Discovering Web Services to Specify More Complete System Requirements

Konstantinos Zachos, Neil Maiden, Xiaohong Zhu, and Sara Jones

Centre for HCI Design, City University, London
kzachos@soi.city.ac.uk, n.a.m.maiden@city.ac.uk,
x.zhu@soi.city.ac.uk, s.v.jones@city.ac.uk

Abstract. Service-centric systems pose new challenges and opportunities for requirements processes and techniques. This paper reports new techniques developed by the EU-funded SeCSE Integrated Project that enable service discovery during early requirements processes and exploit discovered services to enhance requirements specifications. The paper describes the algorithm for discovering services from requirements expressed using structured natural language, and demonstrates it using an automotive example. The paper also reports a first evaluation of the utility of the environment that implements this algorithm when improving the specification of requirements with retrieved services.

1 Developing with Web Services

Web and software services are operations that users access via the internet through a well-defined interface independent of where the service is executed [15]. Service-centric systems integrate software services from different providers seamlessly into applications that discover, compose and monitor these services. Developments in service-centric computing have been rapid. Leavitt [3] reports that worldwide spending on web services-based software projects will reach $11 billion by 2008. However, there has been little reported software engineering research to address how to engineer service-centric systems.

One consequence of service-centric systems is that requirements processes might change due to the availability of services. Discovering candidate services can enable analysts to increase the completeness of system requirements based on available service features. However, for this to happen, new tools and techniques are needed to form service queries from incomplete requirements specifications – tools and techniques developed in the EU-funded SeCSE Integrated Project.

SeCSE's mission statement is to create new methods, tools and techniques for requirements analysts, system integrators and service providers that support the cost-effective development and use of dependable services and service-centric applications in the European automotive and telecommunication sectors [12]. The four-year research project covers four main activity areas – service engineering, service discovery, service-centric systems engineering, and service delivery. In this technical research paper we describe new tools and algorithms for discovering services to use to make requirements specifications more complete.

Sections 2 and 3 describe SeCSE's service-centric requirements process and two research challenges that it generates. Sections 4 and 5 describe our response to these challenges – SeCSE's environment and service discovery algorithm. The algorithm extends information retrieval techniques to overcome these two challenges. Section 6 reports a first evaluation of the usefulness of the environment and algorithm when specifying requirements for service-centric systems. The paper ends with a discussion of future work to extend the environment and its algorithms.

2 Discovering Services in SeCSE

In previous SeCSE work we report an iterative and incremental requirements process for service-centric systems [2]. Requirements analysts form queries from a requirements specification to discover services that are related to the requirements in some form. Descriptions of these discovered services are retrieved and explained to stakeholders, then used to refine and complete the requirements specification to enable more accurate service discovery, and so on.

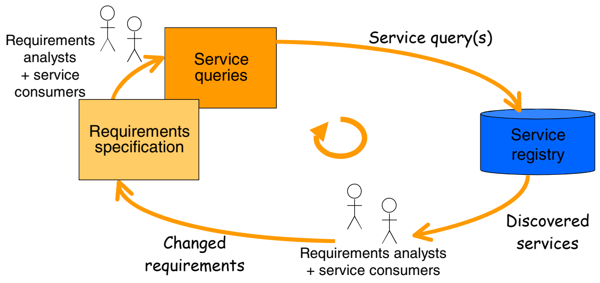

Requirements analysts + service consumers
Service queries
Service query(s)
Requirements specification
Service registry
Discovered services
Changed requirements
Requirements analysts + service consumers

Fig. 1. SeCSE's Requirements Process

Relevance feedback, as it is known, has important advantages for the requirements process. Stakeholders such as service consumers will rarely express complete requirements at the correct levels of abstraction and granularity to match to the descriptions of available services. Relevance feedback enables service consumers and analysts to specify new requirements and re-express current ones to increase the likelihood of discovering compliant services. Furthermore, accurate relevance feedback provides information about whether requirements can be satisfied by available services, to guide the analysts to consider alternative build, buy or lease alternatives or trade-off whether requirements can be met by the available services.

The process has 2 important features. Firstly, to ensure its widespread industrial uptake, the process uses established specification techniques based on structured natural language. For example, to specify system behaviour the process uses UML use case specifications. To specify the required properties in a testable form for generating service monitoring policies it uses the VOLERE requirements shell [6]. As such the process extends the Rational Unified Process (RUP) without enforcing its use or mandating unnecessary specification or service querying activities.

Secondly, the process uses services that are discovered from service registries to support requirements processes in different ways [2]. During early requirements processes it uses services to challenge system boundaries and discover new requirements. For example, if no services are found with an initial query, SeCSE provides advice on how to broaden the query to find services that, though not exactly matching the needs of the future system, might provide a useful basis for further specification. During late requirements processes it uses services to decompose and refine requirements and restructure them to enable more effective service monitoring. Service descriptions provide the requirements team with important quality-of-service information, for example about likely system performance and reliability, used to specify measurable fit criteria for requirements [6].

3 Two Research Challenges for SeCSE

To deliver the SeCSE requirements process we need two new capabilities that overcome common characteristics of natural language requirement specifications – ambiguity and incompleteness. These capabilities are designed to generate queries that will discover services using requirements that are ambiguous and incomplete. Consider the requirement for a car's route planning system: *the system shall provide the driver with directions to a chosen destination by the most direct route*. It is incomplete because it does not state what the directions are and what direction information is needed. It is also ambiguous because it does not define what is the sense of the "most direct" route. There are several possible meanings of *direct*. Does the analyst mean *direct in spatial dimensions; proceeding without deviation or* interruption; straight and short, or does s/he mean having *no intervening persons or agents*?

