

# Aspect Traceability through Invertible Weaving

Victor Winter, Harvey Siy, Mansour Zand, Prasanna R. Aryal  
Department of Computer Science\*  
University of Nebraska at Omaha  
{vwinter,hsiy,mzand,paryal}@mail.unomaha.edu

## Abstract

*The dichotomy resulting from tangled and untangled representations may prove beneficial with respect to the manipulation and analysis of system requirements. This paper describes an invertible approach to weaving requirements documents that is based on  $\beta$ -conversion as defined in the  $\lambda$ -calculus.*

## 1 Overview

Aspect-orientation provides two distinct representations of an artifact: untangled and tangled. In its untangled form, an artifact is partitioned into aspects and non-aspects. In the tangled representation, aspects and non-aspects are woven together. Both representations serve useful purposes. For example, at the source code level the untangled representation can help programmers understand and manage the complexity of a software system. On the other hand, when in tangled form, the source code has a representation that reflects the computation to be performed during execution of the software.

In recent years, the abstractions made possible by aspects have been lifted to the design and requirements phases of software engineering. The dichotomy resulting from tangled and untangled representations may prove beneficial with respect to the manipulation and analysis of system requirements expressed at this level of abstraction. For example, a software requirements team may find it beneficial to perform an analysis on a set of requirements documents and to then manually trans-

late the requirements into an untangled form. The untangled form could then be the initial document presented to the design team. On the other hand, in tangled form the requirements may be more suitable for review by the customer. The rationale why this might be the case presumes that system requirements may originate from a number of sources. Requirements originating from one source (e.g., stakeholder) may overlap with requirements coming from another source. In this framework, sources need not be aware of each other and it is easy to envision a situation where a set of requirements is generated in which requirements between documents have numerous interrelationships and cross-cutting concerns.

To date, the majority of research in AOP has viewed weaving as unidirectional translation. Here, we consider an invertible weaving function. We propose a framework where an initial set of tangled requirements  $\mathcal{T}_0$  is manually translated into an untangled form  $\mathcal{U}_1$  during a preliminary analysis phase. In untangled form, a requirements document  $\mathcal{U}_1$  consists of two components: (1) an untangled requirements document, and (2) a set of aspects.

Given  $\mathcal{U}_1$ , a tangled representation  $\mathcal{T}_1$  can then be obtained automatically using a weaver. Formally, we will denote this translation by  $\mathcal{U}_1 \xrightarrow{\text{weave}} \mathcal{T}_1$ . Ideally,  $\xrightarrow{\text{weave}}$  will produce a set of requirements  $\mathcal{T}_1$  that is sufficiently similar to the original requirements  $\mathcal{T}_0$  that its correctness can be certified by an appropriate source, such as the customer or a domain expert. Given that such a link can be established one can also consider the development of an inverse-weaver  $\xleftarrow{\text{unweave}}$ . The development of  $\xleftarrow{\text{unweave}}$  establishes a formal, and bidirectional, link between the tangled and untangled forms of the requirements. Let  $\mathcal{M}$  denote a meaning function from requirements documents to their mean-

---

\*This work is supported by the UNO Center for Study and Development of Aspect-Oriented Systems

ing. The essential property that must hold in this framework is:

$$\begin{aligned} \mathcal{M}[\mathcal{T}_0] &\equiv \mathcal{M}[\mathcal{U}_1] \wedge \\ \mathcal{U}_1 &\xrightarrow{\text{weave}} \mathcal{T}_1 \wedge \mathcal{U}_1 \xleftarrow{\text{unweave}} \mathcal{T}_1 \end{aligned} \quad (1)$$

Formula 1 asserts that the manual translation of  $\mathcal{T}_0$  must produce a requirements document  $\mathcal{U}_1$  that is **semantically equal** to  $\mathcal{T}_0$ . Furthermore, the unweaving function must be a **syntactic inverse**. That is, when applied to  $\mathcal{T}_1$ , the function  $\xleftarrow{\text{unweave}}$  must produce a requirements document that is syntactically identical to  $\mathcal{U}_1$ .

