Delft University of Technology
Software Engineering Research Group
Technical Report Series

Towards Software Component Procurement Automation

Hans-Gerhard Gross, Marco Lormans, Jun Zhou

Report TUD-SERG-2007-002

TUD-SERG-2007-002

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Towards Software Component Procurement Automation

Hans-Gerhard Gross[1] Marco Lormans[2]

Software Technology
Delft University of Technology
Mekelweg 4, NL-2628 CD Delft
The Netherlands

Jun Zhou[3]

The First Research Institute of
the Ministry of Public Security
No 1 South Road of the Capital Gymnasium,
100044 Beijing, China

Abstract

One of the first steps of component procurement is the identification of required component features in large repositories of existing components. On the highest level of abstraction, component requirements as well as component descriptions are usually written in natural language. Therefore, we can reformulate component procurement as a text analysis problem and apply latent semantic analysis for automatically identifying suitable existing components in large repositories, based on the descriptions of required component features. In this article, we motivate our choice of this technique for feature identification, describe how it can be applied to feature tracing problems, and discuss the results that we achieved with the application of this technique in a number of case studies.

Keywords: Software Component, Repository, Feature Mapping, Document Analysis.

1 Introduction

Before a software component can be assembled to form part of a new system, it must be located on a market, its fitness for the purpose has to be determined in terms of functionality and behavior, and it must be selected according to non-functional application requirements. These steps are called component procurement, and they are performed prior to component integration [23,29]. Procurement involves two stakeholders, the component provider, who develops and offers components, and

[1] Email: h.g.gross@tudelft.nl
[2] Email: m.lormans@tudelft.nl
[3] Email: zhoujun@fri.com.cn

the component customer, who requires components in order to assemble a new application. Customer and provider are two roles that may be assumed by individuals of the same working group, of the same organization, or of different organizations (third party).

In the software domain it is common that customers adapt their requirements specifications partially to the components already available, and component providers offer dedicated variants of their existing components. This requires adaptation which it is motivated by the following considerations:

- If component customers devise their applications entirely according to their own requirements, it is unlikely, or at least very difficult for them, to find existing components which will map exactly to their preset specifications.
- When building up systems entirely from existing components according to the predefined specifications of the component vendors, component customers will loose their distinction over their competitors who are using the same domain-specific components. Today, market distinction is primarily achieved through the distinct "look and feel" of the software functionality, and not so much based on the underlying hardware.
- Pure outsourced custom development is typically too costly.

Procurement involves the identification of candidate components for a particular purpose and an assessment of the amount of adaptation to be carried out. During integration, one of the candidate components is selected and integrated into the customer's framework. In other words, the adaptations are implemented. Finally, the integration must be assessed qualitatively, i.e. through testing, or analysis. Certainty about the success of a working assembly can then only be assured after extensive assessment and testing, along the lines described in [22]. An overview of the activities and artifacts associated with component procurement and integration including development steps is displayed in Figure 1. From the figure, it becomes apparent that application engineers have to go through an entire development cycle involving requirements engineering and analysis, design and modeling, and implementation and testing, in order to figure out whether a candidate component can be integrated in the particular context of a given application.

Procurement and integration would be greatly alleviated if both parties identified in Figure 1 would use the same specification styles for required and provided component interfaces, if they would apply the same semantics for their requested and offered component behavior, and if they would communicate on the same level of abstraction. This is typically not the case now, nor likely in the future, so that both stakeholders go back to the least common denominator for specification, natural language. It is the single most important notation for writing down software requirements [7], and common practice in component procurement and integration is that engineers select candidate components based on pure textual descriptions.

This article addresses the challenges of the first activity displayed in Fig. 1, component procurement. Procurement deals with mapping required component features to provided component features and assessing how well existing components meet the stated requirements. We argue that component procurement can be automated, at least partially, on the highest level of abstraction, to support

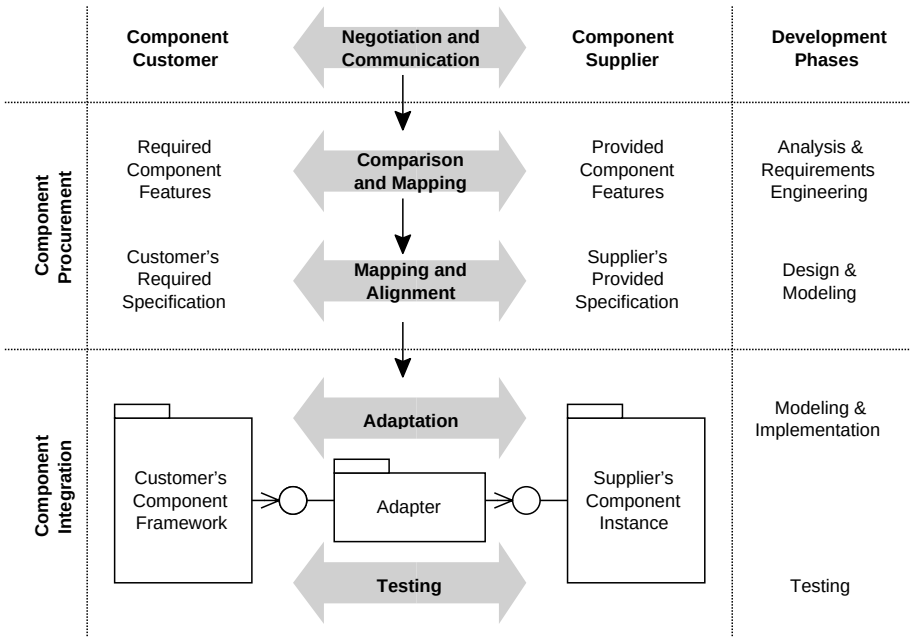


Fig. 1. Activities and artifacts associated with component procurement and integration.

application engineers in the selection of few candidates that are likely to match the stated requirements out of a huge collection of components. Those candidate components may be suitable for adaptation to the given context of the customer's existing component framework.

In section 2, we outline existing techniques such as component development frameworks, component models, and coordination notations, that address the problems of component identification, selection, and adaptation. These techniques are geared mainly toward lower levels of abstraction. In section 3, we reformulate the component procurement problem as a document analysis problem and demonstrate how Latent Semantic Analysis can be applied to identify requirements written in natural language in a component repository automatically. Section 4 outlines and discusses the findings from experiments that we have carried out with the technique, and section 5 summarizes and concludes this article.