To handle incompleteness and ambiguity when discovering services we have designed and implemented the two capabilities listed in Table 1. We extend query expansion techniques previously only applied to WSDL service specifications [16] to incomplete statements of requirement to generate more complete service queries. And we apply term disambiguation techniques from information retrieval [9] to ambiguous statements of requirement to generate unambiguous service queries. The claimed innovation is to import research from related disciplines and extend it to handle problems specific to requirements engineering and service discovery.

Table 1. SeCSE's two new querying capabilities

Requirements	SeCSE querying capabilities
Incompleteness	Expansion of service query with terms that have similar meanings
Ambiguity	Disambiguation of query terms using pre-defined term senses

In SeCSE we adopted an engineering paradigm and prototyped a requirements-based service discovery environment, reported in the next section, that implemented these new capabilities as proof of concept and to enable evaluation of their usefulness with our industrial partners.

4 SeCSE's Service Discovery Environment

The environment has three main components: (i) UCaRE, a module to document requirements and generate service queries; (ii) EDDiE, the service discovery engine; (iii) the service registries. We describe these 3 components in turn.

4.1 SeCSE's Service Registries

The environment discovers services from federated SeCSE service registries that store both the service implementation that applications invoke and one or more facets that specify different aspects of each service. Current service registries such as UDDI are inadequate for discovering services using criteria such as cost, quality of service and exception handling. Therefore SeCSE has defined 6 facets of a service – signature, description, operational semantics, exception, quality-of-service, cost/commerce, and testing [8] – that specify features that are important when discovering services. Each facet is described using an XML data structure. The requirements-based service discovery reported in this paper uses the description and quality-of-service facets. The quality-of-service facet is used to refine selection once services are discovered.

Figure 2 presents an example of part of the description facet for a service called *YNavigation*, which finds the location of a reseller of some commodity. Again, to facilitate industrial take-up, SeCSE assumes that service providers describe each facet of a service using structured natural language, due to the excessive effort needed to document services more formally. For example the *ServiceGoal* attribute describes the purpose of the service as an end-state expressed in structured natural language. The *ShortServiceDescription* atribute describes the service's behaviour using a short paragraph similar to a use case précis, whilst the *LongServiceDescription* attribute describes this behaviour in more detail using structured English similar to a use case normal course. Service providers specify and publish services in SeCSE registries using SeCSE's service specification tool reported at length in [8].

Owner: FIAT
Service goal: A reseller for a commodity to purchase is found
Short service description: This service helps you to find the nearest location where you could talk with our reseller. Calculates the arrival time on the basis of the current car position, road preferences and car features.
Service rationale: Car drivers this service to find the commodity.

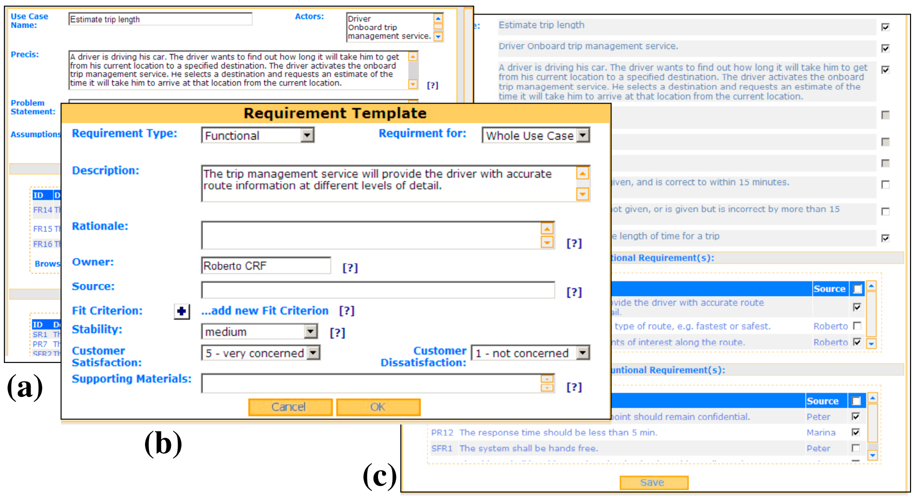
Fig. 2. Example of part of one service with SeCSE's description facet

SeCSE's service registries are implemented using eXist, an Open Source native XML database featuring index-based XQuery processing, automatic indexing. The EDDiE service discovery engine queries these registries using XQuery, a query language designed for processing XML data and data whose structure is similar to XML. Generated queries are transformed into XQueries that are fired at the service description facets of services in the SeCSE service registries.

4.2 The UCaRE Requirements Module

Analysts express requirements for new applications using UCaRE, a web-based .NET application. UCaRE supports tight integration of use case and requirements specifications – a requirement expressed using VOLERE can describe a system-wide requirement, a requirement on the behavior specified in one use case, or a requirement on the behavior expressed in one use case action. UCaRE allows analysts to create service queries from use case and requirements specifications.

At the start of the requirements process, analysts work with future service consumers to develop simple use case précis that describe the required behaviour of a new system. Figure 3(a) shows a use case précis expressed in UCaRE, taken from our industrial automotive partners, to specify what a driver might want from an in-car route planner. The précis is repeated in a readable form in Figure 4. Figure 3(b) shows a simple requirement, also from these partners, associated with the précis expressed using the UCaRE VOLERE shell. An analyst can specify functional and qualities requirements such as the two also shown in Figure 4.



(a)

Use Case Name: Estimate trip length

Actors: Driver Onboard trip management service.

