# Dynamic data sharing for facilitating communication during emergency responses

<*Note*: for first submission leave authors and affiliation blank for double blind refereeing>
<If your paper is accepted, you will add these back in for the second upload>

**First author's name**
Affiliation
e-mail address

**Second author's name**
Affiliation
e-mail address

**Third author's name**
Affiliation
e-mail address

**ABSTRACT**

This paper describes the CHAIn system, which is designed to facilitate data sharing between disparate organisations during emergency response situations. It uses structured data matching to reformulate failed queries in cases where these queries failed because of incompatibilities between the query and the schema of the queried datasource. This reformulation is done by developing matches between the schema according to which the query is written and the schema of the queried datasource. These matches are then used to reformulate the query and retrieve responses relevant to those expected by the original query. Despite the growing interest in intelligent query answering, we believe that integration of data matching into query answering is novel, and allows users to successfully query datasources even if they do not know how the data in that source is organized, which is often necessary during emergency responses. We describe the proof-of-concept system we have developed and the encouraging evaluation we have so far carried out.

**Keywords**

Matching, query, data interpretation, communication

## INTRODUCTION

Fast, effective data sharing is essential in many situations; one of the fields in which this is particularly pressing is in the field of emergency response. Emergency response situations are characterised by the coming together of large numbers of organisations, each of which is likely to have large amounts of data, much or some of which may be pertinent to the current emergency. Some of these organisations may be well known to each other, but others will be unknown and potentially untrusted. Datasources even of known organisations may be highly dynamic. Reports into previous emergency responses frequently cite poor communication and failure to effectively share information as a significant barrier to an effective response (HMSO, 2007). There is much interest in increasing levels of automation in this process as an attempt to address these problems. There are many problems surrounding such automation; the one in which we are particularly interested is that of mismatch between datasources.

Successful querying of a datasource depends on a good understanding of that datasource, thereby ensuring that the schema of the query correctly aligns with the schema of the queried datasource. If data querying is part of an automated process, such knowledge depends on being able to anticipate in advance exactly what datasources will be relevant and knowing accurately what the schema and data representations of that source will be at the time of querying. If such knowledge is possible, then effective communication is best addressed by pre-alignment of data sources, where ideally these datasources use the same fixed vocabulary for easy integration. However, in the general case such an approach is unrealistic. In a highly dynamic environment, such as emergency response, it is usually not valid to assume we will know exactly whom we will need to interact with in advance, or exactly what the context of this interaction will be. Assuming so precludes the possibility of dynamic interaction with new organisations, not anticipated at design-time, and of interacting with known organisations that have updated or altered their data in some way. Whilst pre-alignment is desirable where possible, depending only on this enormously limits the possible interactions during the response.

Since speed is of the essence in emergency situations, relying on human ability to identify these problems and update queries accordingly is not feasible; in addition, this depends on the ability to access the schema of other people's data, which is not always permitted or practical. We therefore believe that automatic reformulation of queries based on matching between the schema of the original query and that of the queried datasource is a necessary part of automating this communication process.

This paper introduces the CHAIn (Combining Heterogeneous Agencies' Information) system, which can be used by an owner of a datasource to formulate appropriate responses to incoming queries, even when these queries fail to match the datasource at the schema level and/or the data level[1]. We can assume that the schema is available without loss of privacy, because CHAIn is a system that is installed by data owners to facilitate the controlled sending out of their data, and details of their schema need not be passed to any other party. Potential responses are ranked, and the process may be fully automated or may be used as a basis for fast, efficient human interaction with large datasources.

The matching component of CHAIn is based on the structure-preserving semantic matching (SPSM) algorithm (Giunchiglia, McNeill, Yatskevich, Pane, Besana and Shvaiko, 2008), originally developed in the OpenKnowledge project (www.openk.org) for dynamic service integration. In contrast to other matchers, this algorithm matches not just nodes in a hierarchy but *structured* terms and is thus applicable to the structure of a query in the way that standard ontology matching is not. The main contribution of this paper is in the adaptation of this successful matching process to the problem of dynamic query matching in (potentially) large data. Our hypothesis is that we can use dynamic matching to provide meaningful results to queries that would otherwise fail.