2 Related Work and Faced Problems

Despite all the advances in component technology over the last decade, i.e., deployment environments, and run-time platforms such as CORBA, JavaBeans, COM or .NET [43], today, component procurement and integration on higher levels of abstraction, and early during application development, is still not addressed adequately. In order to kick-off the "software industrial revolution" [11], and component-based software development to become a success story, we need effective mechanisms to mediate between various component abstractions, behavioral descriptions, and non-functional properties, on the highest level of abstraction possible.

2.1 Component Mapping and Coordination on the Integration Level

Most of the progress in component-based development has been achieved on the lowest level of abstraction, the implementation level, or component wiring level with all the apparent component platforms and technologies, or component models described in [43]. These technologies support the (automated) interconnection of component services based on so-called interface description languages (IDL).

The component-based development method and formal language research communities have come up with solutions for component feature mapping on abstraction levels above implementation. Various approaches have been proposed over the years to alleviate the typical component adaptation and wiring problems. Some of the methods proposed are of a more formal nature, such as CL [25], Koala [44], Piccola [34], or Abstract Behavior Types [2]. Some others view component integration from a more global perspective and embed the concepts of composition in an overall, less formal development framework. The most commonly known of these so-called development methods, most of which are based on object technology, are OMT [41], Fusion [10], ROOM [42], HOOD [40], OORAM [39], Catalysis [17], Select Perspective [1], FODA [28], Rational Unified Process (RUP) [26], to name only the most commonly known. Many of the concepts coming from these methods are readily applied in industry more or less successfully on an intra-organizational level, e.g., the Rational Unified Process. However, development methods are not universally applicable across organizational boundaries. Due to their complexity, they are usually embedded deeply in an overall organizational context, so that their concepts are not transferable easily between customers and suppliers. Moreover, they are bound to distinct graphical notations, and particular tools, that do not necessarily permit easy exchange of information between different stakeholders, i.e., in text form.

The previously mentioned formal component composition and coordination languages elevate the reasoning about properties of component composites to a higher level of abstraction, trying to avoid a full implementation cycle. However, they seem not to have made their ways into industry, simply because industry is afraid of the high initial investment associated with the introduction of rigorous specification techniques. Koala is an exception, because it is coming out of an industrial context. Although, Koala provides syntactical mappings for object wiring only and does not consider behavior.

The model-driven software development community proposes to build a number of (UML) models for each component [5,8,37] that can be directly compared and adapted. For example, the KobrA method proposes a two-stage approach to component reuse [4], creation of a conformance map and the derivation of a semantic map. KobrA proposes to use UML models as native specification notation and requires from external component interfaces to be specified in UML. Externally procured components must therefore be described in UML in order to conform to KobrA's native representation (conformance map). The UML models of the acquired external component can then be compared with the UML models of the integrating component framework and adaptations negotiated. The semantic map is the collection of UML models that fully describes the adaptations necessary, and it can be regarded as the specification of an adapter component.

2.2 Component Feature Mapping on the Procurement Level

On the procurement level (depicted in Figure 1), natural language is the single most important communication notation [7,15]. Because of this apparent lack of formalism of the requirements, the procurement level is not readily supported by tools in any way.

In general, there are two orthogonal or supplementing ways to address the feature mapping problem on higher levels of abstraction and make component or system descriptions more amenable to automated processing.

(i) A bottom up approach to elevate the degree of formalism that is used on the implementation level up to the requirements engineering level. This is what the MDA community is currently aiming at [8], or what the component coordination languages propose: Introduction of (semi) formal descriptions on higher levels of abstraction than the implementation level and suitable model-mapping and -transformation mechanisms.

(ii) A top-down approach to introduce more structure and rigor in natural language requirements and specifications on the highest levels of abstraction. Examples of this approach comprise the usage of scenarios and templates [20,24,27,38], or controlled natural languages such as Attempto Controlled English [18,19].

Whereas, in the first approach, it is tried to apply similar formal specification techniques on higher abstraction levels that are used on the lower end of the spectrum, the second approach acknowledges the fact that natural language is the primary notation on the requirements and analysis level, and introduces structure and rigor. The goal of both approaches is the realization of mappings between stated goals and provided features, or required and provided component attributes on higher levels of abstraction along the lines that, for example, CORBA provides in terms of mappings on the programming language level [36]. More structured requirements, rigor, and formalization are the key ingredients to make component description mappings more amenable to automatic processing and automated reasoning.

In the following, we illustrate the challenges of component requirement and feature mapping on the procurement level based on an example component-based embedded system. Figure 2 shows the component tree of an existing Vehicle Alarm Terminal that can be built into vehicles operated by safety/security services such as ambulances, fire engines, police cars, or armored cars. The alarm system is used to track the location of a vehicle through GPS and Galileo (in the future), report the status of a vehicle, through radio or GSM network, issue an alarm if the vehicle leaves a predefined geographic area, or if something is wrong with the vehicle, and call help in an emergency, for example, an armed hold-up of an armored car.

Each of the components of the alarm terminal displayed in Fig. 2 is described according to a standard set of text documents. The textual specifications of the components are derived and decomposed from the system-level requirements, such as the example requirements stated in Table 1. When the system had been developed from scratch, i.e., development of the first alarm terminal system, engineers had decomposed it into components according to architectural and cohesion considerations. However, in component-based application development reuse is the key driving force, so that in subsequent versions or variations of the alarm terminal, engi-