Precis: A driver is driving his car. The driver wants to find out how long it will take him to get from his current location to a specified destination. The driver activates the onboard trip management service. He selects a destination and requests an estimate of the time it will take him to arrive at that location from the current location.

Problem Statement:

Assumptions

(b)

Requirement Template

Requirement Type: Functional

Requirment for: Whole Use Case

Description: The trip management service will provide the driver with accurate route information at different levels of detail.

Rationale:

Owner: Roberto CRF

Source:

Fit Criterion: ...add new Fit Criterion

Stability: medium

Customer Satisfaction: 5 - very concerned

Customer Dissatisfaction: 1 - not concerned

Supporting Materials:

(c)

Estimate trip length	✓
Driver Onboard trip management service.	✓
A driver is driving his car. The driver wants to find out how long it will take him to get from his current location to a specified destination. The driver activates the onboard trip management service. He selects a destination and requests an estimate of the time it will take him to arrive at that location from the current location.	✓
	☐
	☐
	☐
iven, and is correct to within 15 minutes.	☐
ot given, or is given but is incorrect by more than 15	☐
e length of time for a trip	✓

	Source	☐
vide the driver with accurate route ail.		✓
type of route, e.g. fastest or safest.	Roberto	☐
nts of interest along the route.	Roberto	✓

		Source	☐
	oint should remain confidential.	Peter	✓
PR12	The response time should be less than 5 min.	Marina	✓
SFR1	The system shall be hands free.	Peter	☐

Fig. 3. Specification of a use case (a) and requirement (b) in UCaRE, and selection of use case and requirements attributes to generate service queries (c)

Precis:	A driver is driving his car. The driver wants to find out how long it will take him to get from his current location to a specified destination. The driver activates the onboard trip management service. He selects a destination and requests an estimate of the time it will take him to arrive at that location from the current location.
FR1:	The trip management service will provide the driver with route information at different levels of detail.
PR1:	The trip management service will provide the driver with route information within 10 seconds.

Fig. 4. A simple use case précis and requirements for an in-car route planner application, which are used to formulate queries with which to discover services

The analyst then uses the simple tick-box feature shown in Figure 3(c) to select attributes of use cases and requirements to include in a service query. Each service query is formed of one or more elements of a pre-defined type such as a requirement description or rationale, or a use case précis, pre-condition or action. UCaRE maps these element types to service query elements to deliver the seamless integration of service querying with requirements specification that we believe is important for industrial uptake of UCaRE. These integration features are described at length in [18]. The analyst then uses additional UCaRE features described in the next section to refine each generated service query.

An analyst using UCaRE can generate one or more service queries from the specification of a system. Each query is a structured XML file containing structured natural language statements. Because these statements are derived from requirements and use cases, each is potentially ambiguous and incomplete. Each query is then passed to EDDiE, the service discovery engine.

5 SeCSE's Service Discovery Algorithm

The main function of the service discovery algorithm is to discover descriptions of candidate services expressed using the service description facet shown in Figure 1 with service queries composed on the structured natural language statements. Non-functional requirement types fulfil important roles during service selection once discovered, but their use is not described further in this paper.

The algorithm implements SeCSE's two new capabilities:

1. Query expansion – the addition of terms in the service query that have the same or similar meaning to existing query terms, to make the query more complete;
2. Term disambiguation – selecting the meaning, or sense of each term in the query to enable query expansion, thus making the query unambiguous.

The algorithm has the 4 key components shown in Figure 5. In the first the service query is divided into sentences, then tokenized and part-of-speech tagged and modified to include each term's morphological root (e.g. *driving* to *drive*, and *drivers* to *driver*). Secondly, the algorithm applies procedures to disambiguate each term by defining its correct sense and tagging it with that sense (e.g. defining a *driver* to be a *vehicle* rather than a *type of golf club*). Thirdly, the algorithm expands each term with

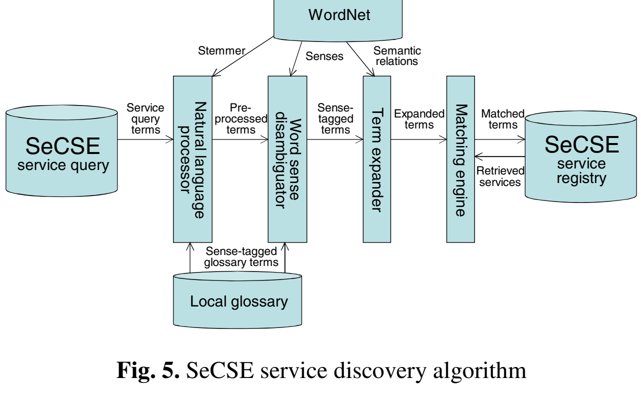


Fig. 5. SeCSE service discovery algorithm

other terms that have similar meaning according to the tagged sense, to make it more complete and increase the likelihood of a match with a service description (e.g. the term *driver* is synonymous with the term *motorist* which is also included in the query). In the fourth component the algorithm matches all expanded and sense-tagged query terms to a similar set of terms that describe each candidate service, expressed using the service description facet, in the SeCSE service registry. Query matching is in 2 steps: (i) XQuery text-searching functions to discover an initial set of services descriptions that satisfy global search constraints; (ii) traditional vector-space model information retrieval, enhanced with WordNet, to further refine and assess the quality of the candidate service set. This two-step approach overcomes XQuery's limited text-based search capabilities.