The paper is organised as follows. The next section uses a worked example to describe the aims of the system. The following section describes the process of the system in more detail, and the next section then discusses the results we have obtained by using this system on various emergency response datasources. The penultimate section puts our work in the context of other related work and the final section concludes the paper and discusses some of the key issues we need to address in developing CHAIn from a proof-of-concept system to a system that is usable in the field.

## WORED EXAMPLE

Our techniques are designed primarily with an emergency response scenario in mind. Such situations are data rich, and are characterised by the need to share data quickly and effectively with collaborators who may be previously unknown, and may not be trusted. Formulating correct queries under these circumstances is extremely difficult, and there is, therefore, a pressing need for automated query reformulation.

Consider a flooding scenario. Imagine an environmental organisation which is trying to determine how full the rivers are to anticipate the course of the floods. They will have their own sensors monitoring this, but may want to get additional information from other organisations which also have sensors - for example, other environmental agencies from upstream regions and private companies who monitor water levels for their own commercial purposes but may be agreeable to sharing data during an emergency. Perhaps in their own datasources they frame readings from their sensors as follows:

*measurement(Reporter_ID,Node,Level,Date)*

[1] A mismatch at the *schema level* is one where the structure of the data differs: for example, the columns of a database are in a different order. A mismatch at the *data level* is one where the schemas match at this particular point, but the specific data differs: for example, dates are given as DDMMYY rather than MMDDYY.

They will then send this query to other organisations which they believe may have relevant information. If the organisation were automatically developing queries to populate this table, the query that would be formulated, in SQL-like format, would be:

**SELECT** *reporter_ID, node, level, date*

**FROM** *measurement*

If there is no mismatch at either the data or the schema level, the query will succeed and appropriate responses will be returned without the need to invoke the CHAIn system.

However, the queried organisation may organise their data differently on many levels. For instance

*reading(Node,Date,Water_level)*

There are various mismatches here, in the words used, in the organisation of the arguments and in the number of arguments. The query above will fail, so returning an appropriate response will require alignment with the datasource schema.

## QUERY RESPONSE PROCESS

The goal of the CHAIn system is to return an appropriate response to a query on a datasource despite the fact that the query is not correctly formulated for that datasource. Whenever a query fails, CHAIn will first investigate whether the schema used by the query is correct with reference to the schema of the queried datasource. If it is not, matching is used to determine whether there is anything in the schema of the datasource that approximates to the schema of the query, and, if so, reformulates the query accordingly. If a query fails when there is no mismatch in the schema - either because the original schema of the query was correct according to the schema of the datasource, or because the query has been reformulated to reflect the schema of the datasource - the CHAIn system will then consider potential mismatches at the data level. Ultimately, the query will be responded to with a set of responses using the data that CHAIn considers the most appropriate response to the original query, or it will fail if there is nothing in the queries datasource that is considered sufficiently relevant.

In this section, we describe the various aspects of the process, with reference (where appropriate) to the worked example.

### The role of Humans

CHAIn can operate completely automatically because the matching process ranks potential matches. It is, therefore, possible to produce a single best-ranked response to the original query. Setting CHAIn to be fully automated is the best approach where responses are needed very quickly and quality of matching is not important. However, we primarily view CHAIn as a method for allowing human users to interact with large datasources quickly and effectively, providing them with the tools to make informed decisions. It is therefore the general expectation that when the matching process is completed, a set of results passing a given threshold for matching quality, or constrained by the number of required results, is presented to a user that owns, or belongs to an organisation that owns, the queried datasource. These results are presented with clear information about where approximations were made, where parts of the query were left unanswered, and so on. This allows for the input of local knowledge into context-free matching, and helps datasources where localised terminology is used to become more universal comprehensible. For example, *reading* is considered by WordNet to be a direct hypernym of *measurement*, so a match may be found on that basis. However, someone who understood the data may know that *reading* in this context refers to book readings, and is not a correct match for *measurement*. These kinds of poor matches will often be picked up automatically because there will be poor matching between the arguments of the terms, but expert human input can still be valuable and there are many situations in which local knowledge is invaluable.

