Seamlessly Integrating Service Discovery into UML Requirements Processes

Konstantinos Zachos, Xiaohong Zhu, Neil Maiden & Sara Jones
Centre for HCI Design, City University, Northampton Square, London EC1V 0HB

Abstract: In this paper we argue that processes, techniques and tools for service-centric systems engineering need to be integrated into existing development methods to ensure their uptake and use. This paper reports a new service discovery tool designed to integrate with the Rational Unified Process and UML and offer seamless generation of service queries from requirements specifications. It describes tool features that overcome research challenges for seamless integration. It demonstrates these features from an example of a real-world, service-based automotive application.

Categories and Subject Descriptors: D.2.1 [Requirements/Specifications] Elicitation methods (e.g., rapid prototyping; interviews; JAD); Methodologies (e.g., object-oriented, structured).

General Terms: Design

Keywords: Use Cases, Requirements, Web Services, Service Discovery.

1. DEVELOPING WITH WEB SERVICES
Service-centric systems integrate software services from different providers into applications that discover, compose, invoke and monitor these services. Developments in service-centric computing, for example standards such as SOAP and WSDL, have been rapid. However, there has been little reported software engineering research to address how to engineer service-centric systems – a gap that we are addressing in the EU-funded SeCSE Integrated Project [15]. In this paper we report research to integrate service discovery mechanisms into existing software development methods to enable the effective uptake of these mechanisms.

SeCSE’s mission statement is to create new methods, tools and techniques for requirements analysts, system integrators and service providers to support the cost-effective development and use of dependable services and service-centric applications [15]. The four-year research paradigm covers four main activity areas – service engineering, service discovery, service-centric systems engineering, and service delivery. We report new methods and tools for service discovery.

One consequence of service-centric systems is that requirements processes will change due to the availability of services. Discovering candidate services will enable analysts to explore and validate system requirements based on available service features. However, for this to happen, new tools and techniques are needed to form and execute service requests and queries from incomplete requirements specifications – tools and techniques developed through research that we are undertaking in SeCSE. In this paper we describe a web-based tool for authoring use cases and requirements that analysts use both to specify requirements for future systems and to seamlessly integrate requirements with service requests and queries. Service queries are then used for discovering candidate services that can be applied to improve requirements specifications.

Sections 2 and 3 of this paper describe SeCSE’s service-centric requirements process and research challenges of integrating service discovery into established systems development methods. Sections 4 and 5 describe 2 new SeCSE modules that, we conjecture, integrate service discovery seamlessly into the RUP software development method. Section 6 reports a first usability evaluation of the modules undertaken by one of SeCSE’s industrial partners. The paper ends with implications from evaluation findings and an outline plan of future work.

2. DISCOVERING REQUIREMENTS AND SERVICES IN SECSE
In previous SeCSE work we report an iterative and incremental requirements process for service-centric systems [8]. 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 the stakeholders, then used to revise and refine the requirements specification to enable more accurate service discovery, and so on. Our approach builds on Fischer et al.’s [6] observations about how design queries are incrementally improved by critiquing results from previous queries. Relevance feedback, as this is known, has important advantages for the requirements process. Stakeholders such as service consumers will rarely express requirements at the correct levels of abstraction and granularity to match the descriptions of available services. Relevance feedback enables service consumers and requirements analysts to re-express requirements 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 explore trade-offs to see whether most requirements can be met at acceptable cost by the available services. In this sense SeCSE’s requirements process is similar to commercial off-the-shelf (COTS)-based software development processes such as COMPOSE [9] and SCARLET [11] that use relevance feedback from selected packages to improve the requirements specification. However, SeCSE goes further by...
seeking to exploit the large numbers of candidate services that will be available in service registries, in contrast to the COTS focus on one or two software packages at a time.

SeCSE’s requirements process is depicted in Figure 1. Service queries are extracted from a service request which has been constructed from a requirements specification, then fired at service registries to retrieve service descriptions that are explained to requirements analysts and service consumers to enable them to select the most appropriate service(s). These services are then used to change the requirements using different strategies. For example, if no services are found with an initial query, the tools provide 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.