The WordNet on-line lexicon fulfils an important role for three of the algorithm's components. WordNet is a lexical database inspired by current psycholinguistic theories of human lexical memory [5]. It has two important features. Firstly it divides the lexicon into four categories: nouns, verbs, adjectives and adverbs. Word meanings, called senses, for each category are organized into synonym sets (synsets) that represent concepts, and each synset is followed by its definition or gloss that contains a defining phrase, an optional comment and one or more examples. Secondly WordNet is structured using semantic relations between word meanings that link concepts. Relationships between concepts such as hypernym and hyponym relations are represented as semantic pointers between related concepts [5]. A hypernym is a generic term used to designate a whole class of specific instances. For example, *vehicle* denotes all the things that are separately denoted by the words *train*, *chariot*, *dogsled*, *airplane*, and *automobile*, and is therefore a hypernym of each of those words. On the other hand, a hyponym is a specific term used to designate a member of a class, e.g. *chauffeur*, *taxidriver* and *motorist* are all hyponyms of *driver*. A semantic relation between word meanings, such as a hypernymy, links concepts.

WordNet is an essential component of SeCSE's two new capabilities. WordNet's word senses and definitions provide the data with which to disambiguate terms in service queries. WordNet's semantic relations link terms to other terms with similar meanings with which to make service queries more complete.

EDDiE implements the WordNet.Net library, the .Net Framework library for WordNet [10]. The library provides public classes that can be accessed through public interfaces. For example, to look up a word to see if it is in the dictionary, the following code sample achieves this using one of the public classes:

`if(WNDB.is_defined(word,pos).NonEmpty)` where *word* is a string, *pos* is a part-of-speech. The next sections describe in more detail how the algorithm exploits WordNet to discover service descriptions from service queries.

5.1 Natural Language Pre-processing

This component prepares the structured natural language service query for sense disambiguation and term expansion. In the first step the text is split into sentences and word tokens. For example, when using white space as the delimiter for splitting the sentence *the engine is misfiring*, we get the following tokens: *the*, *engine*, *is*, *misfiring*. In the second step the algorithm identifies complex nominals (e.g. the term *automotive highway*) based on domain-specific terms defined within a glossary (see 5.2.1) and term definitions in WordNet. In the third step the algorithm identifies and removes all terms defined in a list of stop words (e.g. prepositions and pronouns). Next, all terms are tagged with their corresponding part-of-speech (e.g. singular common noun, comparative adjective, etc.) and classified accordingly using an improved version of the Brill Tagger [1]. In the fifth step each term is converted to its morphological root (e.g. *driving* to *drive*). Finally, all duplicate occurrences of a term are removed so that each term is stored only once with its cardinality, as reported in [16].

Returning to our example of the service query generated for requirements shown in Figure 3, the algorithm produces the first version of the XML service query, showing only the noun terms (e.g. *driver, car, location, destination, service*, etc) processed from the use case précis element in Figure 4. The next section describes how EDDiE determines the correct sense of each term.

5.2 Word Sense Disambiguation

Assigning the correct sense to a word in context requires syntactic, semantic and pragmatic knowledge about the word itself, its part of speech, and its context [13]. The pre-processing described in section 5.1 adds syntactic and semantic information to query terms through part-of-speech tagging and WordNet. With this component the algorithm completes disambiguation by iteratively using context knowledge from the project glossary, requirements analyst and other terms in the service query through 7 procedures. Word sense ambiguity is problematic in information retrieval with small queries [14]. However requirements-based service discovery uses larger queries that offer more terms with which to disambiguate. Each procedure is increasingly costly to apply. The first, the cheapest, exploits prior analysis work that the analyst normally undertakes with UCaRE. The next 5 are applied automatically. The seventh, the most expensive, demands analyst input. The 7 procedures are: (i) defining the glossary; (ii) defining single term senses; (iii) defining synonyms; (iv) defining hypernyms; (v) frequency-based senses; (vi) context-based senses, and; (vii) user selection.

For each term T the algorithm determines its sense S using one of 7 procedures that are applied in order. If Procedure *i* does not provide any positive result, then Procedure *i+1* will be applied. In a generic iteration of the algorithm the input is a list of pre-processing terms $T = [t_i,\ldots,t_n]$, and a list of associated senses $S = [St_i,\ldots,St_n]$. T represents all terms to be disambiguated and S represents the semantic meaning of T, where St_i is either the chosen sense for t_i or the empty set, i.e. the term is not yet disambiguated. A set of ambiguous terms $A = \{t_i | St_i = \emptyset\}$ is also maintained. T is initialized with the empty set $T = \{\}$ and A with the list formed by all terms parsed from the Natural Language Processor. The output is the updated list S of senses associated with the input terms T.

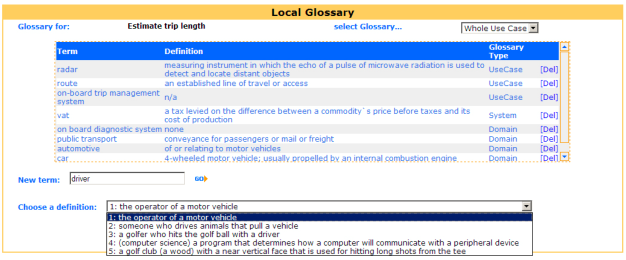
The disambiguation procedures are described below.

Procedure 1: Defining the Glossary. This procedure minimizes ambiguity in the original service query. During SeCSE's requirements process UCaRE maintains a project glossary. Terms in the requirement and use case specification are defined in an interactive glossary that accesses WordNet directly to offer one or more pre-defined senses that the analyst selects and assigns to the term. Figure 6 shows a UCaRE screenshot in which the analyst selects the correct sense #1 for *driver* and tags it in the project glossary, i.e. *a vehicle carrying many passengers; used for public transport.*

Hence, the term *driver* is stored in the glossary with the sense #1. If the term *driver* appears in the list *T*, this procedure finds the term in the glossary and marks it as having sense #1 in *S*.