This user (from the organisation receiving the query) will be able to decide to send back one or more (or none) of the responses suggested by CHAIn to the querier. The current implementation simply presents a ranked list of possible responses to a user, with information about the matching. However, a more usable system would need to focus on HCI considerations to allow the user to navigate possible responses easily and effectively. On the querying side, the querying organisation could automatically accept this response if desired, or it could also be filtered by a human user within that organisation to determine whether to accept the response, or which of a

list of potential responses is most appropriate. This will depend on what the information is needed for; for example, a response missing one particular attribute of the query may be useless, whereas one missing another attribute may be of some value. In the running example, perhaps *reporter_ID* is not essential because the *Node* information can be used to determine where the reading is coming from, so a match where this is missing would be less valuable than one where it is present, but still of some use. However, perhaps the query is primarily to discover the water level at various places, so if the attribute *water_level* were missing, the response would have no value to the querier.

### Lifecycle of the Process

1. A query to a particular datasource is received by the organisation (or individual) that owns the datasource.
2. If the query fails, it is first determined whether this is due to mismatch at the schema level. If not, go to step 6. If so, naive matching is done on the schema of the datasource to narrow down the search space to likely matches.
3. Potential matches are sent pair-wise with the query to the SPSM matcher.
4. The SPSM matcher returns matches together with a score, which can be used to reject poor matches and rank good matches.
5. The query is reformulated according to the matches and resent to the datasource.
6. If the query fails, we look for mismatches at the data level, since we know that at this stage the query is correct with respect to the schema.
7. Potential responses to the query passing a given threshold are presented, with appropriate annotation describing the match, to a user who is knowledgable about the datasource. This user then choses any number of matches to be returned to the querier. (This step can be omitted and the responses can be returned according to automatic ranking if required).
8. The querier receives the response(s) to their original query, together with information about the matches that were used to produce the response, and uses these as they feel appropriate.

### Format of the Datasource

The schema-matching part of the process matches first-order terms: that is, terms of the form $predicate(Arg_1,Arg_2, ..., Arg_n)$, where each $Arg_i$ may itself be a function. This process is therefore applicable to data formats that can be translated into this format. Since this is a very general format, this includes a vast number of common representations. Currently, the CHAIn system can be used for SPARQL queries to RDF datasources and for SQL queries to RDB datasources, but it could easily be extended to be applicable to more formats. Extending to a new format requires extensions to the querying and reformulating part of the process, but does not affect matching, the central process, or the interaction with the user.

### Narrowing Down Search Space

The matching part of the process is done by the SPSM algorithm, which does pairwise matching of structured terms. This is an expensive process, and it is not feasible to do it between a single query and every possible (potentially large) datasource owned by the receiving organisation. It is therefore necessary to perform a filtering step, narrowing down the datasources to a subset of likely candidates. This is done through fairly naive keyword matching on the query datasource name to the datasources of the receiving organisation. This allows us to select a set of datasources and their corresponding schemata as candidates for matching the query. There is, of course, a tension in determining how wide this net should be cast: we do not wish to exclude potentially good matches; however, we do not wish to end up with a very large subset of the data, because of the cost of performing structured matching on a large dataset. Initial results (see Evaluation section) indicate our approach is workable, but extensive simulation is necessary to determine more precisely where this sweet spot lies.

After a query failure for the query dataset name, we compute the *Related Terms* set, to which we then try to match the dataset names owned by the receiving organisation. The schema of the each of the matching dataset names and the query schema are the SPSM input. *Related Terms* is computed based on the SUMO and Wordnet ontologies, and the process is explained below. For example, this process would find 38 related terms for the predicate name *measurement*, including *sampling*, *observation* and *reading*. The narrow-down process would then search the target datasource for any predicates with any of these names, and return them. Perhaps, for example, it would find an exact match for *measurement* and a match for the related term *reading*.

### *SUMO*

Compute *Related Terms*: For the following process we consider *(i)* the original query datasource name $Q$, *(ii)* the set $SQ=\{m_1,m_2,...,m_l\}$ which consists of all subwords of the original query datasource name (if $Q$ is multi-word).