Figure 1. SeCSE’s iterative and incremental requirements process

SeCSE’s requirements process has 2 important features that distinguish it from other processes. Firstly, to ensure its industrial uptake, the process uses established requirements specification techniques based on structured natural language. For example, to specify the required system behaviour the process uses UML use case specifications [7]. And to specify the required properties in a testable form for generating service monitoring policies it uses a modified version of the VOLERE requirements shell [14].

Secondly, the process uses services that are discovered from service registries to support requirements processes in different ways. It uses services to challenge system boundaries and to discover new requirements. The process also supports requirements decomposition, refinement and restructuring to enable more effective service monitoring with important quality-of-service information, for example about likely system performance and reliability.

3. RESEARCH CHALLENGES

To deliver SeCSE’s service-centric requirements process to our partner service integrators and consumers such as Fiat and Computer Associates, we needed new capabilities to integrate service discovery and selection processes and tools into established systems development methods. Existing service discovery methods [e.g. 5] require software developers to undertake extra tasks on top of existing acquisition, modeling, specification and verification work. Extra tasks will always be needed to exploit the benefits of service-centric computing, however too many tasks can critically reduce the usability and adoption of a method, whilst manual generation of new artifacts such as service requests from requirements can lead to incomplete and inconsistent queries. Therefore, in SeCSE, we are seeking seamless integration of a service discovery method within existing development processes, and in particular the RUP [7] and UML used by SeCSE’s industrial partners. Seamless integration of service discovery from UML system models during architectural specification is reported in [17]. In this paper we describe features of UCaRE, the software tool supporting SeCSE’s requirements process, which we claim provides seamless and therefore usable and useful service discovery capabilities within the RUP.

Furthermore, to discover candidate services more effectively, UCaRE also needs features to overcome ambiguities in natural language specifications. For example the requirement for a car’s on-board engine diagnosis system – the system shall provide the driver with directions to the nearest garage – is ambiguous; for example what is a driver, and what is the nearest garage? As we describe later, unambiguous requirement terms are a pre-requisite for effectively expanding service queries with additional terms with similar meanings that match with terms from service descriptions. Therefore we also describe UCaRE features to support the definition and disambiguation of requirement terms during the requirements process prior to service request generation.

4. HOW UCaRE WORKS

UCaRE is implemented as a web-based .NET application. It is structured using a three-layer model. Its presentation layer defines the graphical user interface. The application layer defines the system logic specified in the ASP.NET pages that generate the dynamic GUI and C# components where classes are grouped by application area. The database layer stores all persistent data about use cases and retrieved services.

UCaRE provides analysts with 2 main capabilities. Firstly it enables them to specify requirements and use cases for a future system, which might or might not be service-oriented. It extends UML use case specifications [1] with requirements expressed using the VOLERE shell [14]. Both use cases and requirements are specified in structured natural language, consistent with how analysts specify requirements in the RUP. Secondly it enables the analyst to generate, from these specifications, one or more service requests that are fired as queries at service registries to discover candidate services compliant with the specified system. To make the requirements process both usable and useful UCaRE provides quick, simple-to-use features to extract natural language elements of the specification to generate service queries.

Figure 2. SeCSE’s service discovery environment

UCaRE passes generated service requests to EDDiE, SeCSE’s service discovery engine [18] as shown in Figure 2. EDDiE manipulates the natural language text to disambiguate terms with their correct senses (e.g. a driver is a person who controls a vehicle or a type of golf club), then expands each sensed term with synonyms (e.g. chauffeur) and hyponyms (e.g.
person) to increase the likelihood of discovering and retrieving candidate services from SeCSE service registries. The Service Explorer then presents these services to analysts. To disambiguate and expand these terms UCaRE and EDDiE both access WordNet, an online lexicon.

4.1. The WordNet Lexicon

WordNet is a lexical database inspired by current psycholinguistic theories of human lexical memory [12]. WordNet divides the lexicon into four categories: nouns, verbs, adjectives and adverbs. Word 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. During the requirements process UCaRE retrieves these senses so that analysts can establish the correct meaning of terms in requirements and use case specifications, as part of a project glossary, prior to generating service requests that are expanded more effectively into service queries. UCaRE implements the WordNet.Net library, the .Net Framework library for WordNet [3]. The library provides public classes which 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:

```csharp
if(WNDB.is_defined(word,pos).NonEmpty)
```

where word is a string, pos is "noun", "verb", "adj" or "adv". The next sections describes how WordNet is implemented in UCaRE and SeCSE’s requirements process.