Achieving this sets a stage where one can explore the impact of invertible weaving on the problem of requirements changes, traceability of requirements, and impact analysis. Specifically, we are interested in exploring the potential to provide some sort of automated support for changing customer requirements. Ideally, we would like to create a system where small changes in customer requirements can be automatically reconciled with the untangled representation of the requirements, at which point the untangled requirement set could be woven back into customer requirements. This would enable a better understanding of the propagation of requirements changes as well as their impact. We recognize the difficulty of the fully general version of this problem and propose to only support such manipulations and analysis under very restricted conditions.

This focus of this paper is to present some preliminary and foundational results underlying an approach to invertible weaving based on  $\beta$ -reduction.

## 2 (Un)Weaving Requirements Documents

One of the strengths (and weaknesses) of a natural language based requirements document is its semantic flexibility. It is this flexibility that makes requirements document so generally applicable across diverse problem domains. In this section, we impose several minor structural restrictions and extend requirements documents with additional information. Our goal is to produce a requirements document having formal components that can be subjected to weaving without significantly impacting its overall flexibility and adaptability.

We postulate that the requirements of a system be organized into a requirements document satisfying certain basic structural properties. Specifically, we expect a requirements document have a nested block structure and that its referencing conventions conform to static scoping rules. We also require that it be possible to assign labels to blocks. Thus, a requirements document has a hierarchical structure similar to that of a class diagram.

In the syntax considered in this paper, a requirements document has a structure similar to an outline in which the *section* is the mechanism used to group related requirements into a block. A *section* may be optionally annotated with label consisting of an identifier followed by a colon. Following the label is an optional parameter list comprised of identifiers referenced within the plain text of the requirement. Following the parameter list is the plain text of the requirement which is referred to as an *item*.

1. G1: Group 1
    - (a) S1: [x] this is the first labelled item referencing x
      - i. R1: [x] this is the second labelled item referencing x
      - ii. this is an unlabelled item
    - (b) S2: [y] this is the first labelled item referencing y
      - i. R1: this is the second labelled item referencing y
  2. G2: Group 2
    - (a) S1: [x] this is the third labelled item referencing x
      - i. R1: this is a labelled item having no explicitly identified references
      - ii. R2: [x] this is the fourth labelled item referencing x

**Figure 1. An abstract example of an annotated requirements document**

Within this type of requirements document, the *item* is the mechanism used to add a *requirement* to a section. At the present time, we restrict un/weaving to the level of granularity represented

by the *item*. That is, we do not attempt to un/weave unstructured plain text in an arbitrary fashion.

Under these structural assumptions, the *item* can be seen as forming a natural *join point*. A *pointcut* consists of a sequence of labels where individual labels in the sequence may contain one or more occurrences of the wildcard symbol `*`. Thus, a pointcut describes a set of items.

Figure 1 gives an abstract example of a requirements document in which sections (i.e., blocks) have been labelled G1:, S1:, R1:, S2:, R1:, G2:, S1:, R1:, R2:. The label sequence G1:S1:R1: uniquely designates the item whose plain text is: *this is the first labelled item referencing x*. Similarly, the label sequence G2:S1:R2 uniquely designates the item whose plain text is: *this is the fourth labelled item referencing x*. The pointcut G\*:S\*:R1: will match the following set of label sequences: { G1:S1:R1, G1:S2:R1:, G2:S1:R1 }.

## 2.1 A General Model for (Un)Weaving

We develop an aspect-oriented framework where weaving is modelled as a variation of the traditional method/function call mechanism. Recall that a traditional method call makes use of two distinct sources of information: (1) the method definition, and (2) the information provided by the method call (i.e., the values of the actual parameters). In the method call mechanism, the method to be called must be explicitly indicated at the point where the method is to be executed. Furthermore, the actual parameters that should be passed to the called method must also be explicitly indicated.

Code	Comment
method $f(y) = y + 5$	Method definition.
...	
... $f(y1)$ ...	Call point.

We now show, through a sequence of modifications, how weaving can be seen as a variation of the method call mechanism. In the first modification, we remove the actual name of the method at the given call point. In our running example, in the call  $f(y1)$  we remove the reference to  $f$  and abstract it to an unspecified method  $\mathcal{X}$  yielding the abstracted call  $\mathcal{X}(y1)$ .