- $i \in$ *Related Terms* if:
  - *relates*$(i, Q)$ and/or *relates*$(Q,i)$ and *relates* $\in$ *SUMO*
- $n \in$ *Related Terms* if:
  - *relates*$(n, m_i)$ or *relates*$(m_i, n)$ and *relates* $\in$ *SUMO* and $m_i \in SQ$
  - *relates*$(n, m_k)$ or *relates*$(m_k, n)$ and *relates* $\in$ *SUMO* and $m_k \in SQ$
  - $m_i \neq m_k$

### *Wordnet*

Compute *Related Terms*: For the following process we consider the set $SQ=\{m_1,m_2,...,m_l\}$ which consists of all subwords of the original query datasource name (if it is not multi-word, then $SQ$ consists of a single word - the original datasource name).

- For each $m \in SQ$
  - find the Wordnet ID of that word
  - *Related Terms* is the set of all *Hyponym, Hypernym, Subclass, Superclass, Meronym, Similar* that relate to $m$.
- Repeat the same process for each *Hyponym, Hypernym, Subclass, Superclass, Meronym* and *Similar*.

### *Schema Candidates For Matching*

Decompose the datasources names we own into sub-words, if possible. Then, *(i)* match the datasources names with the SUMO and Wordnet *Related Terms* we previously computed, *(ii)* perform string matching. For example, the *Related Terms* for the predicate name *measurement* would consist of 38 instances, including *sampling, observation* and *reading*. We would then target datasources for any predicates with any of these names, and return them. Perhaps, for example, it would find an exact match for *measurement* and a match for the *Related Term reading*.

### Structure-preserving Semantic Matching

To match the query to any of the datasources of the receiving organisation, we need to establish whether there exists a correspondence between their structures, *i.e.*, schemata. Thus, we need full graph-like structure matching, rather than merely matching the nodes of graph-like structure, which is the task commonly performed by matchers. The Structure-Preserving Semantic Matching algorithm (SPSM) has been extensively described elsewhere (Giunchiglia et al, 2008), so we do not cover it in detail here. Rather, we provide the reader with sufficient information to understand the principals of the process.

The SPSM algorithm requires two trees as input and returns a score in [0 1] indicating their similarity, together with mappings from each element of the source tree to elements in the target tree, or to null. These element mappings must be $\in \{=,\sqsubset,\sqsupset\}$. SPSM is a two-step process. It first makes use of adapted conventional ontology matching techniques to investigate relationships between the individual words in the nodes of the trees. The second step matches the structure of the trees to discover an overall relationship. This is crucial because the structure of the tree itself contains a lot of semantic information that must be considered if we are determining whether two terms are equivalent or similar. SPSM, therefore, needs to preserve a set of structural properties (e.g., vertical ordering of nodes) to establish whether two trees are globally similar and, if so, how similar they are and in what way.

### Reformulating the Query

The SPSM process returns a list of ranked responses. We see from the narrow-down process that there are two predicates in the target datasource which are considered relevant: *measurement* and *reading*. These predicates, together with their arguments, are then sent to SPSM. In this case, the results are:

1. reading(Node,Date,Water_level)
2. measurement(Area,Wind_speed,Direction,Date)

Here, we see that the exact predicate match is returned lower than the related predicate match, because its arguments are much less relevant to the arguments of the original query. For each response which exceeds a given threshold, the mappings in the match are then used to reformulate the query.

1. Original Query Schema: *measurement(Reporter_ID, Node, Level, Date)*
   Mappings: *measurement ⊂ reading*
   *measurement/Node= reading/Node*
   *measurement/Level ⊃ reading/Water_level*
   *measurement/Date = reading/Date*

Suggested Query Schema: *reading(Node,Water_level, Date)*

2. Original Query Schema: *measurement(Reporter_ID, Node, Level, Date)*
   Mappings: *measurement = measurement*
   *measurement/Date = measurement/Date*

Suggested Query Schema: *measurement(Date)*

For aspects of the query for which mappings are found, the original term is replaced by the matched term; aspects of the query for which no mappings are found are removed from the query. If the latter occurs, the response will not constitute a complete response to the query but may contain sufficient detail to be useful.

