Documents
An Experimental Domain Description

Dines Bjørner
Fredsvej 11, DK-2840 Holte, Danmark
E–Mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~dibj

January 20, 2013: 09:46

Abstract

We speculate on the concept of documents. Documents are here considered abstract endurant entities: we stipulate, or assume, no concrete form for documents, whether as printed or electronic material, but we do impose some restrictions that may seem peculiar: copies of a document are distinct from this, have an own, unique identity, and occupy distinct locations. Documents are subject to operations: creation, editing, copying, reading and shredding. Each and every one of these operations leave an indelible mark on the document: the identity of the actor, the time and location of the operation; and the names and explicit argument values of the operations. The ensemble of all such document marks form a trace. If one can speak of a document property then it must be described with no concern for its implementability.

In this document (!) we shall describe such documents as hinted at above, informally, in the form of increasingly terser narratives, and formally, in the RAISE [2] Specification Language, RSL [1].

Contents

1 Documents 5

What Are Documents ? To a beginning we shall consider the concept of 'document' to be an algebra: an indefinite set of endurant entities, "the documents" (!), and a finite set of perdurant entities: the operations resulting in and/or performable on documents. Thus we are left to describe these entities and their operations. We shall start out by a narrative description 6 of operations resulting in or on documents. Through such a description we approach also a description of the endurant document entities by describing their properties and how they may change.

1.1 Document Operations 7

Documents are *cre*ated, *edi*ted, *rea*d, *cop*ied and *shr*edded.

- **type** DocOp = Cre | Edi | Rea | Cop | Shr

We shall later describe these operations in detail.

1.2 Document Space 8

Without committing ourselves too much we can consider the entirety of all documents to reside in a space, ω:Ω of uniquely identified such:

- **type** Ω = DI $\overrightarrow{m}$ D

We exemplify expressions involving $\omega, u\iota$ amd d.

- **dom** ω:

  The expression **dom** ω stands for the set of unique document identifiers of all documents in the space ω.

- $\omega(u\iota)$:

  Let $u\iota$ stand for a unique document identifier in **dom** ω then $\omega(u\iota)$ stands for a document. 9

- $\omega\cup[\,u\iota\mapsto d\,]$:

  Let $u\iota$ stand for a unique document identifier not in **dom** ω and d for a document, then $\omega\cup[\,u\iota\mapsto d\,]$ stands for a space, say ω' which is like ω except that a new document, with a fresh, unique document identifier ι, has been joined to ω.

- $\omega\dagger[\,u\iota\mapsto d'\,]$:

  Let $u\iota$ stand for a unique document identifier in **dom** ω and d' for a document, then $\omega\dagger[\,u\iota\mapsto d'\,]$ stands for a space, say ω' which is like ω except that $u\iota$ now maps to document d'.

- $\omega\backslash\{u\iota\}$:

  Let $u\iota$ stand for a unique document identifier in **dom** ω then $\omega\backslash\{u\iota\}$ stands for a document space that no longer records a document by name $u\iota$, that is,

  ⍟ **axiom** $\omega\backslash\{u\iota\}\cup[\,u\iota\mapsto\omega(u\iota)\,] = \omega$.

1.3 Document Properties 10

1.3.1 Unique Document Identifiers

Consider a set of documents: $\{d_i, d_j, \ldots, d_k\}$. Documents, d:D, are uniquely identified, $u\iota$:DI. Thus documents in $\{d_i, d_j, \ldots, d_k\}$ all have *the unique identifier property*, i.e., the unique identifiers: $\{u\iota_i, u\iota_j, \ldots, u\iota_k\}$, respectively. We can therefore associate a *unique identifier* function which from avery document observes its *unique identifier*;

- **value** *uid_D*: $D \to DI$

1.3.2 Actors 11

Actors, $\alpha : A$, perform these operations on documents at times, τ:T, and at locations, ℓ:L. Actors also have the *the unique identifier property*, i.e., have unique identifiers $\alpha\iota : AI$.