4.2. UCaRE’s Conceptual Architecture

To deliver the seamless integration that we seek we designed UCaRE on top of SeCSE’s conceptual model, part of which is shown in Figure 3. The left hand side shows associations between use cases and requirements that were shown to be successful during the specification of complex air traffic management systems [10]. A requirement – a desirable property of the system that the analyst shall test compliance for – can be a requirement on all the behaviour specified for the system, all of the behaviour specified in one or more use cases, or the behaviour specified in one or more use case actions. These associations enable requirements to be specified at different levels of granularity. They provide expressions of context for each requirement so that the analyst can specify measurable and testable criteria. They also enable analysts to report requirements in ‘stories’ that enable the analysts to communicate requirements more effectively during reviews.

The right hand side of Figure 3 shows key associations between service requests and service queries. For each service request the analyst generates, EDDiE can generate zero, one or many XML service queries that are fired at SeCSE service registries.

We link use case and requirement specifications to service requests through elements that an analyst imports into a service request from these specifications. An element is an atomic, non-decomposable typed string such as a use case précis, normal course action description or VOLERE requirement description. Each typed element corresponds to an element of the same type in the service request that the analyst imports into the request. As such each service request is composed of a subset of all elements of an originating use case and requirement specification.

![Figure 3. The relationship between a use case specification, requirement, service request and service query](image)

4.3. Documenting Requirements in UCaRE

At the start of SeCSE’s requirements process, analysts work with future service consumers to develop simple use case précis that describe the required behaviour of the service-centric application. Figure 4 describes a typical précis that describes what a service consumer – a driver – might want from an on-board car service, and Figure 5 defines some simple associated requirements. The first, a functional requirement, specifies what the service shall do, and the second, the non-functional requirement, specifies desirable qualities of the service.

![Figure 4. A simple use case précis for an onboard remote maintenance application](image)

FR1: The remote-maintenance service will provide the driver with directions to the nearest garage.
RR1: The remote-maintenance service will provide the driver with reliable directions to the nearest garage.

![Figure 5. Requirements on the on-board remote maintenance application](image)

UCaRE supports the analyst to use the VOLERE shell [14] to specify requirements such as FR1, as shown in Figure 6. VOLERE enables the analyst to specify the requirement’s type, rationale, source, owner and importance scores as well as measurable fit criterion essential for selecting between discovered services on quality-of-service criteria.
Likewise UCaRE implements use case specifications conformant to UML standards [7]. Each use case is specified in 3 parts – the use case management, use case basics and use case normal course parts. Information in the management part about the use case author, date and source is used to manage use cases in the requirements process [1], and not to generate service requests.

At the beginning of the process the analyst documents basic information about each candidate use case such as the actor names, the précis and functional and non-functional requirements on the behaviour specified in the entire use case. The précis and some requirements for our on-board remote maintenance application are shown in Figure 7. Each element is a candidate element of a service request that can be used in service queries to discover candidate services.

![Figure 7. The basics tab of the use case specification](image1)

Later in the process the analyst expands each use case and specifies normal course actions and requirements on one or more of these actions. Example actions shown in Figure 8 include the on-board diagnosis system detects the engine fault and the on-board diagnosis system diagnoses the category of engine problem. Again each action description is a candidate element of a service request.

![Figure 8. Example normal course of a use case in UCaRE](image2)

UCaRE provides several innovative features that support quick and error-free generation of service requests from a requirement specification. These features are described in the next 3 sections.

### 4.4. Developing a WordNet-based Glossary

UCaRE provides capabilities that an analyst can use to develop then apply a glossary of problem domain terms. We see this as important for disambiguating service requests prior to query expansion in EDDiE. The glossary disambiguates requirements terms by tagging them with predefined senses from WordNet during the requirements process. To define a new term, the analyst types the term in a text box and chooses what part of speech (noun or verb) he/she intends to define. UCaRE retrieves all possible definitions for that term from term senses specified in WordNet, then the analyst selects the correct sense, tags the term with it, and enters it into the glossary to be used during subsequent requirements tasks. For example, in Figure 9, the analyst chooses WordNet sense #2 observe or plot the moving path of something for the verb track that is being entered into the glossary. If a term does not exist in the WordNet glossary, because for example it is a service-oriented computing term such as business process, then the analyst can ignore WordNet senses and enter the new correct sense.

