System architecture induces document architecture

Peter Henderson, Nishadi De Silva
School of Electronics and Computer Science, University of Southampton
Southampton, UK
p.henderson@ecs.soton.ac.uk, n.desilva@ecs.soton.ac.uk

Abstract

The documentation of an architecture is as important as the architecture itself. Tasked with communicating the structure and behaviour of a system and its constituent components to various stakeholders, the documentation is not trivial to produce. It becomes even harder in open, modular systems where components can be replaced and reused in each progressive build. How should documentation for such systems be produced and how can it be made to easily evolve along with the system it describes? We propose that there is a close mapping between the system architecture and its documentation. We describe a relational model for the architecture of open systems, paying close attention to the property that certain components can be reused or replaced. We then use ideas from storytelling and a discourse theory called Rhetorical Structure Theory (RST) to propose a narrative-based approach to architecture documentation; giving both a generic narrative template for component descriptions and a RST-based relational model for the document architecture. We show how the two models (system and documentation) map onto each other and use this mapping to demonstrate how document fragments can be stored, automatically extracted and collated to closely reflect the system's architecture.

Keywords

System architecture, documentation, narratives, RST

1. INTRODUCTION

An architecture is the partitioning of a whole into parts (components), with specific relationships between these parts [1-3]. There is an increasing need for faster software development, and much of this is now dependent on modular architectures with reusable components that allow for quicker evolution and localised updates [4]. Documenting the architectures of such evolving systems is not trivial. Of all the potential stakeholders, we are concerned primarily with the documentation required by developers who are charged with evolving the product. So, the question we ask is - how does one produce documentation for a developer who has to revise the software and thus use most of its documentation?

There are various techniques and guidelines on how to document architectures [1, 5-7]. Our approach, however, looks at this problem from a narratives perspective based on the hypothesis that 'saying it like a story' improves document coherence and readability. There are two issues that need to be considered: each component needs to be documented well and coherently; and, secondly, these component descriptions need to be collated in some way to produce the documentation for a system. For the first, we argue that a document conveys an implicit narrative (or story) to the reader, and that fine-tuning this improves the overall document. We use ideas from Rhetorical Structure Theory (RST) [8] to study and enhance the coherence of this implicit narrative (which we call a **document narrative or DN**) [9]. In this paper, we present a generic DN to document a component's structure and behaviour.

To address the second issue, we develop a relational model for the system architecture (comparable to other relational models in this field [10]) and a RST-based relational model for the document architecture, and show how the two map onto each other. We use this mapping to describe how aspects of the system architecture can be used to guide the structure and sequence of the documentation.

A mapping between the two models as shown here has two major benefits. Firstly, it allows a database to be created that can store the architecture details and the set of associated document fragments. When queried, it is able to return a narrative-based document that reflects the system architecture. Better still, it allows documentation to be reused or replaced where appropriate. Secondly, since there is a strong correlation between the models, system architects will be forced to think of the accompanying documentation from an early stage which will benefit both the system and the documentation. We conjecture that architectures that are easier to document using our technique are better architectures.

The rest of this paper is set out as follows: Section 2 gives some background information; section 3 introduces the relational model for the system architecture and a generic DN for documenting a component; section 4 presents the RST-based relational model for the documentation and illustrates the mapping between the two models; in section 5, we demonstrate our ideas using a simple example and section 6 concludes the paper and discusses future work.

2. BACKGROUND

A significant proportion of a software architect's time is spent interacting with stakeholders and communicating the architecture [11]. A majority of this communication is done via documentation. Architecture documentation is expected to cater to three categories of readers: those selecting this system, those learning to develop typical applications using this system and those intending to modify its architecture

[6]. The work presented in this paper addresses documentation targeted at the third category (even though it could be of use to the first group too).

Because architectures can be so complex, several practitioners and researchers have developed techniques that divide the documentation into **views** which help separate the different aspects of the architecture [1, 5, 12]. The documentation is then composed of the relevant views along with any documentation that applies to more than one view (the 'glue' that binds the views together). Similarly, Kruchten introduced a 4+1 model for an architecture [5] which is a generic way to describe architecture using five concurrent views, each addressing a specific set of concerns important to different stakeholders: the logical view, process view, physical view, development view and a fifth view that contains use cases or scenarios.