1.3.3 Time 12

Time, T, is here considered a simple, dense set of points with simple functions:

- **value** $<, \leq, =, \geq, >, \neq, =: T \times T \to$ **Bool**

1.3.4 Locations 13

Locations, ℓ:L, are considered, for simplicity, a set of points:

- **value** $\neq, =: L \times L \to$ **Bool**

1.3.5 Authorisation 14

We augment the set of the above (non-create) operations (edit, read, copy and shred) with an additional operation: *aut*horisation.

With documents we associate, for each (non-create) operation, sets of authorised persons, i.e., identification of actors, who are allowed to perform these operations.

- **type** Auth = DocOp $\xrightarrow[m]{}$ AI-**set**

$$\mathsf{aut{:}Auth} : \left[\begin{array}{lcl} \mathit{edi} & \mapsto & \{\alpha\iota_{i_1}, ..., \alpha\iota_{i_p}\}, \\ \mathit{rea} & \mapsto & \{\alpha\iota_{j_1}, ..., \alpha\iota_{j_q}\}, \\ \mathit{cop} & \mapsto & \{\alpha\iota_{k_1}, ..., \alpha\iota_{k_r}\}, \\ \mathit{aut} & \mapsto & \{\alpha\iota_{\ell_1}, ..., \alpha\iota_{\ell_s}\}, \\ \mathit{shr} & \mapsto & \{\alpha\iota_{m_1}, ..., \alpha\iota_{m_t}\} \end{array} \right],$$

where either of i_p, j_q, k_r, ℓ_s or m_t may be zero, i.e., the operation name maps into an empty set, i.e., no-one is authorised for the given operation.

1.3.6 Document Contents 15

A primary purpose of documents is to "carry" document contents: c:C. Documents are further undefined quantities: either c:$\mathbb{C}$ (also futher undefined), or `"nil"` designatimg an "empty" document:

type C = $\mathbb{C}$ | {$''$`nil`$''$|}

With document contents, c:C, we associate the following functions:

- **value** *is empty* $C \to$**Bool**
  value *forward* $C \to C$
  value *undo* $C \to C$

such that:

- **axiom** *forward* $\circ$ *undo* $= \lambda c{:}C \cdot c =$ *undo* $\circ$ *forward*

1.3.7 Document Traces 16

With every document we associate a trace of quintuplets: an actor identification, a time, a location, the name of the operation and arguments. The trace is a list of length one after the 17
initial create operation. Each operation and function appends one element to the list.

```
type
    Trace = Mark=list
    Mark = s_AI:AI  s_T:T  s_L:L  s_DocOp:DocOp  a_Args:Args
value
    attr_Trace: D → Trace
axiom
    ∀ uι:DI,op:DocOp\{|Cre|},ai:AI,t:T,l:L,ω:Ω •
        let ω′ = int_DocOp(op)(uι)(args)(ai,t,l)(ω) in
        attr_Trace(ω′(uι)) = ⟨(ai,t,l,op,args)⟩⌢attr_Trace(ω(uι)) end
```

1.4 The Operations — Rough Narratives 18

1.4.1 Create Document

When a document, d, is first created it receives (from an oracle) its unique identifier. In addition one can from d observe the identity. α, of the actor who created d, at which time, τ, d was created, and at which location, ℓ, d was created. Since it is an initial operation no authorisation is required. When creating a document the actor may refer to other documents as being a source for the created document – provided, of course, that the creator has at least reading right to these documents. There is not much else to observe from d as it is initially created. We thus say that an initially created document is [otherwise] empty: its document contents is "nil".

1.4.2 Edit Document 19

Editing a document, d, results in a document, d'; d no longer exists, but d' has the same unique identifier as d had. In addition one can from d observe the identity, α, of the actor who created d[1], at which time, τ, d was edited, and at which location, ℓ, d was edited. Let 20
c be the document contents of d. text with which d was edited. Editing d results in a new document contents, c'. Editing is here understood as a process, abstracted as a function, ϵ, from c to c', That function, ϵ always has an inverse, ϵ^{-1}, which when applied to d', $\epsilon^{-1}(d')$,

[1] We here assume that α is in the set of those actors who are edit-authorised

yields d, such that $\epsilon(\epsilon^{-1}(d'))=d'$. That is $\epsilon^{-1}\cdot\epsilon=\lambda x \bullet x$. Editing may refer to other documents, $\{d_a, d_b, \ldots, d_c\}$, in addition to any specified (initial) source documents, as a source of editing. You may think of ϵ and references to $\{d_a, d_b, \ldots, d_c\}$, as well as, of course, d, being arguments of the edit operation.

