Doing and undoing in the framework of Web services

Marie-Claude Gaudel
LRI, Univ. de Paris-Sud & CNRS
(Programming and SE group)

Thanks to:
Michel Beaudouin-Lafon and Stéphane Conery, LRI (Man-Machine Interaction group)
and to Valérie Issarny (INRIA), Nicole Levy (Univ. de Versailles), and the other IST DSoS project members

Swansea, sept. 2005

Dependable composition of web services

- Component web services are shared with an unknown population of other users
- They are managed by independent entities
- Composite web services cannot:
  - lock a component web service for a long time
  - rely on roll back or backward recovery when something goes wrong (impossibility to successfully complete a composed operation, or crash of one component, …)
  - assume that some «pre-commit» feature is available in a component web service

Swansea, sept. 2005

This talk is not on testing 😊

  http://www.lri.fr/asspro/membres/gaudel.fr.html

Swansea, sept. 2005

The travel agent case study

Swansea, sept. 2005
Example of «!transactions!»

Aim and outline of the talk

- Composition of Web services raises some problems of atomicity
  - There is a need to undo some sub-operations when a “composed transaction” cannot succeed
- Similarities with “Undo” in collaborative work (for instance group editors)
- Presentation outline
  - Good old “undo” and “redo”
  - “undo” and “redo” in group editors
  - Transposition to composite Web services

What is undo?

- undo v
  1. vti to open, unfasten, untie, or unwrap something
  2. vt to cancel or reverse the effect of an action
  3. vt to cancel the effect of the last command or action done on a computer, restoring the material being worked on to its previous condition
  4. vt to bring somebody or something to ruin or disaster

Undo-Redo: the good old style

- “linear undo”:
  - Only the previous action is undoable
  - And then the previous one, etc
  - => stacks of possible “undo.s” and “redo.s”
- some actions are not undoable in some states
Bases of the implementation

- History buffer + “redo” stack

- Execution of a new command => addition to the history list

- Undoing the last action of the history list => moving it to the “redo” stack

- Redoing the top action of the Redo stack => moving it to the history list

- “Undo” and “Redo” do not appear in the history list (meta-actions)

- Undo can be performed via state recovery or reverse operations

Non Linear Undo

- A nice wish: any past action is undoable … “if that is meaningful” 😐

- Interlisp (1975)
  - “The user is explicitly warned that nonlinear undo might have unpredictable effect”

- “Selective undo” (Berlage 1994): the user is not able to select “undo” of a command when “It does not make sense” (7).
  - Collaborative graphic editor: GINA system

  - REDUCE system

Collaborative editing: a scenario

- Causal ordering relation (dependent operations)
  - $O_i$ generated at site i, $O_j$ generated at site j, $O_i$ $\rightarrow$ $O_j$ if
    - $i = j$ and $O_i$ generated before $O_j$
    - $i \neq j$ and the execution of $O_i$ at site i happened before the generation of $O_j$

- Independent operations
  - neither $O_i$ $\rightarrow$ $O_j$ nor $O_j$ $\rightarrow$ $O_i$

- Intention of an operation
  - The intention of an operation $O$ is the execution effect that can be achieved when applying $O$ to the state from which $O$ was generated

- Consistency
  - Convergence (same state after the same set of operations), and causality preservation (time stamping), and intention preservation
Back to the example

- $O_2 \parallel O_3$ and $O_2 \parallel O_3$ and $O_2 \parallel O_3$
- $O_2 \parallel O_3 \parallel O_3 \parallel O_3$

More about independent operations
- assume as initial state «abc»
- $O_1$ is Insert[2, X] ⇒ intention: «abc»
- $O_2$ is Insert[3, Y] ⇒ intention: «abcYc»
- Global intention: «abcYc»
- Site 0: $O_1$, $O_2$, «abcYc» ⇒ intention violation ⊗

Solution for intention preservation: Operational transformations
- Site 0: $O_1$, $O_2$, …, with $O'_2 = \text{Insert}[4, Y]$
- $O'_2$ is the result of the so-called Inclusion Transformation $IT(O_2, O_1)$

Transformations (Sun & al, 98, 2000)

- Inclusion Transformation of $O_4$ against $O_2$:
  - $IT(O_4, O_2)$ transforms $O_2$ into $O'_2$, in such a way that the impact of $O_4$ is included in the new parameters of $O'_2$
- Exclusion Transformation:
  - $ET(O_4, O_2)$ transforms $O_2$ in such a way that the impact of $O_4$ is excluded from the new parameters of $O'_2$
  - Example: $O_2 \parallel O_4$, but $IT(O_4, O_2)$ is not sufficient at site 0
  - $O_2$ and $O_4$ are generated at different states, because of the execution of $O_2$ at site 2 before $O_4$
  - At site 0, when arriving after $O_2, O_4$:
    - $O_4$ is transformed into $O'_4 = ET(O_4, O_2)$, because $O_2 \parallel O_4$
    - and then into $IT(O'_4, O_1)$, because $O_1 \parallel O_4$ and $O_2$