We recognise from these previous approaches that it is impossible to capture everything about an architecture in one document. We, therefore, abstract away from development and physical details to a much higher level. At this level, we only focus on descriptions about the software components, what they are made up of and how they interact with other components. We recognise that other audiences may require other types of documentation but they are beyond the scope of this paper.

We are also not the first to employ documenting strategies from another domain in architecture documentation. The **pyramid principle** [13], for instance, has been used to structure architecture documentation [6]. The pyramid principle is based on structuring the document around developing a question-answer dialogue with the reader. So, information is exposed incrementally as answers to questions that arise in the reader's mind. Also, **storyboarding** has been used to identify requirements and select COTS components [14]. In this paper, we make use of our previous work on **narrative-based writing** [9] and apply it to architecture documentation. This combination of narratives and RST in this domain is a novel approach. (A brief introduction to RST is given in section 4 and the features of narrative-based writing required for this paper are included where necessary. More can be found in [9].)

3. A RELATIONAL MODEL FOR SOFTWARE

As with most architectural descriptions, the central concept in our model is a **component**. A component can either be atomic or have subcomponents plugged into appropriate **slots**[1]. These subcomponents, in turn, can be made of sub-subcomponents and so on. This continues until a level is reached where the components can be considered as 'black boxes' (i.e., it is unnecessary and beyond the scope of the documentation to dwell deeper into the hierarchy of decomposition). This leads us to the first relation in our model:

contains (container:component, slot, component:component)

Components also have dependencies on other components. This is, in fact, essential for modular systems where the behaviour of the whole is only realised when the constituent components work together. We call this the *uses* relation. Component A *uses* B if A (user) uses an interface provided by B (service).

uses (user:component, service:component)

A particular benefit with open, modular architectures like the ones we focus on is that a component can be replaced by another component if it provides the similar functionality and interfaces. This can happen, for example, when two suppliers manufacture comparable components leaving the implementer to pick one depending on other criteria such as price and reliability. Of course, this option to replace usually works only in one direction. A superior component B' that can perform all the functions of an inferior component B (and more) can be used to replace B. However, B cannot be used in situations where a B' is required. This brings about the third relation *replaces*:

replaces (superior:component, inferior:component)

A diagrammatic representation of the three relations is shown below.

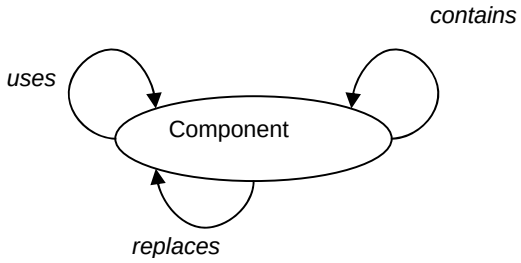

Fig. 1. Diagrammatic representation of our relational model for the system architecture

We realise that, when compared to languages such as UML with numerous relations, our model may appear limited. However, for the purposes of this paper, the model given above is sufficient.

4. A RELATIONAL MODEL FOR DOCUMENTATION

In our previous work, we have researched and developed a technique called narrative-based writing [9] to improve the coherence of technical documents such as research proposals. The technique required authors to first formulate a "document narrative" (DN): an explicit précis of what the authors wanted to convey to the readers in a story-like

[1] The idea of a 'slot' gives us a the flexibility to have multiple subcomponents of the same type plugged into different slots within the same component.

form. The DN is then analysed using a discourse theory called Rhetorical Structure Theory (RST) [8]. RST helps add more meaning and supportive reasoning to the DN and also gives an indication of how well it is structured. The DN and the corresponding RST analysis are then used to produce the document. The technique was particularly useful in collaborative writing where multiple authors had differing opinions about the document's objectives and structure.

We use this technique here to compose fragments of documentation corresponding to the components in the architecture. However, before proceedings, it is necessary to give a brief overview of RST and how it can be applied to text.