![Figure 9. The UCaRE glossary window](image3)
4.5. Auto–Completing Requirement and Use Case Specifications

UCaRE exploits glossary terms with their WordNet-tagged senses to provide analysts with auto-completion capabilities that we conjecture are novel in requirements engineering environments. When the analyst begins to type a word in selected parts of a requirement or use case specification, UCaRE suggests possible glossary terms that match to the incomplete word as a selection list. The analyst can then choose one offered term or complete the word his or herself if the term is not in the glossary.

The auto-completion feature uses various JavaScript objects, including the `regexp` object, which runs through the array containing the keywords and matches them one by one, after which a DHTML object is created dynamically using the C# auto-completion component.

This auto-complete feature offers several potential benefits. Firstly, writing the requirements and use cases is faster, especially if the glossary contains long terms such as on-board diagnosis management system. Secondly the analyst is encouraged to use the glossary in real-time in an easy-to-use way, thus reducing unintentional requirements ambiguity and inconsistency.

Furthermore we are enhancing the auto-completion feature with sentence structures that prescribe how use case actions and requirements descriptions should be written. UCaRE matches these sentence structures to the sentence that the analyst is writing, then refines term selection based on term types such as noun, verb and adjective according to the matched sentence structure. In UCaRE we exploit 8 case grammars derived from theoretical research in linguistics that were applied previously during use case authoring [2]. Three are listed in Figure 10.

**Figure 10: case grammars applied to use case authoring**

So how does UCaRE apply these guidelines? If the analyst enters the incomplete sentence the system informs the driver..., then UCaRE will match the sentence to candidate structures including CG1, CG2 and CG3. Based on the match UCaRE infers that system is a noun and informs is a verb. According to the structures the new term d will be an actor or object rather than a verb, hence the auto-completion feature offers the terms driver rather than drive or driving as shown in Figure 11.

**Figure 11: Auto-Complete feature in UCaRE**

4.6. Generating Service Requests

UCaRE supports a simple two-step process for the analyst to generate service requests and queries from the requirement specification, as shown in Figure 12. In the first step the analyst uses UCaRE’s PRE-REQUEST form to select and include elements of the specification in a service pre-request. To do this, the analyst simply clicks one or more tick boxes of elements to include in the pre-request. In the second step the analyst use the REQUEST form to select parameter values such as the types of term expansion to employ and service registries to search, and if needed add further search information to the service request. Saving this service request automatically generates service queries that EDDiE fires at the selected registries. Each step is simple to do and can involve as little as 4 mouse clicks to fire queries at the registries.

**Figure 12. Two-step process to generate service queries**

Each service request is formed from one or more elements in the requirement specification. An analyst generates a service request using either requirement elements for the whole system, one use case, or one or more actions in one use case. Figure 13 shows how an analyst selects and generates a service pre-request in the pre-request form. The analyst clicks the PRE-REQUEST button then selects the elements he/she wishes to include in his/her service request by simply checking or un-checking tick boxes for each element. For example, Figure 13 shows a pre-request form composed of the ticked use case précis element and two requirements – FR8 and RR1, which will provide the basis of a service request. Figure 14 shows the XML service request generated for our example. The request follows an XML schema that defines its structure, content and semantics.

**Figure 13. Selecting elements for a service request**

**Figure 14. The XML service request formed of elements in the use case specification**
Next the analyst moves to the service request form shown in Figure 15. With this form the analyst can expand terms in the request using different forms of expansion, such as expand the request with synonym or hypernym terms, on different parts of speech such as nouns, verbs and adverbs. The analyst can also add information not in the original specification to the service request, to tailor generated service queries, to support more exploratory service discovery independent of the specified requirements. The analyst can also select one or more service registries to search. SeCSE supports federations of heterogeneous service descriptions and implementations. The analyst can exploit this federation to localize service searches. In Figure 15, analyst selects to search the SeCSE service registry based in Rome.

