling is done using HTTP(S) and downloading linked data RDF need documents iteratively, starting at an index RDF document that lists all needs managed by this node. Using the publish-subscribe mechanism in combination with crawling, a matching service is able to gather all past need information of a node and stay up to date about newly created needs and need life cycle events.

The main functionality of a matching service consists of calculating matches between needs and report this information to the nodes in the form of hint messages. Hint messages have a score attribute that describes the quality or certainty of the match as a number between 0 and 1. When nodes receive hint messages from matching services, they can decide to add this information as RDF documents linked to the corresponding need documents. This hint information can be presented to the user who decides to open a connection to the matched need if he or she is interested in further communication and transaction.

[1]Note that more need types can be defined as needed. For such types to work, an interpretation of their meaning must be shared between client applications and matching services, however.

## 4. IMPLEMENTATION

Any matching service for the Web of Needs can implement the three said interfaces to communicate with nodes. In order to support development of such services, we provide a prototypical implementation [5] of a matching service in Java that can be adapted to create custom implementations.

A matching service in the Web of Needs will typically be used in an open and distributed infrastructure and should be able to process a lot of needs from different independent nodes. Therefore the system architecture needs to provide failure tolerance as well as high scalability by distribution and parallelization of tasks. Another architectural requirement is extensibility to allow for multiple different matching algorithms to be applied in parallel by the system.

To meet these requirements, the matching service prototype was constructed out of loosely coupled components based on the actor framework *Akka* [1]. According to the actor architecture, components are realized as actors that communicate by message exchange.

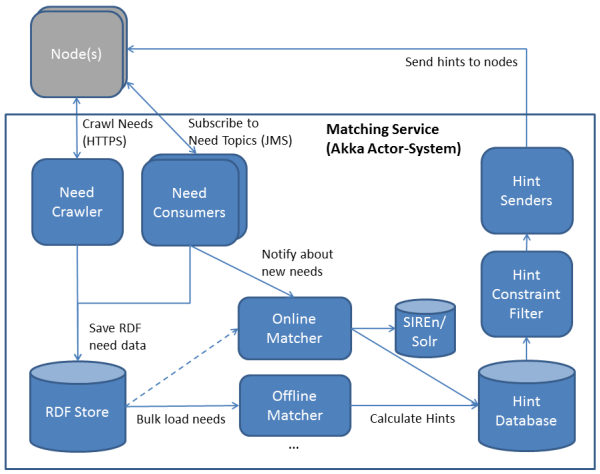


**Figure 1: Matching service implementation with online and offline matching algorithms**

As shown in figure 1 there are actors like *need consumers, hint senders or need crawlers* that implement the interfaces to the nodes. The matching service uses a *local RDF data store* to hold need data, received or crawled from nodes for matching. Currently, two matching algorithms are implemented (*online matcher* and *offline*

*matcher*) that can be plugged into the system on demand to compute matches in parallel and with different strategies. Both of these matchers forward the computed matches as hint messages to an internal *hint database* that keeps track of all the computed hints. Using that database the *hint constraint filter* can filter out duplicate hints or apply other constraints. Hints that pass the filter are actually sent out to the nodes by the *hint sender*. All these actors are connected through an event bus where they can subscribe and publish events inside the matching service.

# 5. INTEGRATION OF MATCHERS

The first matching algorithm of our prototype implementation is called an *online matcher* which means its goal is to compute matches in real time when needs are created. The corresponding actor subscribes to need events on the event bus and computes matches for every new need event received. These matches are immediately published on the event bus to be sent to the corresponding node as hint messages. Information retrieval mechanisms, provided by *Solr* [2] servers, are used for the matching. Every RDF need document is converted to JSON-LD [3] and saved to the Solr index. Matches are computed by automatically generating and executing Solr queries from incoming needs. These queries are specific for the current need and find similar need documents (similar title, description, tags, date, location etc.) of the appropriate need type that are already in the *Solr* index.