Code	Comment
method $f(y) = y + 5$	Method definition.
...	
... $\mathcal{X}(y1)$ ...	Abstracted call point.

In the second modification, we add information to the definition of  $f$  specifying that  $\mathcal{X}$  can be resolved to  $f$ . This information, traditionally called a *pointcut*, establishes a formal connection between the call point  $\mathcal{X}(y1)$  and the definition of  $f$ . At this time we do not concern ourselves with the actual syntax used to express pointcuts.

Code	Pointcut
aspect $f(y) = y + 5$	$\mathcal{X}$ can be resolved to $f$
...	
... $\mathcal{X}(y1)$ ...	

In a purely aspect-oriented framework, the abstract call  $\mathcal{X}(y1)$  is even further simplified to  $y1$ . Thus, all information concerning the call is removed from the call point (which is now referred to as a *join point*) and is moved to the pointcut associated with the aspect definition.

## 2.2 Un/weaving via $\beta$ -Conversion

Because of the inability to automatically analyze the semantics of natural language-based requirements documents, we will retain a call structure having a form that essentially corresponds to the form  $\mathcal{X}(y1)$  described in Section 2.1. We also think of un/weaving in terms of  $\beta$ -conversion as it is defined in the  $\lambda$ -calculus [2].

In the  $\lambda$ -calculus,  $\beta$ -conversion is a rule establishing an equivalence between  $\lambda$ -expressions. When applied from left-to-right, this rule is referred to as  $\beta$ -reduction. When applied from right-to-left, this rule is referred to as  $\beta$ -abstraction. For example, given the definition  $f = \lambda y. y + 5$  and the pointcut  $\mathcal{X}$  *can be resolved to  $f$*  we define weaving in the following manner:

$$\begin{aligned} \mathcal{X} = f &\Rightarrow \mathcal{X}(y1) = f(y1) = (\lambda y. y + 5)(y1) \\ &(\lambda y. y + 5)(y1) \xrightarrow{\beta} y1 + 5 \end{aligned} \tag{2}$$

Furthermore, we can now also define unweaving as the inverse of weaving as follows:

$$(\lambda y. y + 5)(y1) \stackrel{\beta}{\leftarrow} y1 + 5 \quad (3)$$

It is important to note that in the proposed weaving framework the abstraction  $\stackrel{\beta}{\leftarrow}$  is guided by a finite set of aspect definitions and it is this finite set that makes  $\stackrel{\beta}{\leftarrow}$  feasible.

From a theoretical perspective, a major concern is whether unweaving is confluent. That is, whether the application of unweaving steps always eventually produce the same requirements document. From a practical perspective, a concern is how one would go about recognizing candidate terms for (un)weaving. Consideration of both of these issues lie beyond the scope of this paper.

### 2.3 Composing Requirements

At the level of abstraction considered, both requirements as well as the advice associated with aspects consist primarily of natural language fragments (e.g., sentences and phrases). The inability to formally analyze or structure (e.g., parse) such natural language-based entities places restrictions on how the advice in aspects can be composed with requirements during weaving. The lack of formality also places limitations on the extent to which automated support can be provided to determine that advice/requirement compositions result in tangled requirements that are semantically well formed.

At this time, rather than placing additional restrictions on requirements and advice, we leave the question of whether a requirements document or an advice-requirement composition is well formed open (e.g., subject to user certification). Furthermore, at present, we will only permit the concatenation operator  $\cdot$  to be used when composing an advice entity with a requirement entity. For example, if  $\mathcal{R}$  is a requirement and  $\mathcal{A}$  is an advice, then  $\mathcal{A} \cdot \mathcal{R}$  denotes a **before** composition and  $\mathcal{R} \cdot \mathcal{A}$  denotes an **after** composition.