Rhetorical Structure Theory (RST)

RST was developed in 1988 by Mann and Thompson [8]. The theory attributes the coherence of a text to implicit logical relationships that exist between parts (usually called segments) of that text. So, for instance, segment A and B can be involved in a MOTIVATION relationship which means that segment B provides some information to motivate the action(s) in segment A. In Mann and Thompson's original paper, they define 23 such relationships with precise definitions of the sorts of text that can be involved in each.

In RST, the segments of text are classified as nuclei or satellites. Nuclei are considered essential to the understanding of the text. Satellites provide supporting material to the nuclei but are not absolutely necessary. Most relationships exist between a nucleus and a satellite. Returning to the example of the MOTIVATION relationship before, it can be illustrated using the diagram below. Note that the arrow always goes from the satellite to the nucleus.

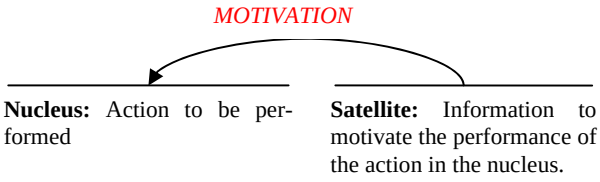

Fig. 2. A MOTIVATION relationship in RST

Some relationships like SEQUENCE can exist between more than two segments of equal importance (so, two or more nuclei). We have briefly described the RST relationships that appear in this paper in Table 1.

In order to do a RST analysis, the first step is to divide the text into segments. Each segment should have functional integrity and is often a clause or a sentence. The next step in a bottom-up analysis is to identify relationships that exist between pairs of segments. Segments involved in a relationship can, in turn, become involved in another relationship. Hence, the process is recursive and continues until all the segments can be assembled into a tree of relationships called a RS-tree. Mann and Thompson conjecture that if a RS-tree can be formed involving all the segments, then the text is coherent. However, if there are non-sequitors or difficulties producing this tree, then the text may need restructuring. This is a valuable guide when evaluating the structure and coherence of a text [8].

Relationship	Description
Background	Satellite provides background information to the nucleus
Elaboration	Satellite elaborates the information in the nucleus
Justify	Satellite justifies the information presented in the nucleus
Motivation	Satellite motivates the reader to perform the action in the nucleus
Sequence	Multiple nuclei that follow each other in sequence
Restatement	Satellite is a restatement of the information in the nucleus

Table 1. The RST relationships used in this paper

A Narrative-based Component Description

We look first at applying the narrative-based writing technique to describing each component. What we want to end up with is a generic structure that can be used for all components. Bearing in mind that a 'component' in our case can mean anything from a composite system to an atomic sub-component, some of the key concepts that need to be conveyed in the documentation are its behaviour, subcomponents (if any), whether it is able to interact with other components and, if appropriate, brief comparisons to similar products that are available. However, what is the best order to place this information in? This is where a DN can help. Trying to construct a narrative helps identify the natural sequence to the information and even recognise segments that are missing. A generic DN for the component descriptions (divided into 7 segments) is presented below along with a possible RST analysis of it. We say "a possible analysis" because it is viable that different analysts will produce different RS-Trees. The important point is to agree with the co-authors on the analysis and be able to form a tree (see Figure 3) which helps gauge the level of coherence of the text.

"[Select component X][1] [because it meets the set requirements and has some advantages over comparable technologies in the market.][2] [It is also a vast improvement from previous versions.][3] [It can receive the following instructions and perform the necessary tasks in response.][4] [The behaviour was grouped as it is done in this component for several good reasons.][5] [Furthermore, X can also interact with other components that it needs to in the following ways to produce the desired effect.][6] [On closer inspection, X is composed of multiple subcomponents that, when combined, enable its functionality. These components are x_1-x_n and they will be described later.][7]"