Procedure 2: Defining Single-Sense Terms. This procedure exploits the existence of terms with only one sense in WordNet, called monosenous terms, and tags them automatically with that sense. For example, the compound noun *motor vehicle* has one sense defined in WordNet and is tagged with that sense #1.

Procedures 3&4: Defining Synonyms and Hypernyms. Procedure 3 finds query terms that are semantically connected to already-disambiguated terms (i.e. terms with a tagged sense) and for which the connection distance is 0 as computed using WordNet hierarchies. A semantic distance of 0 between two terms defines that both belong to the same synset, and therefore the new term is tagged with the same meaning as the connected term. For example consider the terms *passenger* and *rider* in *T*. The noun *passenger* is a monosemous word disambiguated with Procedure 2. One of the senses of the noun *rider*, sense #4 (*a traveler riding in a vehicle (a boat or bus or car or plane or train etc) who is not operating it*), appears in the same synset with *passenger* #1, so the procedure tags *rider* with sense #4.



Local Glossary

Glossary for: Estimate trip length   select Glossary... Whole Use Case

Term	Definition	Glossary Type	
radar	measuring instrument in which the echo of a pulse of microwave radiation is used to detect and locate distant objects	UseCase	[Del]
route	an established line of travel or access	UseCase	[Del]
on-board trip management system	n/a	UseCase	[Del]
vat	a tax levied on the difference between a commodity` s price before taxes and its cost of production	System	[Del]
on board diagnostic system	none	Domain	[Del]
public transport	conveyance for passengers or mail or freight	Domain	[Del]
automotive	of or relating to motor vehicles	Domain	[Del]
car	4-wheeled motor vehicle; usually propelled by an internal combustion engine	Domain	[Del]

New term: driver

Choose a definition: 1: the operator of a motor vehicle

1: the operator of a motor vehicle
2: someone who drives animals that pull a vehicle
3: a golfer who hits the golf ball with a driver
4: (computer science) a program that determines how a computer will communicate with a peripheral device
5: a golf club (a wood) with a near vertical face that is used for hitting long shots from the tee

Fig. 6. Sense definition during requirements specification with UCaRE

Procedure 4, defining hypernyms, works in a similar way. It finds query terms that are semantically connected to already disambiguated terms but for which the connection distance is the maximum 1 as computed using WordNet hierarchies. A semantic distance of 1 between two words indicates that both belong to the same hypernymy/hyponymy relation and therefore the new term is tagged with the same meaning as the connected term.

Procedure 5: Frequency-Based Senses. This procedure assigns the most frequent sense to a term irrespective of its context [17]. This heuristic has been used to baseline supervised word sense disambiguation systems [4]. Its high performance is due to the skewed frequency distribution of word senses. WordNet has a 200,000-word sample of hand-tagged senses through the SemCor project [5]. However, infrequent words can lead to sense bias. Therefore our solution is to constrain the use of this procedure to terms that achieve both a threshold on the frequency of the predominant sense, and a threshold on the ratio between the first sense and the next.

If both are satisfied, the term is tagged with sense #1 from WordNet. Consider the noun *location* appearing in *T*. The term has 3 senses and all senses have appeared in the semantically tagged corpora. The first sense has 992 semantic tags, the second and third senses have both 2 tags. As this scenario satisfies the condition described earlier, sense #1 (*a point or extent in space*) is selected for *location.*

Procedure 6: Context-based Senses. The SemCor bigrams method forms two pairs, one with the previous word, the other with the next word, and searches for these pairs in SemCor corpus [5]. If in all of the occurrences of these pairs, the given word has the same sense, and the number of occurrences is bigger than a preferred threshold, then we assign that sense to the word. For example the term *approval* in *T* has the query context *committee approval of.* The pairs formed are *committee approval* and *approval of.* There are no occurrences of the first pair but four occurrences of the second, more than the set threshold 3, and in all these occurrences the sense of approval is sense #1 (*the formal act of approving*), hence this sense is tagged.

Procedure 7: User Selection. In this most costly procedure the analyst selects the sense for any term that could not be disambiguated using the 6 previous procedures. All pre-defined senses (represented through the gloss) for the term are presented to the analyst to select and assign. Reliance on the other 6 procedures means that user selection should only be needed for a small number of terms. Consider the ambiguous noun *direction* in *T*. The analyst can select the correct sense #1 for *direction* and tags it in the definition window, i.e. *a line leading to a place or point.*

If a term t_i could not be disambiguated through any procedure, then St_i becomes 0, i.e. the term is still ambiguous. A term t_j which is not included in WordNet and defined in the project glossary, St_j becomes -1.