The repair process takes as input the mismatched query (both data and schema) and the SPSM matching results and produces new queries which are considered schematically correct. We consider the following cases:

- The query dataset name can match to more than one dataset name.
- Each dataset match comes with the corresponding argument matches.
- Within the same dataset, an argument can have more than one match (more than one argument that can be mapped to).
- A dataset match might correspond to more than one new query.
- A dataset match that does not have any argument matching is discarded.

#### Forming the final query - Data Level Repair

The inputs of this process are all the query schemata that have been formed by the previous process, together with their corresponding data. For each of these we form the corresponding query and we send it to the appropriate dataset. If no answers are returned, given that the schema is correct, we know that there is a mismatch at the data level. The goal of this process is to identify the data values that cause the query failure.

The steps we perform are the following:

1. Select one data value at a time and uninstantiate it.
2. Send query to the dataset again.
   a. Any answers that are returned constitute a potential response to the query.
   b. If no answers are returned, go to 1.

The system currently does no matching at the data level, instead returning all responses which match some of the query data, or, if necessary, only the schema of the query. We consider matching at the data level important and it is in our future plans.

### Returning Query Responses

Up to this point, the process has been fully automated and, as mentioned earlier, it is possible to run the CHAIn system without any user interaction if desired. However, we feel that some level of user interaction will be important in improving the quality of results. We also see the CHAIn system as a useful tool in allowing efficient human interaction with large datasources, guiding the human as to how to appropriately respond to queries to the datasource by reducing a large datasource to a handful of potential responses, together with information to help them understand in what way the responses answer the query, and which aspects of the

query are unanswered.

Helping users, who may have a good understanding of the data used by their organisation but are unlikely to be experts in representation and matching, to appropriately and efficiently evaluate a set of potential responses to a query, weighing them against each other with reference to the particular ways in which each individual response fails to be an exact response to the original query, is a complex and demanding task, requiring high-quality techniques in human-computer interaction. Our current approach is to present the relevant information to the user in a simple manner. Building a more sophisticated, user-oriented solution to this is an important aspect of future work.

## EVALUATION

Large-scale evaluation of this kind of matching is difficult. This is chiefly because:

- we need to develop specific queries in each case, ideally generated from data and schema from existing datasources;
- these need to be sent to a datasource from which we could realistically expect an answer - that is, the query and the receiving datasource need to be similar enough to expect results;
- we need to analyse the results, with the help of experts where appropriate, to judge whether or not the responses are appropriate;
- we need to examine the receiving datasource to see if there were any appropriate responses which were not returned.

A robust evaluation of this approach requires these queries to be generated during the kind of automated process we envisage this work be used in, and the effectiveness of these matches determined through large-scale simulation, to determine what the actual impact of using these matches are. Currently, there is not a great deal of automated data exchange during emergency events, though there is an increasing body of work considering how this could be facilitiated (e.g., Fitzgerald, Bryans and Payne, 2012). Integrating our work with such a system is an important past of future work, but has not been attempted yet. We are therefore considering individual queries and responses, rather than queries and responses generated organically during automated interaction. We have therefore carried out the evaluation as follows: *(i)* We have sourced appropriate data from a collaborator, SEPA (Scottish Environmental Protection Agency). *(ii)* We have sourced online different datasources which discuss similar things, such that we might expect a query from such a datasource to find an appropriate response in our SEPA data.

It is not immediately apparent from the behavior of the system what constitutes success or failure. The system returning results constitutes failure if these responses are not appropriate; similarly, the failure of the system to return any results constitutes success if there was no relevant data in the target datasource. We therefore need to examine results and datasources carefully to judge the performance of the system.