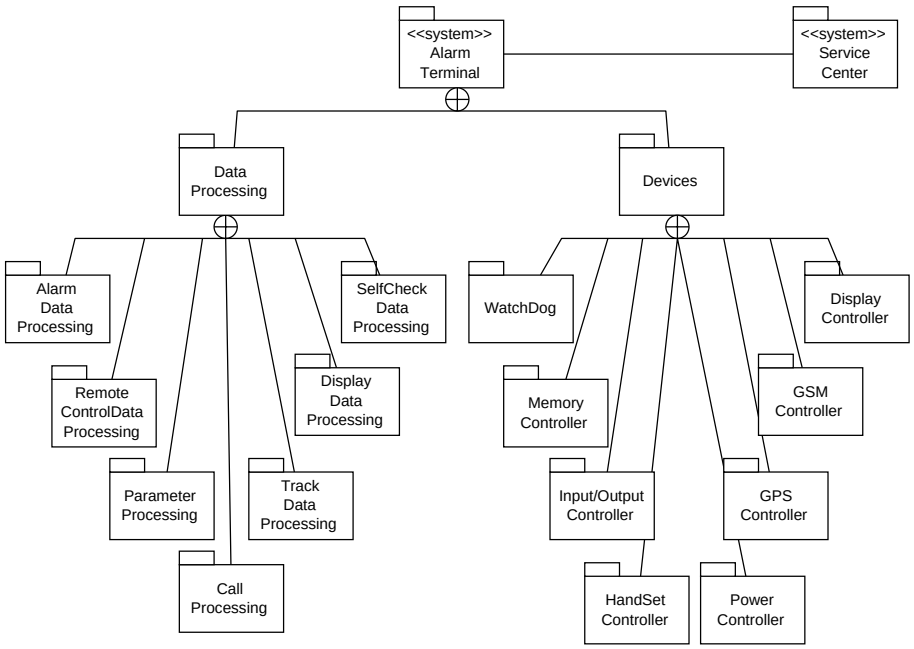


Fig. 2. Logical organization of the components for an example vehicle alarm terminal as containment tree. This organization corresponds to the Design-layer in Fig. 3.

neers will decompose the system according to existing components in the component repository. Figure 3 illustrates the mappings along a composition/decomposition dimension and an abstraction/concretization dimension that must be performed in a component-based development project. System development starts in the upper left hand side corner, with abstract system requirements [22]. These requirements also represent a high level of composition since they are describing the entire system. In a subsequent step (design), the requirements are turned into more concrete sub-system or component specifications. This is where an architecture of the system emerges and a hierarchy of logical components [4].

Initially, we would have chosen an arbitrary component hierarchy following some decomposition rules, had we developed everything from scratch, i.e., embodiment in Fig. 3 with custom development. However, since we aim to increase the degree of reuse, we have to decompose the logical hierarchy according to the constraints of existing physical components, i.e., embodiment in Fig. 3 with assembly of existing components. This is an iterative process that involves

- identification of required features from the logical component hierarchy in the repository of existing physical components
- refactoring of the logical hierarchy according to the components reused, and
- checking that all requirements have been covered by some existing implementation.

From the last item, and from the illustration in Fig. 3 it becomes apparent that component-based development is, to a large extent, a traceability problem [21]. Application engineers have to link requirements with the specified features in the design artifacts, and in the implementation. This ensures that all requirements are implemented through some physical component. The following combinations are

Req-ID	Description
R-SYS-1	Send alarm signal when driver pushes the emergency button in an emergency situation
R-SYS-1.1	The alarm button shall be pushed continually for longer than 2s
R-SYS-1.2	The alarm signal shall include alarm category, GPS position, time of alarm
R-SYS-1.3	The alarm signal shall be sent through the GSM network to the service center
R-SYS-3	The time for reacting on an alarm input shall be less than 50ms
R-SYS-4	The alarm signal shall be encrypted
R-SYS-6	The system shall continuously send alarm signal until receiving the response from service center
R-SYS-8	If the driving route is pre-set, the system shall send an alarm signal when the car deviates from its route
R-SYS-17	The system should be able to perform a self-check and send the result to service center
R-SYS 18	The system should check the main power, the GPS antenna and the emergency button every 50ms and issue an alarm signal if they are broken
R-SYS-22	The emergency alarm shall have the highest priority in case several alarm inputs happen simultaneously
R-SYS-31	The system shall have a backup battery
R-SYS-31.1	The system shall switch to backup battery when the voltage of main power is lower than the threshold value
R-SYS-31.2	The system shall charge the backup battery when it is at low voltage
R-SYS-38	Location data should be updated at least every second, and the latest data set should be stored
R-SYS-39	If the system cannot receive the GPS data, the latest data set stored should be used
R-SYS-43	The system shall be able to receive and process commands from the service center
R-SYS-48	The system shall be able to make a phone call using an extent handle
R-SYS-49	The system shall communicate with the handle through RS485
R-SYS-51	The consumed current of the system in stand-by shall be less than 10mA
R-SYS-55	The system shall run at all times (“watchdog”)
R-SYS-67	The system shall provide a 110 emergency call interface
R-SYS-71	The system shall provide a connector for navigator
R-SYS-72	The system shall provide a connector for vehicle black box recorder

Table 1
Example system-level requirements of a vehicle alarm terminal.

thereby feasible:

- One requirement is linked to one component.
- One requirement is linked to several components.
- One component implements several requirements.
- Some requirements cannot be traced to an implementation (the features will be implemented from scratch).

The traceability links may also be established bi-directionally which is of particular importance for embedded system development, in order to assess to which extent component features are not traceable to requirements. This tells us how much of the functionality provided by a reused component is actually needed in our system and how much of it is an undesired overhead.

In this section we have argued that component procurement is, to a large extent, a feature traceability problem. System requirements must be traceable to the design artifacts in the decomposition hierarchy, and to their actual physical implementations. As far as text documents are concerned as primary means for describing system and component properties, we can use advanced document analysis techniques for automated tracing and linking. How this can be done is described in the next section.

