Formal Description for SaaS Undo

Hernán Merlino, Oscar Dieste, Patricia Pesado, and Ramón García-Martínez

PhD Program on Computer Sc. School of Computer Sc. National University of La Plata
Information Systems Research Group. Productive & Technologic Development Dept.
National University of Lanús
Instituto de Investigaciones en Informática LIDI. Facultad de Informática. UNLP - CIC
Empirical Software Eng. Group. School of Computer Sc. Madrid Polytechnic University
hmerlino@gmail.com, odieste@fi.upm.es,
ppesado@lidi.info.unlp.edu.ar, rgarcia@.unla.edu.ar

Abstract. This paper proposes a highly automated mechanism to build an undo facility into a new or existing system easily encapsulated into a service. The use of services strategy simplifies greatly the design of the undo process and encapsulates most of the functionalities required. We present a formal description when to use this service under alignments of software as a service.

Keywords: Undo, Services as a Service, and Usability component.

1 Introduction

Usability patterns were conceived with the aim of making usable software development simpler and more predictable [1]; in general usability requirements are included at an advanced stage of system development [2], when there is little time left and the key design decisions have already been taken.

The goal of this paper is to provide a formal description to detect availability to include software as a service (SaaS) [3] for undo usability patterns [4]. This provides the functionality necessary to undo actions taken by system users. This team decided to start with Undo pattern, because it is a common usability features in the literature [5].

Several authors have proposed alternatives to undo pattern, these alternatives focus on particular applications, notably document editors [6-7] although the underlying concepts are easily exportable to other domains. However, these proposals are defined at high level, without an implementation (or design) reusable in different types of systems. These proposals, therefore, do not solve the problem of introduction of usability features in software.

Undo has two alternatives of implementation in a system: (a) state operations. This option is present in systems where Undo functionality is a core for application, e.g. word editors, Applications without these functionality are not an option; (b) stateless operation. In these applications Undo functionality is only a plus for application, e.g. applications with forms to include and update data in a data base.

G. Herzwurm and T. Margaria (Eds.): ICSOB 2013, LNBIP 150, pp. 217–222, 2013.

Our proposal detects a second subset of cases (stateless operations) in a highly efficient manner. If formal description is aligned to application or section of application, the architect can use SaaS for Undo [3]. The importance of having an automated solution for those is that they are the most frequents operations that occur in information systems.

The use of services for building applications is a very efficient way to reduce complexity and development time, creating an Undo service is a valid alternative to be taken into account by software engineering. We have implemented the framework to use Software as a Service (SaaS). Beyond scope of this article, the research team is working on the realization of a SOA model [8].

This article is structured as follows. Section 2 describes the state of the art regarding the implementation of undo. Section 3 presents the undo infrastructure, whereas, finally, Section 4 briefly discusses and presents the main contributions of our work.

2 Background

Undo is a very widespread feature, and is prominent across the whole range of graphical or textual editors, like, for example, word processors, spreadsheets, graphic editors, etc. Not unnaturally a lot of the undo-related work to date has focused on one or other of the above applications. For example, [6] Baker and Storisteanu [9] have patented two methods for implementing undo in document editors within single-user environments.

There are specific solutions for group text editors that support undo functionality, such as in Sun [10] and Chen and Sun [11] and Yang [12]. The most likely reason for the boom of work on undo in the context of document editors is its relative simplicity.

The problems of undo in multi-user environments have also attracted significant attention. Abrams and Oppenheim [13] have proposed mechanisms for using undo in distributed environments, and Abowd and Dix [4] proposed a formal framework for this field. In distributed environments, the solution has to deal with the complexity of updates to shared data (basically, a history file of changes) [14].

Several papers have provided insight on the internal aspects of undo, such as Mancini [15], who attempted to describe the undo process features.

Another important aspect which has been worked out is the method of representation of the actions performed by the users in Washizaki and Fukazawa [16], where a dynamic structure of commands is presented that represents the history of commands implemented.

Patents, like the method for building an undo and redo process into a system, have been registered [17]. Interestingly, this paper presents the opposite of an undo process, namely redo, which does again what the undo previously reverted. Other authors address the complexities of undo/redo as well. Thus, for example, Nakajima and Wash [18] define a mechanism for managing a multi-level undo/redo system.

The biggest problem with the above works is that, again, they are hard to adopt in software development processes outside the document editor domain. The only

noteworthy exception to this is a design-level mechanism called Memento [19]. Uses of Services in the enterprise build architecture models that are directly dependent upon the business strategy [20]. Service oriented architecture has the following characteristics [21]: (a) services are self-contained and modular (b) services support interoperability, (c) services are loosely coupled, (d) services are location-transparent, (e) services are composite modules, comprised of components.

The solutions presented are optimized for particular cases and are difficult to apply to other domains; on the other hand, it is necessary to include a lot of code associated with Undo in the host application.

3 Theoretical Justification

This will be done in two steps; first we will describe how to undo operations that do not depend on its state, the procedure to undo these operations consists in reinjection input data at time t-1, second we prove that reinjection input always produces correct results.

3.1 Initial Description

The most commonly used option for developing an undo process is to save the states of objects that are liable to undergo an undo process before they are put through any operation; this is the command that changes the value of any of their attributes. This method has an evident advantage; the system can revert without having to enact a special-purpose process; it is only necessary to remove and replace the current in-memory objects with objects previously saved.