Some Technicalities

- To make a long story short… IT and ET must be defined for any couple of basic operations. Very often the result is the identity. IT and ET are application dependent.
- Context of an operation $O$:
  - $CT_{O_i}$, list of operations needed to bring the system from some initial state to the state on which $O$ is defined
- “context equivalent” relation
  - $O_x \equiv O_y$ if $CT_{O_x} = CT_{O_y}$
- “context preceding” relation
  - $O_x \rightarrow O_y$ if $CT_{O_x} = CT_{O_y} + CT_{O_y}$
- Reversibility requirement
  - if $O_x \rightarrow O_y$, then $O_y = ET(IT(O_x, O_y), O_y)$
  - if $O_x \leftarrow O_y$, then $O_y = ET(IT(O_x, O_y), O_y)$

This was «undoing!», what about «undoing!»?

- Let $\text{Undo}(O_i)$ generated or received at site $k$, with history buffer $HB_k = O_1 \ldots O_{j-1} \ldots O_k$
- Assumption: for any $O_i$ there is a reverse operation $\bar{O}_i$
  - Reminder: backward recovery cannot be assumed
  - $\otimes i < n : \text{execution of } O_i$
  - $\otimes i = n : \text{execution of } O'_i$, obtained by transformation of $O_i$
    - such that:
      - $O_1 \ldots O_{j-1} \ldots O'_i \ldots O_k$ has the same effect as $O_1 \ldots O_{j-1} \ldots O'_i \ldots O_k$
      - Roughly, the transformation of $O_i$ into $O'_i$ impacts the overall $O_{j-1} \ldots O_k$ and the transformation of $O_{j-1} \ldots O_k$ into $O'_{j-1} \ldots O'_k$
  - $\text{execution of } O'_i$ and then update of $HB_k$
Transformations for undoing

- Just the same ones as those for doing!
- Note that the strategy above is equivalent to doing $O_i$, with
  - $O_i \boxplus O_j$ for $1 \leq x \leq i$, and
  - $O_i \Box O_j$ for $i+1 \leq x \leq n$
- $O'_i = \text{LIT}(O_j, HB_{i+1,n})$
- where LIT is the generalisation of IT to lists of operations
- NB: the new $HB_i$ is not $O_1 \ldots O_i O_{i+1} \ldots O_n$, but $O_1 \ldots O'_i O_{i+1} \ldots O'_n$
- This allows an elegant and efficient treatment of $\text{Redo}(O_j)$
- See Sun & al. papers… not needed for web services

Transposition to Web services

Slightly simpler than in collaborative editing

- No problem of causal ordering: “doing” is straightforward (in first approximation…)
- “undoing” could follow the transformational model

Undo any operation at any time in Web Service?

- Requirements (transactional attitude of composable Web services, similar to Mikalsen & C°)
  - All undoable “operations” in the CWS are reversible in their WS
  - There is a unique history buffer for each WS, at least when “composite transactions” are performed
  - IT and ET are defined
- Back to the example: $HB = O_1 O^*_1 O^*_2 O_2 O^*_3$, and then $\text{Undo}(O_2)$…
  - Execution of IT($O_1, O^*_1$)
  - Modification of HB into $O_1 O^*_1 O^*_2 O^*_3$ ET($O^*_1, O_2$)
  - see next slide
Back to the example

- **HB = O₁ O₄⁻¹ O₃⁻¹ O₂ O₃⁻¹**
- **Undo(O₂)**
  - Execution of IT(O₂, O₄⁻¹)
  - Modification of HB into O₁ O₄⁻¹ O₃⁻¹ O₂ O₃⁻¹

Some research issues

- It gives a nice general model. How to instantiate it?
- The “Travel Agent” case study
  - O₂ is some flight reservation
  - O₄⁻¹ is another reservation for the same flight
    - which has been satisfied \(\rightarrow IT(O₂, O₄⁻¹) = O₂\)
    - which is in a waiting list \(\rightarrow MT(O₂, O₄⁻¹)\) satisfies the 2nd reservation
  - O₃⁻¹ is then a successful reservation
- Wanted: a definition of “has the same effect as” in
  - \(O₃⁻¹ O₄ O₃⁻¹\) has the same effect as \(O₃⁻¹ O₄ O₃⁻¹ \ldots O₃⁻¹\)
  - Observational equivalence of states...
    - to be extremely flexible... Actually, it may not be an equivalence
      - The waiting list was full: O₄⁻¹ will not be satisfied, even if O₂ is undone...
    - Strong relation with intention preservation

Some other issues

- It works when transactions are not too long, the possible operations are not too numerous, the transformations do not take too much time (lock of the site during the transformations...)
- Possibility of providing generic “wrappers” for making web services composable?
- If interested:
  - « Toward undoing in Composite Web Services », in : Architecting Dependable Systems III, pp. 59-68, LNCS 3549, 2005

Formalisation

- A. Imine, P. Molli, G. Oster, and M. Rusinowitch. « Proving correctness of transformation functions in real-time groupware ».