We adopt a syntax for specifying an aspect as shown in Figure 2. Here the header portion of an aspect consists of the name of the aspect followed by a colon followed by a pointcut description followed by a list of formal parameters. Typically, a formal parameter will be a *wildcard identifier*. That is, an identifier containing one or more occurrences of the wildcard symbol  $*$  (e.g.,  $x*1$ ). Formal parameters represent identifiers that are referenced (literally) within an advice component. Formal para-

eters must be properly instantiated (i.e., matched with) a concrete identifier occurring within the requirement to which the advice is woven. This instantiation of wildcard identifiers within advice is accomplished via  $\beta$ -conversion as described in Section 2.2. We would like to point out that the primary reason for using wildcard identifiers within an advice entity is to facilitate the unweaving process.

An aspect body consists of an equation explicitly relating advice to requirements. The left-hand side of such an equation consists of a term containing uninstantiated variables (e.g.,  $\mathcal{R}$ ) that abstractly denote the requirement at the join point. The right-hand side of this equation consists of a term denoting the composition of an advice entity (e.g.,  $\mathcal{A}$ ) with the requirements term (e.g., the variable  $\mathcal{R}$ ).

In this context, weaving and unweaving are defined as a directional application of the equation that forms the body of an aspect. We refer to such a directional equation as a *rewrite rule*. Un/weaving can now be formally defined as the reduction of a requirements document with respect to a given set of rewrite rules. Un/weaving is complete when a requirements document reaches a *normal form*.

Body	Weaving	Unweaving
$\mathcal{R} = \mathcal{A} \cdot \mathcal{R}$	$\mathcal{R} \rightarrow \mathcal{A} \cdot \mathcal{R}$	$\mathcal{R} \leftarrow \mathcal{A} \cdot \mathcal{R}$
$\mathcal{R} = \mathcal{R} \cdot \mathcal{A}$	$\mathcal{R} \rightarrow \mathcal{R} \cdot \mathcal{A}$	$\mathcal{R} \leftarrow \mathcal{R} \cdot \mathcal{A}$

In a classical setting, a *normal form*, with respect to a set of rewrite rules, is defined as a term to which no rewrite rules can be applied. For terminating systems, such terms can be obtained through the exhaustive application of the rule set. In this case, confluence is necessary to assure that the order in which rules are applied during the course of exhaustive application is irrelevant. In our framework, we believe this definition of normal form is suitable for defining the unweaving of requirements documents.

When the advice equations are oriented in the weaving direction the resulting rule set will generally be nonterminating. Thus, obtaining a normal form through exhaustive application is infeasible. There are several ways the nontermination issue can be addressed and while interesting from a theoretical perspective most approaches are not particularly interesting in practice. In this paper we address the issue by controlling the application of rewrite rules so that an aspect can be woven to a particular join point at most once. This restriction assures that the weaving rule set is terminating.

Header	Formal Parameters	Body	Comment
name: pointcut	[args]	$\mathcal{R} = \mathcal{A} \cdot \mathcal{R}$	Before composition
name: pointcut	[args]	$\mathcal{R} = \mathcal{R} \cdot \mathcal{A}$	After composition

**Figure 2. An abstract view of the aspect structure.**

$\mathcal{T}_0$ : Initial Requirements Document in Tangled Form
<p>1. Functional requirements.</p> <p>(a) The switch must route 911 calls to a public safety answering point (PSAP) under all circumstances.</p> <p>2. Call processing:</p> <p>(a) Call processing must route the call according to internal tables to the switch of the called party.</p> <p>(b) The call processing subsystem must deny call completion if the caller has an overdue account, unless it is a 911 call in which case, it must be allowed to complete.</p> <p>3. Billing:</p> <p>(a) Every call must be metered, unless it is a 911 call.</p> <p>4. Caller ID blocking:</p> <p>(a) When the caller ID blocking feature is enabled, the switch must not forward the caller's phone number. However, the caller ID blocking feature is superseded for a 911 call.</p> <p>5. Capacity requirements:</p> <p>(a) The central office must support X number of calls per hour during peak hours. If there is a shortage of resources at peak hours, 911 calls take priority.</p> <p>(b) The central office must provide Y number of trunks in order to support the call load X. The central office must provide Z number of trunks dedicated for 911 service on top of Y.</p> <p>(c) The central office must provide Y number of physical trunks connected to the long distance tandem office. The central office must provide Z number of physical trunks connected to a 911 tandem switch.</p>

**Figure 3. An example of an initial requirements document in tangled form:  $\mathcal{T}_0$**

### 3 A Concrete Example

To date, a prototype invertible weaver has been implemented in the strategic programming language TL [8]. Strategic programming languages are languages in which transformations can be expressed through a collection of rewrite rules whose application is explicitly under user control [4]. Strategic programming languages typically provide a variety of combinators and traversals allowing sets of rewrite rules to be applied to terms in interesting ways.