Now let us return to our automotive service query after all 7 procedures have been applied. The partial XML service query in Figure 7 shows the noun terms processed from the use case précis. For example, the term *car* is tagged with sense #1, i.e. *4-wheeled motor vehicle; usually propelled by an internal combustion engine*, whilst the term destination is tagged with sense #3 i.e. *written directions for finding some location; written on letters or packages that are to be delivered to that location.*

```
<SingleTerm>
  <Term termID="1" occur="1" pos="NN" wnsn=”-1”>onboard trip management </Term>
  <Term termID="2" occur="1" pos="NN" wnsn=”1”>car</Term>
  <Term termID="3" occur="3" pos="NN" wnsn=”1”>driver</Term>
  <Term termID="4" occur="1" pos="NN" wnsn=”1”>estimate</Term>
  <Term termID="5" occur="1" pos="NNS" wnsn=”1”>request</Term>
  <Term termID="6" occur="2" pos="NN" wnsn=”3”>destination </Term>
  <Term termID="8" occur="3" pos="NN" wnsn=”-1”>location</Term>
  <Term termID="11" occur="1" pos="NN" wnsn=”2”>time</Term>
</SingleTerm>
```

Fig. 7. An extract of the XML service query after word sense disambiguation, showing the sense number (wnsn) of different nouns in use case précis elements

5.3 Query Expansion

Word mismatches are a fundamental problem to overcome in service discovery. Simply stated, it means that service consumers and providers use different words to express their requirements and service descriptions [11]. The severity of the problem decreases as queries get longer and the likelihood of words co-occurring in the query and service descriptions increases. Through query expansion in EDDiE, the query is expanded using words or phrases with similar meaning to those in the query so that the chance of matching words in relevant service descriptions is increased. Query expansion techniques from information retrieval are essential for effective requirements-based service queries. More formal ontologies for most requirements domains are not available, so synonym-based query expansion using ontological information in WordNet is one of the few viable options.

Elsewhere Wang & Stroulia [16] report a web service discovery technique that combines WordNet with matching on the structure of the WSDL service specification to expand queries with semantically similar words. However, it is limited to formal representations in WSDL. Our innovation is to expand service queries to handle requirements expressed in natural language and compatible with established processes.

EDDiE uses ontological information from WordNet to extract semantically related terms for query terms. As such prior disambiguation is essential to ensure that term expansion uses the correct sense, otherwise queries are expanded incorrectly. Hence only disambiguated terms are considered. EDDiE uses 3 expansion methods:

- *Synset expansion*: terms are replaced by their synsets, for example the term *car* is replaced with its synset for sense #2 [*car, auto, automobile, machine, motorcar*].
- *Hypernym expansion*: terms are augmented by their WordNet direct hypernyms, for example the hypernym of *car* is *motor vehicle*.
- *Gloss words expansion*: terms are augmented with the terms in their glosses, for example the sense #1 definition of the term *garage* is a *4-wheeled motor vehicle; usually propelled by an internal combustion engine.* Hence *motor vehicle* and *engine* are extracted.

Continuing with our example, Figure 8 shows an extract of the XML service query after query expansion. Two terms – *car* and *estimate* – have been expanded with synonyms, hypernyms and gloss terms. For example the query now also contains terms that include *auto*, *automobile*, *motorcar* and *vehicle* as well as *car*, and *approximation*, thus increasing the likelihood of discovering relevant service descriptions. All other terms in Figure 5 are expanded in the same manner, creating a larger query composed of more terms with similar meanings.

```
<SingleTerm>
  <Term termID="2" occur ="1" pos="NN" car</Term>
  <Term termID="4" occur ="1" pos="NN" wnsn="1">estimate</Term>
  ...
  <Term termID="13" pos="NN" refTerm="2" expType="synonym">auto</Term>
  <Term termID="14" pos="NN" refTerm="2" expType="synonym">automobile</Term>
  <Term termID="16" pos="NN" refTerm="2" expType="synonym">motorcar</Term>
  <Term termID="15" pos="NN" refTerm="4" expType="synonym">estimation</Term>
  <Term termID="15" pos="NN" refTerm="4" expType="synonym">approximation</Term>
  ...
  <Term termID="31" pos="NN" refTerm="2" expType="hypernym"> motor vehicle</Term>
   <Term termID="32" pos="NNS" refTerm="4" expType="hypernym">calculation</Term>
   ...
  <Term termID="32" pos="NNS" refTerm="2" expType="gloss">vehicle</Term>
</SingleTerm>
```

Fig. 8. The same extract of XML service query after query expansion, showing synonyms, hypernyms and gloss terms for original terms *car* and *estimate*

5.4 Query Matching

The expanded query is transformed into one or more XQueries that are fired at the service description facets of services in SeCSE service registries. Once an initial set of service descriptions has been retrieved using XQueries, a traditional vector-space [7] model information-retrieval step, enhanced with WordNet, is applied to refine and extract the most similar services from the set. As reported earlier, SeCSE's service description facet in Figure 2 is structured using attributes that facilitate matching with the elements of the service query. For example, expanded terms describing use case actors in the query are matched to terms that describe service consumers, expanded terms from the use case précis are matched to terms in the short service description, and expanded terms in normal course actions of the use case specification are matched to terms describing atomic service operations.

In the traditional vector-space model, documents and queries are represented as T-dimensional vectors, where T is the total number of distinct words in the document collection after pre-processing. Each term in the vector is assigned a weight that reflects the importance of a word in the document. This value is proportional to the frequency a word appears in a document and inversely proportional to number of documents in which this word appears [7]. The WordNet vector-space model involves the maintenance of vectors for each service description property and corresponding expanded query elements, atomic and compound terms, where compound terms consist of multiple atomic terms, for example *nearest location*. We employ the WordNet vector space model to retrieve services that are most similar to the input description on the respective vectors. Corresponding vectors from service description properties and expanded query elements are matched to provide similarity scores. Matching scores of original terms are assigned twice the weight as matching scores of expanded terms (synonyms, hypernyms, gloss terms). A higher overall score indicates a closer similarity between the source and target specifications.