For each evaluation test we select a dataset and we formed a query based on the schema of that dataset[2]. Then we send the query to the other datasets we own. We have so far carried out 26 evaluation tests. Of these, 14 returned responses to the queries. In all 14 cases, the responses were judged appropriate: answers were relevant to the query. 12 did not return a result. In 4 of these cases, this was because the narrow-down process returned no results, and in all cases this was judged an appropriate response because there was nothing in the target datasources which matched the query predicate name (not returning irrelevant results is almost as important as returning relevant results). 8 of these failed at the SPSM matching after receiving input from the narrow-down process. Of these, 3 were appropriate failures, because the arguments of the narrowed-down predicates were not similar enough to the query to justify return data; 5 failed because of implementational issues which we believe can be fixed without difficulty.

There is some difficulty establishing exactly what an *appropriate response* is. Our approach has been to consider the kind of data that would have been generated by the query on the original datasource (owned by the querier) compared to the kind of data returned by the target datasource to determine whether they are reasonably similar. We have also presented many of our results to SEPA to check that they agree with our judgements in these cases.

In this section, we present two tests, to give a flavour for the results we have achieved. In all both cases the data

[2] The selected dataset always has relevant information with those to query.

is mapped to SEPA datasets[3].

**Test 1** We formed a query based on the EuroStat[4] Fresh Water Resources dataset schema:

water( resource, measure, geo, timePeriod).

The narrow-down process returned four candidates at the dataset/predicate level:

1. waterBodyPressures
2. waterBodyMeasures
3. surfaceWaterBodies
4. bathingWaterLocations

Once SPSM had performed structured matching on these four, only one (waterBodyPressures) passed the threshold, returning the following matches:

1. water ↦ waterBodyPressures
   water(timePeriod) ↦ waterBodyPressures(identifiedDate)
   water(geo) ↦ waterBodyPressures(waterBodyId)
   water(measure) ↦ waterBodyPressures(assessmentCategory)
   water(resource) ↦ waterBodyPressures(source)

After applying the matches, the reformulated query is: *waterBodyPressures(identifiedDate, waterBodyId, assessmentCategory, source)*, which we sent to the datasources and retrieved the corresponding answers. The timings from the first call of our system, until the finish are: 2m10.208s, from which 0m49.725s is the amount of CPU time spent in user-mode code and 0m 8.023s is the amount of CPU time spent in the kernel within the process.

**Test 2** We formed a query based on the UK Environment Agency[5] BathingWaters dataset schema:

uk_BathingWaters( sampleClassification, prefLabel, long, lat, northing, easting, latestComplianceAssessment, type, country, district, envelope, latestBathingWaterProfile, sedimentTypesPresent, uriSet, regionalOrganisation, yearDesignated, latestSampleAssessment, eubWidNotation, waterQualityImpactedByHeavyRain, zoneOfInfluence, samplingPoint, complianceClassification, primaryTopic, extendedMetadataVersion, definition, label)

The narrow-down process returned four candidates at the dataset/predicate level:

1. bathingWaters
2. bathingWaterLocations
3. waterBodyMeasures
4. waterBodyPressures

SPSM retuned the following matches:

3. uk_BathingWaters ↦ waterBodyMeasures
   uk_BathingWaters(label) ↦ waterBodyMeasures(waterBodyId)
   uk_BathingWaters(primaryTopic) ↦ waterBodyMeasures(secondaryMeasure).
4. uk_BathingWaters ↦ waterBodyPressures
   uk_BathingWaters(label) ↦ waterBodyPressures(waterBodyId)
   uk_BathingWaters(type) ↦ waterBodyPressures(pressureType)
   uk_BathingWaters(prefLabel) ↦ waterBodyPressures(activity)
   uk_BathingWaters(sampleClassification) ↦ waterBodyPressures(affectsGroundwater)
1. uk_BathingWaters ↦ bathingWaters.

[3] SEPA datasources: bathingWaterLocations, waterBodyPressures, waterBodyMeasures, surfaceWaterBodies, bathingWaters, classifications

[4] epp.eurostat.ec.europa.eu

[5] www.environment-agency.gov.uk/

uk_BathingWaters(label) ↦ bathingWaters(description)

After applying the above matches we formed the following queries *(i) waterBodyMeasures(waterBodyId, secondaryMeasure) (ii) waterBodyPressures( waterBodyId, pressureType, activity (iii) bathingWaters(description)* which we sent to the datasets and retrieve the corresponding answers. The timings from the first call of our system, until the finish are: 3m55.407s, from which 1m25.673s is the amount of CPU time spent in user-mode code and 0m18.614s is the amount of CPU time spent in the kernel within the process.