The language TL is well-suited for the implementation of weavers and invertible weavers for several reasons. First, TL supports the definition

of higher-order rewrite rules, strategies, and traversals. Through the use of a higher-order rule (essentially a template) it becomes straightforward to dynamically convert aspects into strategies, which can then be applied to a requirements document to produce a document in un/woven form. Second, TL performs matching using a standard first-order matching algorithm that has been extended to permit matching on wildcard identifiers. Though a few systems have been developed that provide associative as well as associative-commutative matching/unification algorithms, to our knowledge TL is unique to the extent that it directly supports wildcard matching (a simple form of associative matching) on tokens (e.g., identifiers).

$\mathcal{T}_1$ : Tangled Requirements	$\mathcal{U}_1$ : Untangled Requirements
<p>1. Call_processing:</p> <p>(a) CP1: [<b>the_call</b>] { if <b>the_call</b> = 911 then call processing must route <b>the_call</b> to a PSAP, else }. Call processing must route <b>the_call</b> according to internal tables to the switch of the called party.</p> <p>(b) CP2: [<b>call</b>] { if <b>call</b> = 911 then call must be routed to a PSAP, else }. The <b>call</b> processing subsystem must deny completion of the <b>call</b> if the caller has an overdue account.</p>	<p>1. Call_processing:</p> <p>(a) CP1: [<b>the_call</b>] Call processing must route <b>the_call</b> according to internal tables to the switch of the called party.</p> <p>(b) CP2: [<b>call</b>] The <b>call</b> processing subsystem must deny completion of the <b>call</b> if the caller has an overdue account.</p>
<p>2. Billing:</p> <p>(a) B1: [<b>call</b>] { if <b>call</b> equal 911 then call must not be metered; otherwise }. Every <b>call</b> must be metered.</p>	<p>2. Billing:</p> <p>(a) B1: [<b>call</b>] Every <b>call</b> must be metered.</p>
<p>3. Caller_ID_blocking:</p> <p>(a) CIB1: [<b>callers_phone_number</b>] { if <b>call</b> = 911 then the <b>callers_phone_number</b> must be forwarded, else }. When the caller ID blocking feature is enabled, the switch must not forward the <b>callers_phone_number</b>.</p>	<p>3. Caller_ID_blocking:</p> <p>(a) CIB1: [<b>callers_phone_number</b>] When the caller ID blocking feature is enabled, the switch must not forward the <b>callers_phone_number</b>.</p>
<p>4. Capacity_requirements:</p> <p>(a) Cap1: The central office must support X number of calls per hour during peak hours. { If there is a shortage of resources at peak hours, 911 calls take priority }</p> <p>(b) Cap2: [<b>Y</b>] The central office must provide Y number of trunks in order to support the call load X. { In addition, the central office must provide Z number of trunks dedicated for 911 service on top of Y. }</p> <p>(c) Cap3: [<b>Y</b>] The central office must provide Y number of physical trunks connected to the long distance tandem office. { In addition, the central office must provide Z number of physical trunks connected to a 911 tandem switch on top of Y. }</p>	<p>4. Capacity_requirements:</p> <p>(a) Cap1: The central office must support X number of calls per hour during peak hours.</p> <p>(b) Cap2: [<b>Y</b>] The central office must provide Y number of trunks in order to support the call load X.</p> <p>(c) Cap3: [<b>Y</b>] The central office must provide Y number of physical trunks connected to the long distance tandem office.</p>