1.4.3 Read Document 21

Reading a document, d, does not change its unique identifier nor the operation-authorisations, but "reveals" the document contents, c. But, as other operations do, reading leaves a "footprint": who, *alpha*, did the reading[2], where, ℓ, was the reading done and at which time. τ. The operation-authorisations is unchanged. When reading a document, i.e., basically its contents, c, the reader may come across references to initial or editing source documents. Reading these is subject to the same constraints as listed just above – and are seen as separate, in a sense, "parallel" reading.

1.4.4 Copy Document 22

Copying a document, d, results in two documents, d' and d'', the 'copy', d'', which is a copy of the 'master', but with a new unique identifier; the 'copied master', d', with the same unique identifier as d had; and with d no longer existing. From d' and d'' one can observe the appropriate α, τ, and ℓ[3]. From d', the copied master, one can also observe the unique identifier of the copy, and From d'', the copy, one can also observe the unique identifier of the copied master, i.e., the master. Operation-authorisations for d' and d'' are unchanged (inherited) from d.

1.4.5 Shred Document 23

The shredding operation

1.4.6 Authorise Document 24

The authorisation operation

[2] We here assume that α is in the set of those actors who are read-authorised
[3] We here assume that α is in the set of those actors who are copy-authorised

2 Formalisation, I/II 25

2.1 States 26

2.1.1 Narrative

1. There a documents, $d : D$, unique [document] udentifiers, $u\iota : UI$, actor names $alpha : A$, times, τ, T, locations, $\ell : L$, and
2. document contents, $c : C$, which are either "nil" of some proper contents (in $\mathbb{C}$).
3. There are spaces of documents, $\omega : \Omega$.

2.1.2 Formalisation

```
1. type
1.   D, UI, AI, T, L
7.   C = ℂ | {|"nil"|}
3.   Ω = UI ~m~> D
```

2.2 Trace Wellformedness 27

2.2.1 Narrative

4. A trace is a sequence of zero or more actor, time and location triplets which is
5. which is wellformed if triplet times are ordered in increasing sequence.

2.2.2 Formalisation

```
4.  type
4.    AITL = s_AI:AI × s_T:T × s_L:L
4.    Trace′ = AITL*
4.    Trace = {| tr:Trace′ • wf_Trace(wf) |}
5.  value
5.    wf_Trace: Trace → Bool
5.    wf_Trace(tr) ≡
5.      ∀ i:Nat•{i,i+1}⊆dom tr⇒s_T(tr(tr[i]))<s_T(tr(tr[i+1]))
```

2.3 Operations: Syntax and Signatures 28

2.3.1 Narrative

6.
7. There are operations:

a create,
b edit,
c read and
d copy;
e authorise and
f shred.

2.3.2 Formalisation

```
7.  value
7a.   create: UI-set × Auth → AITL → Ω→ Ω × UI
7b.   edit: UI → (C → C) → AITL → Ω → Ω
7c.   read: UI → AITL → Ω → Ω
7d.   copy: UI → AITL → D × Ω → Ω
7e.   authorise: UI → Auth → AITL → Ω → Ω
7f.   shred: UI → AITL → Ω → Ω
```

2.4 Authorisation 29

2.4.1 Narrative

8. And there are authorisations. *auth*, which we model as maps from operation names to sets of actor names.
9. These are wellformed if all authorisations contain all operation names.

2.4.2 Formalisation

```
8.  type
8.    OpNam = edi|rea|cop|aut|shr
8.    Auth′ = OpNm m→ AI-set
9.    Auth = {| auth:Auth′ • wf_Auth(auth) |}
9.  value
9.    wf_Auth: Auth′ → Bool
9.    wf_Auth(auth) ≡ dom auth = {edi,rea,cop,aut,shr}
```

2.5 Operation Semantics 30

2.5.1 Create

We refer to section 1.4.1 on Page 5.