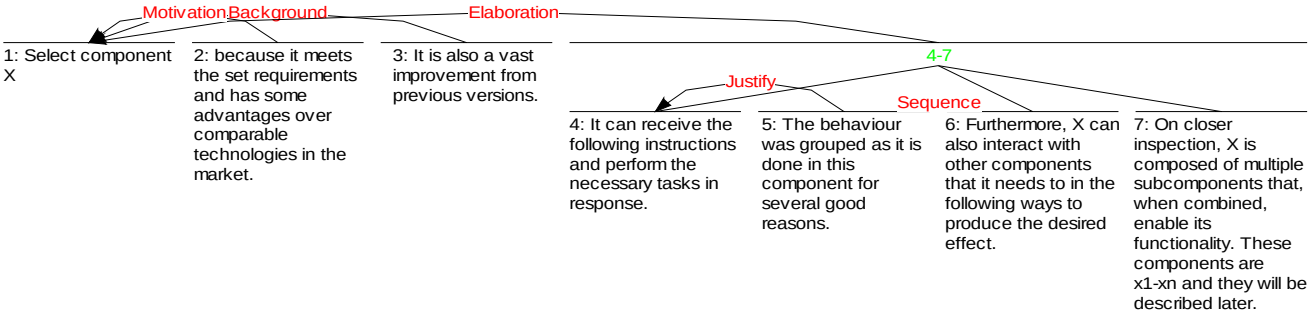

Fig. 3. A possible RST analysis of the generic DN above

Some parts of the narrative may not apply to all components of course. For instance, when describing components that are not going to be further decomposed, segment 7 about subcomponents is not relevant. Segment 2 is seen to provide motivation to convince the reader to choose (or buy) component X in the case where a decision has not yet been made.

It is worth mentioning that this narrative structure applies to the *body* of the document. Additionally, there would be other sections such as the introduction and conclusions which are compulsory in most documents. We call the description of a component adopting this narrative a **FRAGMENT**. A fragment is a self-contained description of an architectural component. Note that a fragment will be divided into several segments prior to doing a RST analysis. For a structured component, the fragments describing its contained components will be organised into a narrative structure where the fragments at the lower level are taken to be RST segments at the higher level.

A Relational Model for Document Architectures

From the above, we see that, for an architecture involving many different components at different levels in the hierarchy, there will be as many document fragments. For a document about the architecture, several of these fragments will need to be placed in a suitable order. Our eventual target is to develop a system where document fragments can be automatically extracted according to the architecture. To this end, we have developed a relational model for the documentation that corresponds to the system architecture. The novelty about this model is that these relations are also from RST. A fragment is central to our documentation model. Conceptually, this is similar to the component in the system architecture model.

Firstly, it needs to be noted that a fragment can be made up of other fragments. This is similar to the *contains* relation in the system architecture except that in the document model, a fragment's *narrative* is composed of other fragments' *narratives*. So, the topmost fragment will contain an description of the system and this is elaborated by fragments about overview of the system which is expanded by subsequent fragments (like sub-sections). We equate this to the RST ELABORATION relationship.

elaboration (fragment, fragment)

For components at the same level, the corresponding fragments need to be presented in an appropriate sequence. We propose using the *uses* relationship from the system architecture to determine the sequence. So, if component A uses component B, then we propose that the most suitable way to document it is to make fragment(A) appear before fragment(B). We call this second relationship SEQUENCE (also a RST relationship). We need to break loops in the *uses* relation by a suitable forward-reference mechanism. We recognise that even then the *uses* relation is only a partial order, but it seems not to matter which order unrelated fragments appear, as long as all the descriptions of the components that use them appear first.

sequence (fragment, fragment)

If components can be replaced by other components, it must be the case that the corresponding fragments can be replaced too. However, it is important to note that the replacement of document fragments works in the *opposite direction* to the replaces in the system architecture. Say, for instance, a newer component A' with more functionality is used to replace component A in a build. However, if fragment(A') is not yet ready, it is still possible to use fragment(A) in this case because only the capabilities of A are expected and realised. However, fragment(A) cannot be used in an instance where A' is required because it will not describe the extended functionality. The closest relationship in RST for this is RESTATEMENT. In RST, this means that one segment says the same thing as another in a different way.

restatement (fragment, fragment)