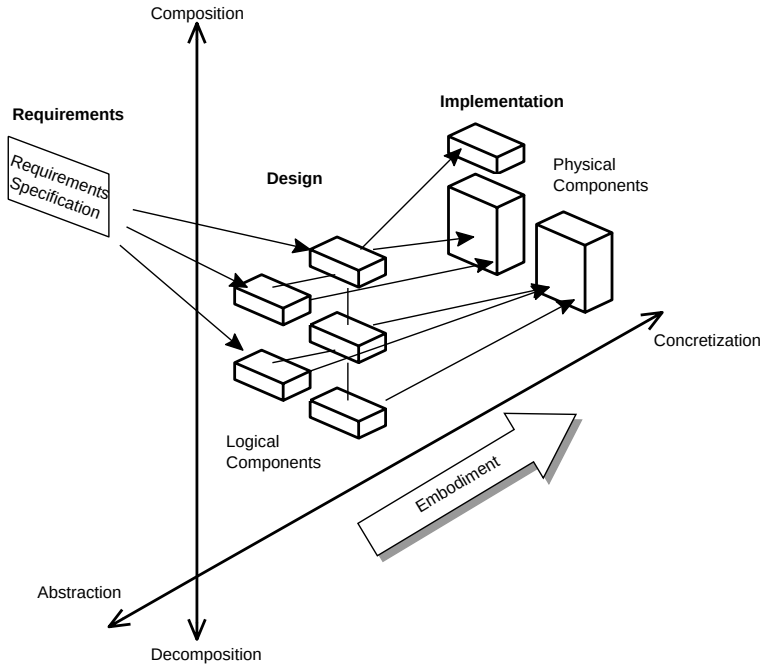


Fig. 3. Feature mapping between high-level system requirements, lower-level logical component specifications, and the existing physical components from the repository. A requirement must be traceable to its corresponding implementation throughout all development stages.

3 Reformulating Component Procurement as Document Analysis Problem

The fundamental approach of document analysis or information retrieval techniques is to "match words of queries with words of documents" [12]. The queries are formulated by a user who would like to retrieve documents of interest (e.g., from the Internet) according to the meaning of the queries.

We can use the same terminology to describe component procurement as "matching words of queries" coming from requirements, for instance, to "words of documents" describing a design; or "matching words of queries" coming from design descriptions to "words of documents" of an implementation. We can therefore reformulate the component procurement problem described earlier as a document analysis problem or an information retrieval problem [35] that is amenable to be solved by typical retrieval techniques such as Latent Semantic Analysis (LSA) [12].

3.1 Latent Semantic Analysis

LSA or Latent Semantic Indexing (LSI) is a very efficient, fully automatic mathematical/statistical technique for extracting and inferring relations of expected contextual usage of words in documents [30]. It takes advantage of implicit higher-order structure in the associations of terms with documents in order to steer the detection of relevant documents (in our case, provided component descriptions in a repository) on the basis of terms in queries (in our case required component descriptions) [12].

LSA is based on a terms-by-documents matrix that represents the occurrences

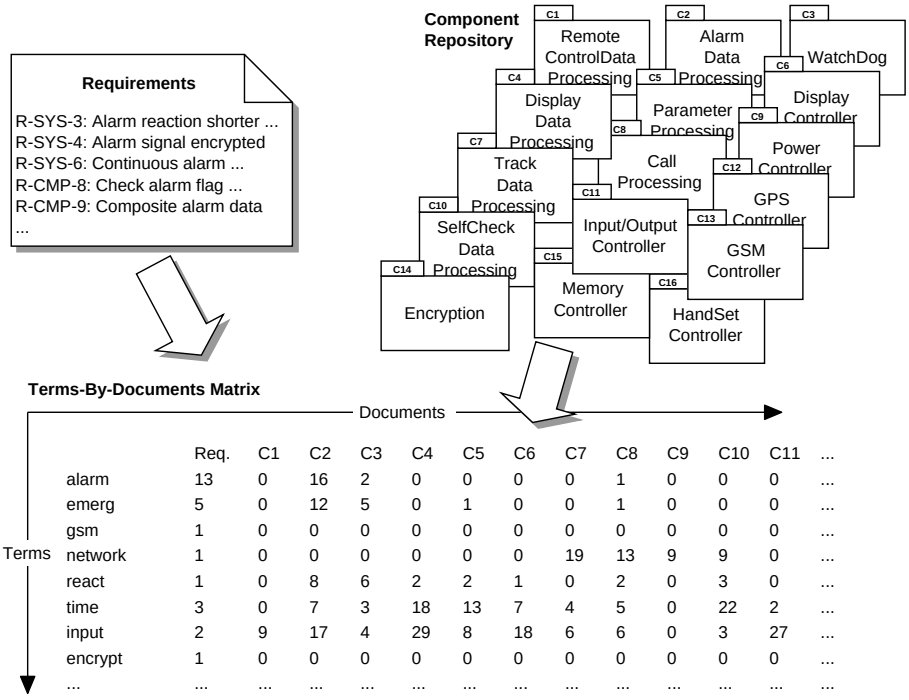
of terms in existing documents. The columns of the matrix A correspond to the documents and the rows correspond to the stemmed and normalized terms. The cells contain the number of occurrences of a term in a document. This matrix A is analyzed by singular value decomposition (SVD) to derive the latent semantic structure model [12], leading to three other matrices T, S, and D^T: $A = T_0 S_0 {D^T}_0$. T_0 and ${D^T}_0$ have orthonormal columns, representing the left and right singular vectors, and S_0 is diagonal, containing the singular values. If the singular values (S_0) are ordered according to size, the first k-largest may be kept and the remaining smaller ones set to zero, leading to a reduced approximate fit with smaller matrices [32]. The product of these new matrices $\hat{A}$ is only approximately equal to A and of rank k: $A \approx \hat{A} = TSD^T$. It represents the amended terms-by-documents matrix. The dimension reduction of the matrices is important in order to filter out sampling error and unimportant details while keeping the essential latent semantic structure intact [12]. It can be regarded as compressing the same (or similar) information that is available in the original terms-by-documents matrix in a smaller space. Taking the correlation coefficients from this matrix, finally yields the similarity between the documents. High values $[-1..1]$ represent high correlation, low values represent low correlation between documents.

3.2 LSA for Establishing Traceability Links

The techniques for establishing semantic links between the concepts of component requirements and the concepts of existing component specifications are initially coming from the software maintenance and reengineering community [13,14,32]. Here, the goal is to establish traceability links between the various development documents of existing software systems in order to make design decisions and model transformations more explicit. In component-based development, we are facing the same challenges. The semantic concepts described in system-level or component-level requirements must be traced to the corresponding concepts of a component repository. This can be done in the following steps represented in Fig. 4 and 5 [30,31,32].

(i) Definition of the traceability model. Here, we have to decide which artifacts of our development project we would like to trace, i.e. system or component requirements on one side, and component descriptions on the other side. The component descriptions may include textual descriptions, source codes, but also component tests. This is about choosing the types of documents to be used in the analysis process.