Figure 9 describes part of XML service match between part of the example expanded query and an extract of the XML service description facet for a service that calculates the arrival time on the basis of the current car position shown in Figure 8. It shows that expanded terms – for example *calculation* (a hypernym of *estimate*) and *journey* (a gloss term of *trip*) are needed to match the service because these terms were missing from the original query. Without such expansion of disambiguated terms, retrieval of this service would not be possible. The match values shown in Figure 7 represent the computed semantic distance between the terms *calculation* and *journey*. These match values are used to compute an overall score for the match between the service query and description. Use of these overall scores is demonstrated in the next section.

```
<SingleTerm>
<QueryTerm QId="11" QueryTerm="trip">
   <MatchTerm MId="12" MatchValue="0.543" Expansion Type="gloss">journey</MatchTerm>
   </QueryTerm>
   <QueryTerm QId="16" QueryTerm="estimate">
   <MatchTerm MId="17" MatchValue="0.368" ExpansionType="hypernym">calculation</MatchTerm>
</QueryTerm> ...
</SingleTerm>...
```

Fig. 9. An extract of the XML service match for the service *YNavigation* with the expanded service query

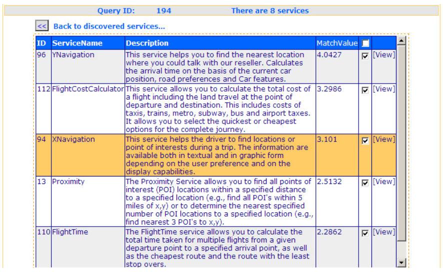
6 Evaluating UCaRE and EDDiE

We evaluated UCaRE and EDDiE with SeCSE's industrial partners. Rather than investigate traditional measures of precision and recall properties of the EDDiE algorithm itself we investigated the utility of algorithm in a requirements workshop. More specifically we explored whether discovered services were sufficient to trigger specification of requirements that had not been specified prior to service discovery. Our assumption underlying this strategy was that high levels of precision and recall were not essential for service discovery – service specifications with lower similarity scores might still enable analysts to discover new requirements.

Four experienced practitioners from SeCSE industrial partners – 2 from Fiat, one from DaimlerChrysler and one from CA, discovered and documented requirements for the in-car route planner system reported throughout the paper. Three of them had extensive experience with such automotive applications. Two of the authors ran the workshop – one facilitated requirements discovery whilst the scribe operated UCaRE.

The workshop was in 2 stages. In the first the facilitator walked the practitioners through the use case précis then normal course to discover requirements for the in-car route planner application that the scribe documented in UCaRE. This process continued until the practitioners were unable to discover more requirements. The scribe then generated a service query from the use case précis and searched a SeCSE-enabled registry containing 112 services for applications that included weather reporting and flight booking as well route planning taken from existing public UDDI registries. The query expanded all term types with possible synonyms, hypernyms and gloss terms.

In the second stage UCaRE displayed each discovered service description as shown on the left-hand side of Figure 10. The practitioners selected which service descriptions to retain as pertinent to the route planner application. The facilitator then walked the practitioners through each service to discover additional route planner requirements from the retained services that the scribe also documented in UCaRE. The facilitator took care to avoid bias and use the same prompts and guidelines before and after service discovery. After the workshop each practitioner completed a questionnaire that ranked each requirement documented during the workshop for its importance and novelty on a simple scale of 1 to 3.


Query ID: 194 There are 8 services

<< Back to discovered services...

ID	ServiceName	Description	MatchValue	☐	
96	YNavigation	This service helps you to find the nearest location where you could talk with our reseller. Calculates the arrival time on the basis of the current car position, road preferences and Car features.	4.0427	✓	[View]
112	FlightCostCalculator	This service allows you to calculate the total cost of a flight including the land travel at the point of departure and destination. This includes costs of taxis, trains, metro, subway, bus and airport taxes. It allows you to select the quickest or cheapest options for the complete journey.	3.2986	✓	[View]
94	XNavigation	This service helps the driver to find locations or point of interests during a trip. The information are available both in textual and in graphic form depending on the user preference and on the display capabilities.	3.101	✓	[View]
13	Proximity	The Proximity Service allows you to find all points of interest (POI) locations within a specified distance to a specified location (e.g., find all POI's within 5 miles of x,y) or to determine the nearest specified number of POI locations to a specified location (e.g., find nearest 3 POI's to x,y).	2.5132	✓	[View]
110	FlightTime	The FlightTime service allows you to calculate the total time taken for multiple flights from a given departure point to a specified arrival point, as well as the cheapest route and the route with the least stop overs.	2.2862	✓	[View]

	Practi - tioner	R	M	S	P
Importance rating	Stage1	2.7	2.5	2.0	2.6
	Stage2	2.3	2.1	1.9	2.2
Novelty rating	Stage1	1.3	2.0	1.6	2.4
	Stage2	1.7	2.2	2.3	2.9

Fig. 10. Discovered service descriptions shown in the SeCSE environment (on left), and average importance and novelty ratings for requirements discovered before and after service retrieval (on right)

The workshop took place as planned. The first stage lasted 60 minutes, during which the practitioners discovered 27 requirements that were documented in UCaRE. The service query then retrieved 11 services, 8 of which the practitioners retained as relevant. The second stage lasted 50 minutes and led to a further 20 requirements from services that were ranked with high and low similarity scores.

At the end of the workshop each practitioner completed the questionnaire (the ordering of the requirements was pseudo-randomized), and their average ratings of the importance and novelty of the 47 requirements are shown on the right-hand side of Figure 8. Results revealed a clear pattern – each practitioner ranked requirements specified prior to service discovery as more important than requirements documented after service discovery, but requirements specified from discovered services were more novel, suggesting that the retrieved services complemented walkthroughs by discovering requirements unlikely to be considered during the requirements process.

To explore how requirements were generated from retrieved services, we undertook a post-workshop analysis to reveal the generation patterns reported in Table 2.