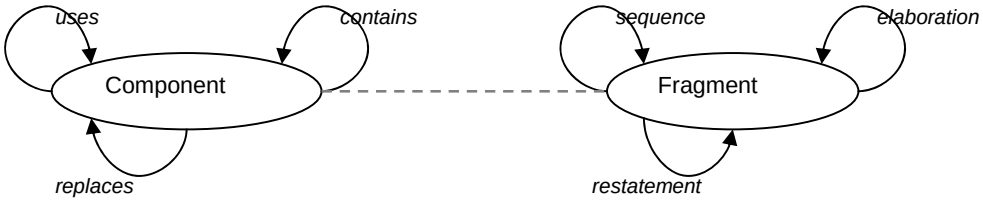

Fig. 4. The mapping between the system architecture model (left) and the document architecture model (right)

The figure above shows the mapping between the system architecture model and the document architecture model.

5. A SIMPLE EXAMPLE

We demonstrate the storage and extraction of document fragments using a simple example of a toaster T. T is made up of two subcomponents: the heating element (H) and the control module (C) which instructs H to start heating when the lever is pushed (thus, C uses H). Furthermore, H has a sub-subcomponent M, the timer.

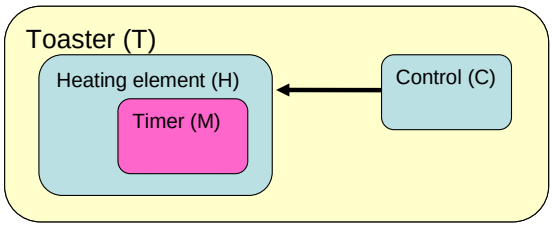

Fig. 5. A simple toaster T

Additionally, we know that a newer version of C, C', that can respond to changes in the 'browning level' made by the user can replace C. Similarly H' is more advanced and can vary the time of heat depending on the browning level. This information can be recorded using relational tables:

contains

container	slot	component
T	h	H
T	c	C
H	m	M

uses

user	service
C	H

replaces

superior	inferior
C'	C
H'	H

A sample document fragment structured according to the DN in Fig 3 for the toaster T is shown below:

T is a basic toaster that can detect when the user has pressed down the lever and start heating the toast for a set time. Once this time has passed, the heating is switched off and the lever returned to its original position. T is composed of two subcomponents: the heating element (H) and the control module (C). These will be described later in the document.

Similar fragments exist for all the components except C' and H'. However, this does not affect the documentation for T which will have the fragments in the order shown below:

fragment (T)
- fragment (C)
- fragment (H)
  - fragment(M)

The hierarchical structure is obtained by the *contains* relation and the sequence from *uses* relation.

Another build of T (T') is made but since component C is not available it is replaced by C'. Fragment(C') does not exist but since only the functionality of C will be realised in this build, the documentation can remain unchanged.

A third build is now made based on T' (T'') which has H' instead of H. However, this time the fragment C cannot be used to describe C' since the additional functionality can now be used because the heating element is able to deal with temperature (browning) changes. Hence, the documentation cannot be completed until fragment(C') and fragment(H') are ready.

With a data model as the one shown, it is possible to determine whether all the fragments are available to produce documentation for a given build. For a simple example like this toaster, this may seem trivial. However, for large systems with hundreds of components where the documentation is received from many sources, the searching of fragments and generation of documentation becomes correspondingly hard.

6. CONCLUSIONS AND FUTURE WORK

Previously, we have worked on architectures and software reuse [15, 16], and more recently on the structure of technical documentation [9, 17]. In this paper we have brought these two strands of research together.

As future work, we will investigate the relevance of this documentation model in different varieties of system evolution. So far we have only studied the case where the components in a system become progressively more advanced. Other changes include re-factoring the system functionality (logically related components can be grouped to form one, say) and the production of a family of products that are based on a common core [4]. Is it then the case that the author starts with a core document that is relevant to all the products and extends it to fit each product?

The data model in this paper has also been implemented so that we are able to carry out further experiments with real systems.

Just as software components are reused to increase productivity, document fragments should also be reused. However, traditional documentation does not lend itself very well to reuse [18]. In order to reuse a component, one has to understand its functionality and how it can be used in a specific context. We cater for this requirement by arguing that successful reuse can be achieved by defining a common structure, extracting common information and extending current documentation.