The results of these two tests, as well as the results of the other 24 tests we carried out, have been discussed with the data owners at SEPA and we believe they constitute good-quality results: that is, when plausible matches exist in the datasource, these are returned by the system; when no plausible matches exist, the system does not return results (with the exception of the five failed tests mentioned above).

## RELATED WORK

The work in this paper is concerned with the extension and adaptation of existing matching techniques to a new domain – that of dynamic query reformulation, specifically in an emergency response scenario. SPSM differs from other matchers in that is applicable to *structured* terms, being built on standard tree-edit-distance techniques (e.g., in (Shasha and Zhang, 1997) and using semantic matching to determine what impact on the overall meaning of a tree altering or removing will have. Previously, it has been used to perform service integration and peer selection in a peer-to-peer network (Osman, Sierra, McNeill, Pane and Debenham, in press). The novelty of our work is in developing a system that can use this structured matching technique to address the problem of returning useful results to a query on a datasource which has not been correctly formulated according to the schema and data within that datasource.

There is a considerable body of work on the problem of intelligent query answering, which aims to do more than simple look-up based on the query. However, this work tends to focus on different problems to the one we are interested in. The issue generally addressed concerns how to return maximum, or maximally useful information and may use information within the datasource - for example, a concept hierarchy - to return information relevant to terms not directly included in the query. However, it is not generally concerned with the problem of querying a datasource with which one is not familiar, so that the terms and structure used may be wholly or partially inappropriate for the datasource: the need to use matching techniques is not considered. (Han, Huang, Cercone and Fu, 1995) describes various approaches to this issue, where intelligent query answering is used to provide more (relevant) detail than the query originally asked for, to provide summaries of data, and to reformulate queries into sub-queries. However, the issue of data matching is not considered; rather, the question is about how much relevant information can be returned, assuming the query is correctly formed. (Han et al, 1995) also raises interesting questions about how to appropriately interact with the user and how to ensure that sensitive data is released in an appropriate way to appropriate people, which are also very relevant to an in-the-field version of our work. (Chu and Chen, 1992) proposes the `DB-Pattern-KB` framework and the type abstraction hierarchy to improve query answering.

Work has also been done on combining data from different sources to provide intelligent responses to queries (see, for example (Hagedorn and Sattler, 2013)). (Buscaldi, Rosso and Arnal, 2005; Hollink, Guus and Wielinga, 2007) investigate how semantic relations based on Wordnet can improve query expansion. However, the problem of data mismatch between the query and the datasource is not considered. In (Calvanese, de Giacomo, Lembo, Lenzerini, Poggi, Rodriguez-Muro, Rosati, Ruzzi and Savo, 2011) the Ontology-Based Data Access framework proposes the use of an ontology to create views of the datasources, so, a user can have access to the data without knowing the data organisation. This work follows the main idea of query rewriting whereas views of the original datasource are created and the original query is rewritten to retrieve answers from those views. In our approach, instead of creating views of the datasources we align the query on the fly to the most appropriate datasources to retrieve answers.

There is some work on query answering over distributed, mismatched data sources; however, this makes more assumptions than our work. For example (Lee, Park, Park, Chung and Min, 2010) describe a rewriting system that is in some ways similar to ours, but assume that the user is provided with all available schemata and generates their queries with this in mind. (Haase and Wang, 2007) address the problem of distributed query answering over heterogeneous ontologies using semantic mappings, but assume that all such mappings are predefined and not generated on the fly. There is also a significant body of work on schema mapping (see (Kolaitis, 2005) for a summary), but this generally assumes pair-wise mappings between full schema, and is thus addressing a different problem to us.