The second matching algorithm implementation is called an *offline matcher*, and is not supposed to compute matches in real time, but for a bulk of needs in certain time intervals. The need information is loaded from the local RDF store by the offline matcher. An extension [12] of a tensor factorization algorithm called RESCAL [9] is applied as a link prediction approach to learn from matches with good feedback in the past and predict possible new matches between needs. Learning of good matches can be based on multiple dimensions (e.g. content and date or location constraints). This computation is more resource and time consuming, but can be a valuable complement to the online matching, since matches can be discovered that have been missed by more simple text retrieval methods. Matches can be published to the event bus the same way as for the online matcher.

The two described matcher implementations can be plugged into the matching service dynamically as independent and distributed components. They apply actors that listen to need changes and send hint messages on the event bus of the matching service. Furthermore, they can access the common RDF need store of the matching service. Following this principle, the matching service implementation can be extended by new custom matchers that apply any kind of matching approach or focus on a specific domain of need ontology.

# 6. FUTURE WORK

Even though there is already a working prototype implementation of the Web of Needs, there are still conceptual problems to be solved when running multiple matching services in parallel in an open setting. Here is a short overview of topics for future work.

Multiple matching services can generate matches for the same needs independently of each other and propose them to the corresponding nodes. The score of the matches is between 0 and 1 and can be set by each matching service autonomously. Therefore, comparing scores of different matching services is an open problem. That means currently there are no explicit common ranking criteria for matches from different matching services. So either the nodes or the user application (with the help of the user) would have to find some kind of intelligent ranking and filtering of duplicates to not flood the user with too many irrelevant matches.

In an open system everybody can provide matching services and generate any number of matches including advertisement, spam or (denial of service) attacks. This problem relates to the one above because there is currently no way to compare the quality of matches from different services automatically. Future research has to find acceptable trade-offs between openness and quality of matches as well as system performance.

Being able to assess quality of matches across different matching services automatically, or coupled with user feedback, would also be a central requirement for building a payment service model for matching. Matching service providers that create useful hints could automatically share some profit of successful transactions that their matching services suggested. Such a model may be a powerful incentive to build high quality matching services that not only focus on advertising certain products, but to generate more helpful matches for users.

# 7. ACKNOWLEDGMENTS

This work was supported by the Austrian Research Promotion Agency (FFG) in the COIN project *USS WON – Usability, Scalability and Security on the Web of Needs*.

# 8. REFERENCES

[1] Akka. http://akka.io/. Retrieved June 22, 2016.

[2] Apache solr. http://lucene.apache.org/solr/. Retrieved June 22, 2016.

[3] Json-ld. http://json-ld.org/. Retrieved August 10, 2016.

[4] Web of needs application image repository on docker hub. https://hub.docker.com/r/webofneeds/. Retrieved June 22, 2016.

[5] Web of needs source code repository on github. https://github.com/researchstudio-sat/webofneeds/. Retrieved June 22, 2016.

[6] Florian Kleedorfer, Christina Maria Busch, Gabriel Grill, Soheil Khosravipour, Fabian Salcher, Alan Tus, and Erich Gstrein. Web of needs - a new paradigm for e-commerce. In *Business Informatics (CBI), 2013 IEEE 15th Conference on, IEEE*, volume 1, pages 94–101. IEEE, 2013.

[7] Florian Kleedorfer, Christina Maria Busch, Christian Pichler, and Christian Huemer. The case for the web of needs. In *Business Informatics (CBI), 2014 IEEE 16th Conference on*, volume 1, pages 94–101. IEEE, 2014.

[8] Frank Manola and Eric Miller. Rdf primer. http://www.w3.org/TR/rdf-primer/, 2004.

[9] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *Proceedings of the 21st international conference on World Wide Web*, pages 271–280. ACM, 2012.

[10] Tommaso Di Noia, Eugenio Di Sciascio, Francesco M Donini, and Marina Mongiello. A system for principled matchmaking in an electronic marketplace. *International Journal of Electronic Commerce*, 8(4):9–37, 2004.

[11] Pavel Shvaiko and Jérôme Euzenat. Ontology matching: state of the art and future challenges. *Knowledge and Data Engineering, IEEE Transactions on*, 25(1):158–176, 2013.

[12] Nikita Zhiltsov and Eugene Agichtein. Improving entity search over linked data by modeling latent semantics. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 1253–1256. ACM, 2013.