Producing high-quality documentation is a complex task. It should ideally parallel the development of the artefact [19] and can benefit from reflecting the structure of the system being described [20]. We have shown that there is a strong mapping between the system architecture and the way in which its documentation is composed and thought about. We believe this will improve the quality of both the architecture and the documentation, and increase the extent to which both can be reused.

REFERENCES

1. Clements, P., et al., *Documenting Software Architectures: Views and Beyond*. 2003: Pearson Education.
2. IEEE, *ANSI/IEEE Standard 1471-2000: Recommended practice for architectural description of software-intensive systems". (Available online at http://ieeexplore.ieee.org/servlet/opac?punumber=4278470; last accessed 5.3.2008).*
3. Garlan, D. and M. Shaw, *An Introduction to Software Architecture.* Advances in Software Engineering and Knowledge Engineering, 1993. **1**.
4. Müller, J.K., *The Building Block Method: Component-based Architectural Design for Large Software-intensive Product Families*. 2003, Universiteit van Amsterdam.
5. Kruchten, P.B., *The 4+1 View Model of architecture.* Software, IEEE, 1995. **12**(6): p. 42-50.
6. Meusel, M., K. Czarnecki, and W. Köpf, *A model for structuring user documentation of object-oriented frameworks using patterns and hypertext* in *ECOOP'97 — Object-Oriented Programming*. 1997, Springer Berlin / Heidelberg.
7. Gatzemeier, F., *Patterns, Schemata, and Types—Author Support Through Formalized Experience*. 2000. p. 27–40.
8. Mann, W. and S. Thompson, *Rhetorical Structure Theory: Toward a functional theory of text organisation.* Text, 1988. **8**(3): p. 243-281.
9. De-Silva, N. and P. Henderson. *Narrative-based writing for coherent technical documents*. in *ACM Special Interest Group on the Design of Communication.* 2007. El Paso, Texas, USA.
10. Holt, R., *Binary Relational Algebra Applied to Software Architecture*, in *CSRI Tech Report 345*. 1996, University of Toronto, Canada.
11. Kruchten, P., *What do software architects do?*, in *Available online at http://www.sei.cmu.edu/architecture/what_architects_do.pdf*. 2006.
12. Soni, D., R.L. Nord, and C. Hofmeister. *Software architecture in industrial applications*. in *17th International Conference on Software Engineering (ICSE)*. 1995. Seattle, USA: ACM Press New York, NY, USA.
13. Minto, B., *The pyramid principle*. 3rd ed. 2002, UK: Pearson Education Limited.
14. Gregor, S., J. Hutson, and C. Oresky. *Storyboard Process to Assist in Requirements Verification and Adaptation to Capabilities Inherent in COTS*. in *First international conference on COTS-Based Software Systems (ICCBSS 2002)*. 2002. Orlando, USA: Springer.
15. Henderson, P. *Laws for Dynamic Systems*. in *International Conference on Software Re-Use (ICSR 98)*. 1998. Canada: IEEE Computer Society.
16. Henderson, P. and J. Yang. *Reusable Web Services*. in *8th International Conference on Software Reuse (ICSR 2004)*. 2004. Spain: IEEE Computer Society.
17. De-Silva, N., *A narrative-based collaborative writing tool for constructing coherent technical documents*, in *School of Electronics and Computer Science*. 2007, University of Southampton: Southampton, UK.
18. Sametinger, J. *Reuse documentation and documentation reuse*. in *TOOLS 19: Technology of Object-Oriented Languages and Systems*. 1996. Paris, France: Prentice Hall.
19. Priestley, M. and M.H. Utt, *A unified process for software and documentation development*, in *Proceedings of the 18th annual ACM international conference on Computer documentation: technology & teamwork* 2000, IEEE Educational Activities Department: Cambridge, Massachusetts. p. 221-238.
20. Sametinger, J., *Object-oriented documentation.* ACM Journal of Computer Documentation, 1994. **18**(1): p. 3-14.