10. The create operation takes as arguments
    - a possibly empty set of unique document identifiers,
    - a possibly empty map of authorisations, and
    - a triplet of actor names, times and locations;

    applies
    - to a state, the space of documents;

and yields

- a possibly updated state and
- the unique document identifier

11. The create operation satisfies its pre-condition
    - if unique document identifiers of the argument are indeed those of documents in the state (space).

31

12. The create operation now yields a new state and a new, unique document identifier
    a. with a unique document identifier not of any document in the argument state space but in the resulting state
    b. which is like the argument state except that the unique document identifier is bound to a document, `d`,
    c. with a one element trace of the argument triplet of actor names, times and locations,
    d. with no contents.
13. It updates the state space to now contain also this document and yields its new, unique document identifier.

32

10. **value**
10. create: UI-**set** × Auth → AITL → Ω→ Ω × UI
10. create(uι**s**,auth,aitl)(ω) **as** (ω′,uι)
11. **pre**: known_uι**s**(uι**s**)(ω)
12a. **post**: ∼known_uι(uι)(ω) ∧ known_uι(uι)(ω′)
12a. ∧ **let** d = ω′(uι) **in** ω′ = ω ∪ [uι↦d]
12d. ∧ inspect_Auth(d) = auth
12c. ∧ inspect_Trace(d) = ⟨aitl⟩
12e. ∧ inspect_C(d) = ″`nil`″ **end**

2.5.2 Edit

33

We refer to section 1.4.2 on Page 5.

14. By an editing function we mean a pair, e_f, i_f, of functions from document contents to document contents such that i_f is an inverse of e_f and, vice versa, e_f is an inverse of i_f.
15. The edit operation takes as arguments
    - a unique document identifier, an editing function and
    - a triplet of actor names, times and locations.

16. The edit operation satisfies its pre-condition
    a. if the argument unique document identifier is that of a document in the argument state,
    b. if the actor is authorised for the edit operation on that document.
17. The edit operation yields a new state
    a. where all documents in the argument and the yielded states other than the one document identified by the unique document identifier are unchanged, and
    b. where the document identified by the unique document identifier in the argument state differes from document identified by the unique document identifier in the yielded state as follows:
        i. the trace is of the yielded document is that of the argument document with the triplet of actor name, time and location appended to its front;
        ii. the argument and the yielded authorisations are unchanged; and
        iii. the yielded contents is the forward editing of the argument contents.
    c. Undoing the yielded contents results in the argument contents.

```
14.  type
14.    EF = (C→C) × (C→C)
14.  axiom
14.    ∀ (fwd,und):EF • fwd°und = λx.x = und°fwd
15.  value
15.    edit: UI × EF → AITL → Ω → Ω
15.    edit(uι,(fwd,und),aitl:(ai,t,l))(ω) as ω′
16a.    pre: uι ∈ dom ω
16b.        ∧ edi ∈ dom(inspect_Auth(ω(uι)))(ai)
17a.    post: uι ∈ dom ω′ ∧ dom ω\{uι} = ω′\{uι}
17a.        ∧ let d = ω(uι), d′ = ω′(uι) in ω′ = ω † [uι↦d′]
17(b)i.     ∧ inspect_Trace(d′) = ⟨aitl⟩^inspect_Trace(d)
17(b)ii.    ∧ inspect_Auth(d′) = inspect_Auth(d)
17(b)iii.   ∧ inspect_C(d′) = fwd(inspect_C(d)) end
17c.    axiom:
17c.        und(inspect_C(ω′(uι))) = inspect_C(ω(uι))
```

3 Legal Documents

4 Bibliographical Notes

References

[1] C. W. George, P. Haff, K. Havelund, A. E. Haxthausen, R. Milne, C. B. Nielsen, S. Prehn, and K. R. Wagner. *The RAISE Specification Language*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1992.

[2] C. W. George, A. E. Haxthausen, S. Hughes, R. Milne, S. Prehn, and J. S. Pedersen. *The RAISE Development Method*. The BCS Practitioner Series. Prentice-Hall, Hemel Hampstead, England, 1995.