(ii) LSA. All documents belonging to one component are copied into one single text file, for term-by-component identification. All requirements that are used for component identification are copied into one document, representing the search queries. All documents are analyzed by LSA, generating a term-by-document matrix. The columns represent all documents, and the rows represent all relevant terms identified in these documents. One document contains the queries (component requirements) all other documents contain the component specifications from the repository. The cells represent the occurrence of terms per document (Fig. 4). SVD generates three new matrices (Fig. 5, $T_0, S_0, {D^T}_0$).



	Req.	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	...
alarm	13	0	16	2	0	0	0	0	1	0	0	0	...
emerg	5	0	12	5	0	1	0	0	1	0	0	0	...
gsm	1	0	0	0	0	0	0	0	0	0	0	0	...
network	1	0	0	0	0	0	0	19	13	9	9	0	...
react	1	0	8	6	2	2	1	0	2	0	3	0	...
time	3	0	7	3	18	13	7	4	5	0	22	2	...
input	2	9	17	4	29	8	18	6	6	0	3	27	...
encrypt	1	0	0	0	0	0	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...

Fig. 4. Analysis and mapping process.

(iii) Reconstruction of the terms-by-documents matrix $\hat{A}$ out of the reduced SVD-matrices (Fig. 5, $\hat{A}$). This represents the same information as the matrix A, though in a smaller sub-space, thereby filtering out irrelevant information. For example, originally, the concept "alarm" appeared 13 times in document "Req.", but LSI "estimates" that 8.4033 should be the adapted number of occurences according to the context usage of the term "alarm" in all other documents. Calculating the correlation coefficients of the reconstructed matrix yields a new matrix representing the similarity of documents: $CorrCoef(\hat{A})$ in Fig. 5. High numbers mean that the concepts from the component requirements document are also contained in a component description from the repository.

(iv) Traceability link selection and identification of the suitable components. The question here is which components are indeed implementing the requirements, or, in other words, where do we draw the line between interesting components and irrelevant components? There are several strategies for "ignoring" such links [32]: cut point (select the top n links), cut percentage (select a set part of the list), constant threshold (select those greater than, e.g., 0.7), variable threshold (select, e.g., best 20% similarity). According to the results in our example displayed in Fig. 5, our analysis method suggests that component C1 appears to be the most suitable candidate (with high probability of 0.9429), but C8 may also be considered (with lower probability of 0.6633).

$T_0 =$

0.1408	0.1106	-0.7725	0.1213	-0.5225	0.1365	-0.2546	-0.0000	...
0.1089	0.0763	-0.5037	0.0593	0.4066	-0.5226	0.5354	0.0000	...
0.0018	0.0005	-0.0221	0.0021	-0.0920	0.4767	0.5136	-0.7071	...
0.1441	-0.7783	0.0006	0.6109	0.0135	-0.0021	0.0021	-0.0000	...
0.1236	-0.0218	-0.2822	-0.0699	0.7286	0.4996	-0.3454	-0.0000	...
0.4840	-0.4768	-0.0381	-0.7194	-0.1173	-0.0647	0.0392	0.0000	...
0.8355	0.3851	0.2597	0.2935	-0.0068	0.0070	-0.0010	0.0000	...
0.0018	0.0005	-0.0221	0.0021	-0.0920	0.4767	0.5136	0.7071	...
...	...	...	...	...	...	...	...	...

$k = 3; S_0 =$

56.6436	0	0	0	0	0	0	0	...
0	28.0933	0	0	0	0	0	0	...
0	0	23.7739	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
0	0	0	0	0	0	0	0	...
...	...	...	...	...	...	...	...	...

$D'_0 =$

0.1019	0.0128	-0.5250	0.0426	-0.6482	0.4540	0.2928	-0.0000	...
0.1327	0.1234	0.0983	0.1325	-0.0087	0.0662	-0.0163	-0.1927	...
0.3909	0.2036	-0.6946	0.1027	0.2003	-0.4452	-0.2716	0.0079	...
0.1123	0.0207	-0.2033	-0.0434	0.7069	0.5160	0.3667	-0.0232	...
0.5859	0.0904	0.2642	-0.2296	-0.1207	0.0388	-0.0280	-0.6398	...
0.2354	-0.1098	0.0216	-0.3555	0.0404	-0.3244	0.6060	0.1012	...
0.3275	0.1271	0.1735	0.0089	-0.0304	0.1811	-0.1577	0.2233	...
0.1710	-0.5121	0.0596	0.5264	-0.0358	-0.2704	0.3333	-0.0103	...
0.1731	-0.3577	-0.0196	0.3084	0.1264	0.3188	-0.3391	-0.0798	...
0.0229	-0.2493	0.0002	0.2759	0.0173	-0.0202	0.0329	-0.2262	...
0.2617	-0.5840	-0.0379	-0.4845	-0.0416	0.0800	-0.2791	0.3632	...
0.4153	0.3362	0.2917	0.3254	-0.0593	0.0626	0.0884	0.5507	...
...	...	...	...	...	...	...	...	...

$\hat{A} =$

8.4033	4.2244	16.6394	-2.6378	1.5174	4.3494	-2.7037	-2.5715	...
5.5935	2.7813	11.3469	-1.7767	1.6187	2.3882	-1.4810	-1.7749	...
0.2173	0.1098	0.4169	-0.0600	-0.0391	0.1866	-0.1130	-0.0096	...
-2.0765	-2.5957	6.4256	2.5521	5.1038	-2.2644	2.7504	18.3839	...
3.2544	1.3799	8.3980	-0.9059	3.1904	-0.1523	0.2383	0.5306	...
1.3678	-1.1172	16.3386	0.5160	17.4719	-11.1565	7.9988	11.2646	...
3.8431	0.6839	19.4945	-2.8134	32.0062	-24.9504	15.3553	-9.1094	...
0.2173	0.1098	0.4169	-0.0600	-0.0391	0.1866	-0.1130	-0.0096	...
...	...	...	...	...	...	...	...	...