**Figure 4. Invertible Weaving:**  $\mathcal{T}_1 \xleftrightarrow{\beta} \mathcal{U}_1$

Using the prototype invertible weaver mentioned above, we take a look at a requirements document for processing phone calls, including 911 calls. Let us begin by considering the problem of providing 911 service. (911 is the official national emergency phone number in the United States and Canada.) Dialing 911 should connect a caller to a Public Safety Answering Point (PSAP) which contacts the necessary local emergency medical, fire, and law enforcement agencies. The phone number and lo-

cation of the caller is also passed on to the PSAP (this feature is also known as Enhanced 911). Due to the emergency nature of a 911 call, it will override many call processing functions and checks in telecommunications software. For example, 911 calls must be completed whether the calling numbers account is delinquent or not. Also, 911 calls are funded out of state telecommunications taxes and are not billed to the caller. Thus, specifying 911 functionality cuts across many requirements.

$\mathcal{U}_1$ : Aspects relating to 911 calls		
route_to_PSAP:	*CP*	[*call] $\mathcal{R} = \{ \text{if } *call = 911 \text{ then call must be routed to a PSAP, else } \} \cdot \mathcal{R}$
meter:	*B1	[call*] $\mathcal{R} = \{ \text{if } call* = 911 \text{ then call must not be metered; otherwise } \} \cdot \mathcal{R}$
forward:	*CIB1	[*phone_number] $\mathcal{R} = \{ \text{if } call = 911 \text{ then the } *phone\_number \text{ must be forwarded; otherwise } \} \cdot \mathcal{R}$
priority:	*Cap1	$\mathcal{R} = \mathcal{R} \cdot \{ \text{If there is a shortage of resources at peak hours, 911 calls take priority. } \}$
capacity1:	*Cap2	[Y*] $\mathcal{R} = \mathcal{R} \cdot \{ \text{In addition, the central office must provide Z number of trunks dedicated for 911 service on top of Y*. } \}$
capacity2:	*Cap3	[Y*] $\mathcal{R} = \mathcal{R} \cdot \{ \text{In addition, the central office must provide Z number of physical trunks connected to a 911 tandem switch on top of Y*. } \}$

Figure 5. Aspects Used for Invertible Weaving:  $\mathcal{T}_1 \xleftarrow{\beta} \mathcal{U}_1$

The original tangled form,  $\mathcal{T}_0$  of phone service requirements document is shown in Figure 3. In  $\mathcal{T}_0$ , concerns relating to 911 calls crosscut the document. An initial (manual) analysis phase results in the requirements document  $\mathcal{U}_1$  in which 911 concerns have been factored out. The forms  $\mathcal{U}_1$  and  $\mathcal{T}_1$  are shown in Figure 4 and can be generated automatically from one another using the aspects shown in 5.

## 4 Related Work

A number of works work have focused on (1) identification and separation of concerns, and (2) weaving such concerns into a requirements document. AORE with ARCaDe [6] is a tool that facilitates separation and composition of crosscutting and non-crosscutting requirements through a flexible composition support. ARCaDe relies on the flexibility of XML for requirements representation and on viewpoints for aspect identification. Composition of separated concerns can be achieved from the XML template composition rules and composition operator. However, ARCaDe doesn't support the unweaving of requirements once they are composed.

Theme/Doc [1] provides an easier approach that helps separation of concerns for requirements modelling. Action words are given to the Theme/Doc tool. Candidate aspects are identified after analysis of the requirements document for occurrences of action words or synonyms of them. The output of Theme/Doc tool is clipped action view, which shows crosscutting concerns separate from the non-crosscutting concerns. Composition rules is in-

ferred from the order in which themes are placed in clipped action view. A crosscutting concern is placed on top of the other concerns it crosscuts. Thus composition rule is to move bottom up while composing. Theme/Doc does not provide a view for composed requirements.