Figure 15. Selecting what types of expansion to perform on a service request

When the analyst clicks the Save button at the bottom of the screen of Figure 15, (a) UCaRE updates the XML service request from Figure 14, (b) it creates a XML service query using the natural language elements from the service request. Figure 16 shows parts of the resulting XML service query for our example after applying SeCSE’s service discovery algorithm described in the next section.

5. EDDIE - SECSE’S SERVICE DISCOVERY ALGORITHM

EDDIÉ’s algorithm discovers descriptions of candidate services using XQueries generated from queries such as the one shown in Figure 16. This section summarizes the algorithm’s description. A full description is provided in [18].

The algorithm has the 4 key components shown in Figure 17. In the first the service request 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 other terms that have similar meaning according to the tagged sense, to increase the likelihood of a match with a service description (e.g. the term driver is synonymous with the term motorist which is also then 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. In the future we plan that user-defined XQuery functions will compute the semantic similarity measurement.

Figure 16. Part of the final XML service query

Figure 17. SeCSE service discovery algorithm

Services identified using EDDIE’s matching engine against requirements information may lead directly to the identification of new requirements as developers discover services which are ready for use in the domain of interest, but which they were previously unaware of. For instance, services discovered by matching with requirements information from which some constraints had been removed may prompt integrators or consumers to remove the corresponding constraints in their requirements, so that they can make use of functionality provided by existing services.

6. UCaRE EVALUATION

Our SeCSE partner Computer Associates (CA) undertook a usability evaluation of UCaRE in the summer of 2005. An experienced business analyst used UCaRE remotely from offices in Brussels (the UCaRE server is in London) to specify real-world requirements from a recent CA requirement capture workshop in the form of business workflows and processes, a conceptual model, and use case specifications. The evaluation investigated UCaRE’s user-friendliness, fitness for use, maintainability and flexibility. Results were delivered in a 40-page evaluation report [4].
In general the results were encouraging. The analyst was able to use UCaRE to specify workflows and business processes as well as more technical use cases in the use case section of UCaRE. It suggested that CA analysts would be able to use the research prototype, which had undergone 6 months development at the time of the evaluation, in an industrial context. However, the report highlighted important areas that need to be improved to support CA’s software development processes. These included: (i) more flexible requirements management, including capabilities to visualize and group requirements, and detect inconsistencies; (ii) effective requirements versioning and status management; (iii) explicit support for business concept and type modeling; (iv) more effective, role-based control of UCaRE’s service discovery results. The report also revealed new requirements on UCaRE, such as the ability for the analyst to prioritize the use of one term sense over another during disambiguation, and handling business logics and formulae in addition to structured language in service requests and discovery.

We are currently implementing these and other changes to UCaRE to produce the baseline version to rollout to partners that include Computer Associates and Fiat.

7. RELATED AND FUTURE WORK

In requirements research there has been little reported work on service discovery. Schmid et al. [16] report a requirements-led process to enable run-time service discovery, but no tool support is described. Elsewhere Esmaeilsabzali et al. [5] report new models for requirements-based service discovery that assume formal expression of system operations. As such work reported in this paper appears to be the first to integrate requirements and service discovery methods and tools. Indeed, the integration revealed some common concerns in both requirements specification and service discovery. Term disambiguation is important in both. Assigning terms with their agreed meanings during requirements processes can improve query expansion when discovering services. Moreover, text written as part of requirements and use case specifications can form elements of service discovery queries that analysts judge as meaningful. These common concerns suggest that more rather than less integration is possible.

As well as summative evaluations of UCaRE with our industrial partners, future SeCSE work will explore new means of exploiting these common concerns. One idea is agent-driven service discovery. Currently the analyst must invoke service queries by forming service requests and firing them at registries. One idea that we are exploring is to develop intelligent search agents that will automatically form and revise service queries in background mode whilst the analyst specifies requirements, then retrieve and present descriptions of services that these agents infer are useful during different requirements tasks. We aim to develop and report on these agents in the next 12 months.

8. ACKNOWLEDGEMENTS

The authors wish to thank all SeCSE partners for their input and support for the reported work. The work is funded by the EU 511680 SeCSE Integrated Project.

9. REFERENCES