$corrcoef(\hat{A}) =$

	Req	C1	C2	C3	C4	C5	C6	...
Req	1.0000	0.9429	0.6253	-0.8798	0.0379	0.1089	-0.1619	...
C1	0.9429	1.0000	0.3330	-0.8551	-0.2225	0.3317	-0.3977	...
C2	0.6253	0.3330	1.0000	-0.5492	0.6994	-0.5509	0.5475	...
C3	-0.8798	-0.8551	-0.5492	1.0000	-0.2695	0.1881	-0.1094	...
C4	0.0379	-0.2225	0.6994	-0.2695	1.0000	-0.9816	0.9789	...
C5	0.1089	0.3317	-0.5509	0.1881	-0.9816	1.0000	-0.9956	...
C6	-0.1619	-0.3977	0.5475	-0.1094	0.9789	-0.9956	1.0000	...
C7	-0.6433	-0.7199	-0.2254	0.9139	-0.1930	0.1883	-0.0963	...
C8	0.6633	0.7412	0.2295	-0.9212	0.1737	-0.1646	0.0727	...
C9	0.5873	0.6603	0.2134	-0.8901	0.2432	-0.2499	0.1585	...
C10	0.5609	0.6320	0.2075	-0.8775	0.2652	-0.2771	0.1860	...
C11	-0.5849	-0.6577	-0.2129	0.8890	-0.2453	0.2525	-0.1610	...
...	...	...	...	...	...	...	...	...

Fig. 5. SVD $(T_0, S_0, D^T{}_0)$, dimension reduction $(S_0, k = 3)$, matrix reconstruction $(\hat{A})$ and correlation coefficients $(CorrCoef(\hat{A}))$.

4 Experiments with LSA and Evaluation

We have applied LSA to a number of tracing problems of different size and complexity, and, in the following, we will discuss our findings and our experience with the technique. It is important to note that, for assessment, we require existing systems for which the links between various development artifacts are already known. In other words, we should apply LSA to legacy systems for which we already know the traceability links between all artifacts (from requirements to implementation), so that we can use this knowledge as a benchmark to assess the performance of LSA for finding (reconstructing) those links. Otherwise, an assessment of the results is difficult as we have seen in one of the case studies performed. As preprocessing tool, before we apply LSA, we use the TMG Matlab toolbox by Zeimpekis and Gallopoulos [45]. It comes equipped with a number of functions for pre-processing, such as stop-word elimination and stemming, and it generates the term-by-document matrices.

4.1 PacMan Case Study

One of the first case studies in which we applied LSA to reconstruct traceability links is a PacMan game used in the Computer Science Bachelor curriculum at Delft University of Technology [31]. The available documentation comprises 10 requirements documents, coming in the form of use case descriptions, 19 documents describing the design of the game (class hierarchy) and 17 documents with test descriptions, and the Java implementation. All information is available as plain text, including the Java code. The key-words of the programming language do not carry any semantic significance and can be eliminated simply by adding them to the list of stop words. They are then filtered out by TMG. Alternatively, we can use Doxygen [16] to generate documentation out of the source codes and use this as a replacement for the sources [32].

We simply included all available documents in our analysis, leading to a corpus of some 1200 relevant terms across all documents. We chose the best 20% of all similarity measures as traceability links. The value for k (matrix reduction) was varied between 10% and 20% in order to assess the effect of the choices on the selection of links. Best results were achieved with k set to 20%. In that case, LSA was able to identify 16 out of 17 traceability links that had been initially defined by the developer of the program, although LSA found many more links (false positives). This was due to the fact that the use cases described in different documents, leading to different implementations are dealing with similar program events, e.g., one with restarting the game after suspension of the game and one with restarting the game after game over. Both requirements describe similar concepts, and they are therefore linked by LSA. The same we found for requirements describing the move of the player and the move of the monsters, or the descriptions about a player bumping into a monster and a monster bumping into the player. All these requirements are describing similar concepts and are therefore linked by the tool. The link which was not identified by LSA was the description of the GUI of the PacMan and its corresponding test suite. An analysis of the requirements document of the GUI and the corresponding test document revealed a mismatch of concepts between

the two artifacts. Whereas the requirements describe the layout and behavior of the graphical elements of the GUI, the test description is about what a person testing the GUI should look at and which elements should be clicked on. The test description of the GUI was therefore linked to most of the other documents, because their concepts appear in the test descriptions. They were not linked to the GUI requirements document, because the concepts described there are different from the ones in the test description, so that LSA does not link them.

4.2 Callisto Case Study

Another case study we conducted at the Technical University of Eindhoven with a software system called Callisto. It is a software engineering tool that can be used to specify component interfaces. We looked at three classes of documents, user requirements specification, design documents and acceptance test plan, and tried to link those with each other and the implementation. In the experiments including the source code, we ended up with 5500 relevant terms. The parameters of the tool were set to the same values as for the PacMan case. LSA was able to trace 63% of the requirements into the code correctly and 94% of the requirements into the test specification accurately. Linking the requirements to the code produced many false positives. Hence, the low rate of correctly recovered links. It is important to note that the requirements and test descriptions had explicit links through unique identifiers. It was therefore possible to trace the requirements to their respective test documents easily, so that an evaluation of the results for the requirements-to-test tracing was straightforward.

Further, we found that LSA had more difficulties to establish the links between the requirements and the design documents, than it had for linking the test suites with the requirements. This can be attributed to the fact that many of the design documents contain UML models in the form of pictures capturing many of the essential concepts. The models were not included in our text-based analysis, so that the concepts described there would not make it into the term-by-document matrix. LSA could therefore only consider part of the information contained in the design documents, leading to much weaker links, and thus the low value of 68%.

4.3 Philips Case Study

With Philips Applied Technologies, we carried out a case study in which we tried to link requirements to component descriptions and test descriptions for part of a DVD recorder software. The initial question was whether or to which extent all requirements agreed in the contract were actually implemented in the end product, in particular, when new requirements or change requests are emerging. This can be seen as the most realistic case in which we tried to map descriptions of functionality to actual component descriptions and assess how well the components cover the required functionality. In addition, we had no explicit traceability matrix produced by the developers of the system available, so that a final assessment and drawing exact conclusions from the case study was difficult. However, it provided many new insights into the performance of LSA for component feature mapping.

There were requirements on different levels of abstraction available and it was

not obvious which of the hierarchy would be the most suitable. For the analysis, we decided to include the first and second highest level of abstraction. Lower-level requirements seemed to include too many details that could not be traced into the component descriptions. Before we carried out LSA, we tried to find explicit links between the documents, aiming to come up with a manually produced traceability matrix. We transformed the text documents into an XML format and performed some text queries using Xpath expressions [9]. However, we could neither uncover the unique identifiers of the requirements, nor the labels of the requirements in any of the other documents (e.g., as a comment).