## CONCLUSIONS AND FURTHER WORK

The problem of successfully querying unknown datasources is an important one in terms of facilitating data sharing during emergency response events. The general assumption in database querying is that the querier knows the datasource and can use the correct schema and terminology when developing their query, but this is a massively limiting assumption as it restricts querying to situations where: 1) one knows in advance what datasources one will wish to query; 2) one has the time to devote to examining the datasource and correctly formulating any queries one may have; 3) the datasource is open and one can examine its schema. We believe this discounts a large number of situations in which one or more of these assumptions are not valid, particularly during emergency response situations, where collaborations are formed dynamically and quickly, where information needs to be passed rapidly, and where the size, number and potential sensitivity of available datasources ensure that it is impossible to adequately understand all relevant data.

The CHAIn system described in this paper uses an established structural matching algorithm - SPSM - to address the problem of mismatched database queries. We are centrally interested in how techniques developed for ontology matching can be used to address other situations in which data mismatch of some sort is a problem. CHAIn is currently a proof-of-concept system and requires significant further development and extensive evaluation based on simulation before we expect it to be usable in the field. However, the implementation and evaluation we have produced so far have demonstrated that this approach to queries which are mismatched with the datasource they are querying is valid and potentially useful. There are many ways in which we intend to extend this work: *i)* Extending the matching techniques, for example to be more domain-sensitive (e.g., by include domain-specific ontologies in the matching process) and to include matching at the data level. *(ii)* Building in information about provenance and trust so that users of the system have more confidence in query responses. *(iii)* Developing more sophisticated ways to interact with users. *(iv)* Integrating the matching system into a larger system which controls the interactions during emergency responses (e.g., how organisations identify and interact with one another). *(v)* Extending the system to cope with more sophisticated datasources, and integrating techniques from intelligent query answering. *(vi)* Extensive evaluation based on simulation.

## REFERENCES


1. Buscaldi, D., Rosso, P. and Arnal, E.S. (2005) A word-net based query expansion method for geographical information retrieval, In *Working notes for the CLEF workshop*.
2. Calvanese, D., de Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M. and Savo, D. F. (2011), The Mastro system for ontology-based data access. *Semantic Web Journal,* 2(1):43-53.
3. Chu, W.W. and Chen, Q. (1992) Neighbourhood and associative query answering, *Journal of Intelligent Information Systems,*1(3-4):355-382.
4. Fitzgerald, J., Bryans, J. and Payne, R. (2012) A Formal Model-based Approach to Engineering Systems-of-Systems, *Collaborative Networks in the Internet of Services.*
5. Giunchiglia, F., McNeill, F., Yatskevich, M., Pane, J., Besana, P. and Shvaiko, P (2008) Approximate Structure-Preserving Semantic Matching, In *Proceedings of the 7th International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, 1217-1234.
6. Haase, P. and Wang, Y. (2007) A Decentralised Infrastructure for Query Answering over Distributed Ontologies. In *Proceedings of the 2007 ACM symposium on Applied Computing*
7. Hagedorn, S. and Sattler, K.-U. (2013) Discovery querying in linked open data, In *EDBT/ICDT Workshops*, 38-44.
8. Han, J., Huang, Y., Cercone, N. and Fu, Y (1995) Intelligent query answering by knowledge discovery techniques. *IEEE Transactions on Knowledge and Data Engineering,* 8:373 390.
9. Hollink, L., Guus, S. and Wielinga, B. (2007) Patterns of semantic relations to improve image content search, *Web Semantics: Science, Services and Agents on the World Wide Web,* 5.3:195-203.
10. HMSO (2007) The Pitt review: lessons learned from 2007 floods. London.
11. Kolaitis, P. (2005) Schema Mappings, Data Exchange and Metadata Management. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems*, 61-75.
12. Lee, J., Park, J-H,. Park, J-M., Chung, C-W. and Min, J-K. (2010). An intelligent query processing for distributed ontologies. *Journal of Systems and Software.* 83 (1) 85-95.
13. Osman, N., Sierra, C., McNeill, F., Pane, J. and Debenham, J. (in press) Trust and matching algorithms for selecting suitable agents, *ACM Transactions on Intelligent Systems and Technology (TIST)*.
14. Shasha, D. and Zhang, K. (1997) Approximate Tree Pattern Matching. In *Pattern Matching Algorithms, 341-371.*