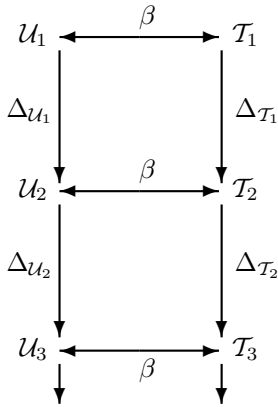
WMATRIX, a natural language processor, has been used to enable the identification of aspects [7]. NAPLES [5] is a product line approach that uses WMATRIX to facilitate requirements analysis. WMATRIX tags each word in the requirements document with its part-of-speech and semantic categories for identification of concerns. The tagged document is then processed with a mining tool (EA miner in [5]) to identify viewpoints and early aspects. The output is an AORE model showing viewpoints, early aspects and composition rules. Aspect Browser [3] has been proposed as a programming environment to identify and manage aspects for programmers. Possible aspects are identified by searching the entire project for redundancies in the code and in identifier names. Once candidate aspects and their associated occurrence counts are determined, a global view of interrelationships of different aspects is provided. Aspect Browser programming environment uses different highlighting colors to indicate presence of aspects in the code. It does not specify composition rules for crosscutting concerns.

The method we propose is different from the methods discussed above in the sense that, in addition to composing separated concerns, it can also unweave the composed requirements back to untangled form.

## 5 Future Work

We believe that invertible weaving makes possible a variety of requirements tracing mechanisms. An important capability in this setting is ability to incorporate, into the invertible weaving framework, changes to requirements that are made directly to the tangled form of the requirements document  $\mathcal{T}_1$  as well as changes made to its untangled form  $\mathcal{U}_1$ .

Consider a tangled requirements document  $\mathcal{T}_2$  that has been obtained by making a single conceptual change  $\Delta_{\mathcal{T}_1}$  to  $\mathcal{T}_1$ . We are presently exploring how and under what conditions  $\Delta_{\mathcal{T}_1}$  can be used as the basis for  $\mathcal{T}_2 \xrightarrow{\text{weave}} \mathcal{U}_2$ . We believe that confluence plays a major role in the ability to trace changes from a tangled form to its untangled representation. We also believe that such tracing capabilities can be extended by modelling changes to requirements documents incrementally in terms of sequences of smaller (atomic) changes; e.g.,  $\Delta_{\mathcal{T}} = \langle \Delta_{\mathcal{T}_1}, \Delta_{\mathcal{T}_2}, \dots, \Delta_{\mathcal{T}_n} \rangle$ . The resulting change structure is shown below.



## 6 Summary

In this paper, we described an invertible approach to weaving. The approach is based on modelling the body of aspect as an equation that relates an arbitrary requirement  $\mathcal{R}$  to a specific piece of advice  $\mathcal{A}$ . Un/weaving can then be defined as directional application of this equation, which in this directed form is referred to as a rewrite rule. Un/tangled forms of requirements documents can be defined in terms of normal forms with respect to a particular orientation of the equations. Confluence plays an important role with respect to normal forms. We conjecture that lack of confluence, in

many cases, is a sufficient indicator that concerns can be further separated. This is something that needs further exploration.

## References

- [1] E. Baniassad and S. Clarke. Finding aspects in requirements with theme/doc. In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design Workshop*, March 2004.
- [2] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1984.
- [3] W. G. Griswold, Y. Kato, and J. J. Yuan. Aspect browser: Tool support for managing dispersed aspects. In *OOPSLA '99 Workshop on Multi-Dimensional Separation of Concerns in Object-Oriented Systems (position paper)*, 1999.
- [4] R. Lämmel, E. Visser, and J. Visser. The Essence of Strategic Programming. Oct.15 2002.
- [5] N. Loughran, A. Sampaio, and A. Rashid. From requirements documents to feature models for aspect oriented product line implementation. In *Workshop on MDD in Product Lines (held with MODELS 2005)*, 2005.
- [6] A. Rashid, A. Moreira, and J. Araujo. Modularisation and composition of aspectual requirements. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 11–20, New York, NY, USA, 2003. ACM Press.
- [7] A. Sampaio, N. Loughran, A. Rashid, and P. Rayson. Mining aspects in requirements. In *Early Aspects (AOSD 2005)*, 2005.
- [8] V. Winter and M. Subramaniam. Dynamic Strategies, Transient Strategies, and the Distributed Data Problem. *Science of Computer Programming (Special Issue on Program Transformation)*, 52:165–212, 2004.