LSA was more effective in that respect. 20 artifacts were analyzed, all coming in the form of text documents. Preprocessing of the artifacts resulted in 2300 terms in the terms-by-documents matrix.

A noticeable outcome from the experiments was the much higher predicted similarity of concepts between the requirements and the component descriptions than the similarity between the requirements and the test descriptions produced by LSA. In the other two case studies, it was the other way round. Apparently, the component descriptions were linked well to the corresponding requirements because every component comes equipped with a general high-level description of its functionality that is expressed in an abstract way similar to the high-level requirements. And, obviously, the test descriptions were linked poorly to the requirements, because the tests were defined according to the low-level design descriptions of the components which did not correspond to the high-level requirements. This mismatch in abstractions makes the importance of a well defined tracability model apparent.

We could not derive concrete results about the performance of LSA in this case because we lacked definite links provided by the developers of this system. They found that many links, indeed, had been identified correctly in our analysis, but for many other links they would not agree.

4.4 Discussion of the Results

Working with the cases presented, provided a lot of insight in how LSA can be applied to linking various types of available documents in a typical software development process. Our primary aim here is to link system level requirements or component level requirements, coming from the decomposition hierarchy of a system, to respective candidate components in a repository by using latent semantic analysis. However, in the experiments we used all available kinds of documents including high- and low-level requirements, intermediate design documents as well as test descriptions and source code. For LSA it does not matter which documents are belonging to which types of artifacts. It simply tries to guess links between all documents included in an analysis based on an underlying semantic structure inherent in these documents. It is our responsibility to attribute the various types of documents to one distinct entity, i.e., one component. This can be done through copying all relevant information into a single file that represents one component description. LSA will link whatever concepts it finds in other documents that are similar to the concepts of our component description to that particular component. It is, therefore, quite robust with respect to the kind of information provided for each component, as long as it comes in textual form.

In the Callisto case study we had many UML diagrams available, apparently containing essential concepts that were not considered in the analysis. This lead to poor linking of concepts in these documents. A textual description would probably lead to much better results. Graphical notations are more and more being used in industry because people can grasp the essentials of such documents more easily. In the future we will have to look at how we can extract this information automatically and make it available in textual form.

It was interesting to see how well test cases could be traced from requirements. Test cases, especially system level tests and acceptance tests, are usually devised according to the information found in the requirements documents. They are therefore very likely to incorporate similar concepts. After all, they represent implicit links between the implementation (execution of tests) and the outcome of the tests (oracle) coming from high-level requirements or design documents. LSA can make this implicit semantic similarity explicit. Consequently, we claim that component specifications should always come together with their respective test suites according to the tester components described in [22,23].

We also observed that low-level implementation-specific test cases could not be traced well to high-level requirements. This was somewhat surprising, since abstraction is the single most important technique for us humans to deal with complex entities, and we expected that we would use the same semantic concepts on higher levels of abstraction that we use on lower levels, though, just getting rid of the details. Apparently, that is not the case, and we have to understand the mechanics of abstraction better.

5 Summary, Conclusions and Future Work

In this article, we have introduced a novel technique for automatically linking requirements to component specification documents through applying latent semantic analysis. Being able to trace concepts that are essential in an application development project to a collection of component descriptions in a repository is the prerequisite for automated component feature detection and analysis. So far, we can only identify the required essential concepts of an application in a component repository, and we can create links in the form of a terms-by-documents matrix to the documents describing the components. However, the links are weighted (coming with a probability), so that we can constrain the number of suitable components to only a few, compared with the potentially huge number of components in a repository.

LSA helps us to identify few relevant components out of a large repository. The experiments that we performed are quite promising with that respect. LSA does not provide support for the next step in component procurement, the assessment of the likely adaptations to be carried out. At this moment we have no answer to this next problem.

For the future, we are planning to perform many more case studies using varying types of documents. It would be interesting to see how more structured documents such as use case descriptions and other templates [27,38], that are more and more used in industry, affect LSA. Will such structures improve its performance or will they have a negative effect? The same applies to more formalized documents, such

as requirements containing logic and formulae. We have seen already how UML diagrams can inhibit the text-based LSA technique. Is that going to be the same with formal expressions? Another issue that we will look at in the future is how we can extract textual concepts from diagrams that are used in industry [8].

References

[1] P. Allen and F. Frost. Component-Based Development for Enterprise Systems: Applying the Select Perspective. Cambridge University Press, 1998.

[2] F. Arbab. Abstract behavior types: A foundation model for components and their composition. In F.S. de Boer and et al., editors, Lecture Note in Computer Science, volume 2852, Springer, 2003.

[3] Attempto Project. Attempto Controlled English. http://www.ifi.unizh.ch/attempto.

[4] C. Atkinson, and others. Component-based Product Line Engineering with UML. Addison-Wesley, 2002.

[5] C. Atkinson and H.-G. Gross. Model Driven, Component-Based Development. In: Business Component-Based Software Engineering. Franck Barbier (Ed.), Kluwer, 2003.

[6] C. Atkinson, C. Bunse, H.G. Gross, C. Peper (Eds). Component-Based Software Development for Embedded Systems. Lecture Notes in Computer Science, vol. 3778, Springer, Heidelberg, 2005.

[7] D.M. Berry and E. Kamsties. Ambiguity in requirements specification. In J. Leitner and J. Doorn (Eds), Perspectives on Software Requirements, pp. 7–44. Kluwer, 2003.

[8] M. Born, I. Schieferdecker, H.-G. Gross, P. Santos. Model-Driven Development and Testing. 1st European Workshop on MDA with Emphasis on Industrial Applications, Enschede, Netherlands, March 17-18, 2004.

[9] J. Clark and S. DeRose. XML path language (Xpath) version 1.0. Technical Report, W3C, November 1999.

[10] D. Coleman et al. Object-Oriented Development – The Fusion Method. Prentice Hall, 1994.

[11] B.J. Cox. Planning the Software Industrial Revolution. IEEE Software, Vol. 7, No. 6, pp. 25–33, November 1990.