Table 2. Identified patterns of requirements discovery from services

Service-requirement pattern	Number occurrences
Requirements expressed new system features that were a consequence of an application that implemented the retrieved service	7
Requirements were expression of a refinement of the features of the discovered service applied to the system under analysis	4
Requirements expressed required inputs to an application that invoked the discovered service	2
Requirements expressed a function that has the potential to satisfy service qualities described in the service description	1
Requirements were associated with the preceding requirement generated from a service description	3
Requirements and services shared concepts that were deeper than the above input, output and consequence relations	2
Requirements and services had no discernible similarities	1

Several patterns, such as the first, suggest that the practitioners would implement a service-centric application. Other patterns represent good practice that we will validate and reinforce in SeCSE's requirements process.

To conclude the workshop, although limited by the coverage of services in the registry, revealed that EDDiE is capable of discovering services deemed relevant from first-cut requirements and a use case specification specified in a one-hour workshop. Furthermore 8 of the discovered services with different similarity scores enabled the experienced practitioners to specify new requirements that they deemed more novel than the earlier requirements, thus providing evidence that EDDiE and UCaRE can deliver SeCSE's iterative and incremental requirements process.

7 Discussion and Future Work

This paper reports a research-based software environment for constructing service queries from natural language requirements specifications, disambiguating query terms using 7 procedures then expanding them with defined senses, and retrieving discovered services from service registries. An evaluation of the environment revealed that experienced practitioners used retrieved services with a range of similarity scores to generate new requirements that were later ranked as more novel than requirements discovered using traditional use case walkthrough techniques. This positive outcome supports our fundamental claim – that candidate services can enable analysts to increase the completeness of requirements. It suggests the potential to use services that the final application might not invoke to inform later architecture design, service composition and implementation tasks. We encourage researchers to think more innovatively about how to use web services in information systems engineering.

Acknowledgements

SeCSE is funded by the EU 511680 Integrated Project.

References

[1] Brill, E.: A simple rule-based part of speech tagger. In: Proc. Third Conference on Applied Natural Language Processing, ACL, Trento, Italy (1992)
[2] Jones, S.V., Maiden, N.A.M., Zachos, K., Zhu, X.: How Service-Centric Systems Change the Requirements Process. In: Proceedings REFSQ'2005 Workshop, CaiSE'2005, pp.13–14, Porto (2005)
[3] Leavitt, N.: Are Web Services Finally Ready to Deliver? IEEE Computer 37(11), 14–18 (2004)
[4] McCarthy, D., Koeling, R., Weeds, J., Carroll, J.: Using Automatically Acquired Predominant Senses for Word Sense Disambiguation. In: Proceedings of the ACL 2004 Senseval-3 Workshop Barcelona, Spain (2004)
[5] Miller, K.: Introduction to WordNet: an On-line Lexical Database Distributed with WordNet software (1993)
[6] Robertson, S., Robertson, J.: Mastering the Requirements Process. Addison-Wesley-Longman, Redwood City (1999)
[7] Salton, G., Wong, A., Yang, C.S.: A vector-space model for information retrieval. In: Journal of the American Society for Information Science. vol.18, pp. 13–620. ACM Press, New York (1975)
[8] Sawyer, P., Hutchinson, J., Walkerdine, J., Sommerville, I.: Faceted Service Specification. In: Proceedings SOCCER (Service-Oriented Computing: Consequences for Engineering Requirements) Workshop, at RE'05 Conference, Paris (August 2005)
[9] Schutze, H., Pedersen, J.: Information retrieval based on word senses. Proceedings of the Symposium on Document Analysis and Information Retrieval 4, 161–175 (1995)
[10] Simpson, T.: (2005) opensource.ebswift.com/WordNet.Net
[11] Singhal, A., Pereira, F.: Document expansion for speech retrieval. In: Proceedings of ACM SIGIR, pp. 34–41, Berkeley, CA, USA (1999)
[12] SeCSE 2005, secse.eng.it
[13] Stevenson, M., Wilks, Y.: The Interaction of Knowledge Sources in Word Sense Disambiguation. Computational Linguistics 27(3), 321–349 (2001)

[14] Stokoe, C.M, Oakes, M.J, Tait, J.I: Word Sense Disambiguation in Information Retrieval Revisited. In: Proceedings of the 26th ACM SIGIR. pp. 159 – 166 Toronto, Canada (2003)

[15] Tetlow, P., Pan, J., Oberle, D., Wallace, E., Uschold, M., Kendall, E.: Ontology Driven Architectures and Potential Uses of the Semantic Web in Software Engineering, W3C (2005)

[16] Wang, Y., Stroulia, E.: Semantic Structure Matching for Assessing Web-Service Similarity, First International Conference on Service Oriented Computing, Trento, Italy (December 15-18 (2003)

[17] Wilks, Y., Stevenson, M.: The grammar of sense: Is word-sense tagging much more than part-of-speech tagging? Sheffield Department of Computer Science, Research Memoranda (1996)

[18] Zachos, K., Zhu, X., Maiden, N., Jones, S.: Seamlessly Integrating Service Discovery into UML Requirements Processes. In: Proceedings 2006 International Workshop on Service-Oriented Software Engineering (SoSE'2006), Shanghai, China, ACM Press, New York (2006)

[19] Zhu, H., Maiden, N.A.M., Jones, S.V., Zachos, K.: Applying Patterns in Service Discovery.In: Proceedings SOCCER (Service-Oriented Computing: Consequences for Engineering Requirements) Workshop, at RE'05 Conference, Paris (August 2005)