[12] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by Latent Semantic Analysis. Journal of the American Society of Information Science, 49(6), pp. 391–407, 1990.

[13] A. DeLucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an Artefact Management System with Traceablity Recovery Features. In: Proc. of the 20th IEEE Int. Conf. on Software Maintenance, pp. 306–350, IEEE Computer Society, 2004.

[14] A. DeLucia, F. Fasano, R. Oliveto, and G. Tortora. A Traceability Recovery Tool. In: Proc. of the 9th European Conf. on Software Maintenance and Reengineering, pp. 32–41, IEEE Computer Society, March 2005.

[15] C. Denger, D.M. Berry and E. Kamsties. Higher Quality Requirements Specifications through Natural Language Patterns. IEEE International Conference on Software-Science, Technology & Engineering, Herzelia, Israel, November 2003.

[16] Doxygen Project. http://www.doxygen.org.

[17] D.F. D'Souza and A.C. Willis. Objects, Components, and Frameworks. Addison-Wesley, 1998.

[18] N. E. Fuchs, U. Schwertel and S. Torge. Controlled Natural Language Can Replace First-Order Logic. Proceedings ASE '99, 14th IEEE International Conference on Automated Software Engineering, Cocoa Beach, Florida, October 1999.

[19] N. E. Fuchs and U. Schwertel. Reasoning in Attempto Controlled English. In: F. Bry, N. Henze and J. Maluszynski (eds.): Principles and Practice of Semantic Web Reasoning, International Workshop PPSWR 2003, Mumbai, India, December 2003. Lecture Notes in Computer Science, Vol. 2901, Springer, 2003.

[20] M. Glinz. Improving the Quality of Requirements with Scenarios. In: Proc. of the 2nd World Congress on Software Quality, pp. 55–60, Yokohama, Japan, 2000.

[21] O.C. Gotel and A. Finkelstein. An Analysis of the Requirements Tracability Problem. In: Proc. of the First IEEE Intl. Conference of Requirements Engineering, Colorado Springs, April, 1994.

[22] H.-G. Gross. Component-based Software Testing with UML. Springer, Heidelberg, 2004.

[23] H.-G. Gross, M. Melideo, A. Sillitti. Self-Certification and trust in component procurement. Science of Computer Programming, Vol. 56, No. 1-2, pp. 141–156, April 2005.

[24] H. Holbrook. A Scenario-Based Methodology for Conducting Requirements Elicitation. ACM Software Engineering Notes, Vol. 15, No. 1, pp. 95–104, 1990.

[25] J. Ivers, N. Sinha, and K. Wallnau. A basis for composition language CL. Technical Report CMU/SEI-2002-TN-026, Software Engineering Institute (SEI), September 2002.

[26] I. Jacobson, G. Booch, and J. Rumbaugh. The Unified Software Development Process. Addison-Wesley, 1999.

[27] E. Kamsties, A. von Knethen, B. Paech. Structure of QUASAR Requirements Documents. Fraunhofer IESE Report No. 073.01/E, November 2001.

[28] K.C. Kang et al. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Software Engineering Institute (SEI), November 1990.

[29] J. Kontio. OTSO: A Systematic Process for Reusable Software Component Selection. Technical Report CS-TR-3478, Department of Computer Science, University of Maryland, 1995.

[30] T.K. Landauer, P.W. Folz, and D. Laham. An Introduction to Latent Semantic Analysis. Discourse Processes, 25, pp. 259–284, 1998.

[31] M. Lormans and A. van Deursen. Reconstructing Requirements Coverage Views from Design and Test using Traceability Recovery via LSI. In: 3rd Intl. Workshop on Traceability in Emerging Forms of Software Engineering, pp. 37–45, Long Beach, November 2005.

[32] M. Lormans and A. van Deursen. Can LSI help Reconstructing Requirements Traceability in Design and Test? In: 10th IEEE Conference on Software Maintenance and Reengineering, Bari, Italy, March 22–24, 2006.

[33] M. Lormans, H.-G. Gross, A. van Deursen, R. van Solingen, and A. Stehouwer. Monitoring Requirements Coverage using Reconstructed Views: An Industrial Case Study. In: Proc. of the 13th Working Conf. on Reverse Engineering (WCRE'2006), pp. 275–284, Washington, October 2006.

[34] M. Lumpe et al. Towards a formal composition language. In Workshop on Foundations of Component-Based Systems, Zürich, September 1997.

[35] Y.S. Maarek, D.M. Berry, and G.E. Kaiser. An Information Retrieval Approach for Automatically Constructing Software Libraries. IEEE Transactions on Software Engineering, 17(8), pp. 800–813, August, 1991.

[36] Object Management Group (OMG). History of CORBA. Technical Report, www.omg.org, 1997 – 2004.

[37] Object Management Group (OMG). Model Driven Architecture. http://www.omg.org/mda.

[38] S. Overhage. UnSCom: A Standardized Framework for the Specification of Software Components. In Weske and Liggesmeyer (Eds), Object Oriented and Internet-Based Technologies, Springer Lecture Notes in Computer Science, Vol. 3263, Heidelberg, 2004.

[39] T. Reenskaug, P.Wold, and O. Lehne. Working with Objects: The OORAM Software Development Method. Manning/Prentice Hall, 1996.

[40] P.J. Robinson. Hierarchical Object-Oriented Design. Prentice Hall, 1992.

[41] J. Rumbaugh et al. Object-Oriented Modeling and Design. Prentice Hall, 1991.

[42] B. Selic, G. Gullekson, and P. Ward. Real-Time Object-Oriented Modeling. Wiley, 1994.

[43] C. Szyperski. Component Software – Beyond Object-Oriented Programming. Addison-Wesley, 2002.

[44] R. van Ommering et al. The KOALA component model for consumer electronics software. IEEE Computer, 33(3), 2000.

[45] D. Zeimpekis and E. Gallopoulos. Design of a Matlab toolbox for term-document matrix generation. Technical Report HPCLAB-SCG 2/02-05, High-Performance Information Systems Laboratory, University of Patras, http://scgroup.hpclab.ceid.upatras.gr/scgroup/Projects/TMG, 2005.

TUD-SERG-2007-002
ISSN 1872-5392