This approach is a simple mechanism for implementing the undo process, although it has some weaknesses. On one hand, saving all the objects generates quite a heavy system workload. On the other hand, developers need to create explicit commands for all operations systems. Finally, the system interfaces (mainly the user interface) have to be synchronized with the application objects to enact an undo process. This is by no means easy to do in monolithic systems, but, in modern distributed computer systems, where applications are composed of multiple components all running in parallel (for example, J2EE technology-based EJB), the complications increase exponentially.

There is a second option for implementing an undo process. This is to store the operations performed by the system instead of the changes made to the objects by these operations. In this case, the undo would execute the inverse operations in reverse order. However, this strategy is seldom used for two reasons. On one hand, except for a few exceptions like the above word processing or spreadsheet software, applications are seldom designed as a set of operations. On the other hand, some operations do not have a well-defined inverse (imagine calculating the square of a table cell; the inverse square could be both a positive and a negative number).

The approach that we propose is based on this last strategy, albeit with a more simplified complexity. The key is that, in any software system whatsoever, the only

commands processed that are relevant to the undo process are the ones that update the model data (for example, a data entry in a field of a form that updates an object attribute, the entry of a backspace character that deletes a letter of a document object, etc.). In most cases, such updates are idempotent, that is, the effects of the entry do not depend on the state history. This applies to the form in the above example (but not, for example, to the word processor). When the updates are idempotent, neither states of the objects in the model or executed operations has to be stored, and the list of system inputs is only required.

3.2 Formal Description

The following definitions and propositions are used to prove (in an algebraic way) that UNDO process (UNDO transformation) may be built under certain process (transformation) domain constrains.

Definition 1. Let $E = \{\varepsilon_j^i / \varepsilon_j$ is a data structure$\}$ be the set of all data structures.

Definition 2. Let ε_j^i be the instance i of data structure ε_j belonging to E.

Definition 3. Let $\varepsilon_j^C = \{ \varepsilon_j^i / \varepsilon_j^i$ is an instance i of the structure $\varepsilon_j\}$ be the set of all the possible instances of data structure ε_j.

Definition 4. Let $o_\tau^{\varepsilon j}$ be a transformation which verifies $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^C$ and $o_\tau^{\varepsilon j}(\varepsilon_j^i) = \varepsilon_j^{i+1}$.

Definition 5. Let ε_j^{Cr} be a constraint of ε_j^C defined as $\varepsilon_j^{Cr} = \{ \varepsilon_j^i / \varepsilon_j^i$ is an instance i of the data structure ε_j which verifies $o_\tau^{\varepsilon j}(\varepsilon_j^{i-1}) = \varepsilon_j^i\}$

Proposition 1. If $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ then $o_\tau^{\varepsilon j}$ is bijective. Proof: $o_\tau^{\varepsilon j}$ is injective by definition 4, $o_\tau^{\varepsilon j}$ is surjective by definition 5, and then $o_\tau^{\varepsilon j}$ is bijective for being injective and surjective. QED.

Proposition 2. If $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ then has inverse. Proof: Let $o_\tau^{\varepsilon j}$ be bijective by proposition 1, then by usual algebraic properties $o_\tau^{\varepsilon j}$ has inverse. QED.

Definition 6. Let o_τ be the set of all transformations $o_\tau^{\varepsilon j}$.

Definition 7. Let Φ be the operation of composition defined as usual composition of algebraic transformations.

Definition 8. Let Σ be the service defined by structure $< E^\Sigma , o_\tau^\Sigma , \Phi >$ where $E^\Sigma \subseteq E$ and $o_\tau^\Sigma \subseteq o_\tau$.

Definition 9. Let $X = o_\tau^{\varepsilon j1} \Phi o_\tau^{\varepsilon j2} \Phi \ldots \Phi o_\tau^{\varepsilon jn}$ be a composition of transformations which verifies $o_\tau^{\varepsilon ji} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ for all i:1...n. By algebraic construction $X : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$.

Proposition 3. The composition of transformations X has inverse and is bijective. Proof: Let be $X = o_\tau^{\varepsilon j1} \Phi o_\tau^{\varepsilon j2} \Phi \ldots \Phi o_\tau^{\varepsilon jn}$. For all i:1...n verifies $o_\tau^{\varepsilon ji}$ has inverse by proposition 2. Let $[o_\tau^{\varepsilon ji}]^{-1}$ be the inverse transformation of $o_\tau^{\varepsilon ji}$, by usual algebraic properties $[o_\tau^{\varepsilon ji}]^{-1}$ is bijective. Then it is

possible to compose a transformation $X^{-1} = [o_\tau^{\varepsilon jn}]^{-1} \Phi [o_\tau^{\varepsilon jn-1}]^{-1} \Phi ... \Phi [o_\tau^{\varepsilon j1}]^{-1}$. The transformation X^{-1} is bijective by being composition of bijective transformations. Then transformation $X^{-1} : \varepsilon_j^{Cr} \rightarrow \varepsilon_j^{C}$ exists and is the inverse of X. QED.

Definition 10. Let UNDO be the X^{-1} transformation of X.

3.3 Use Method

If the evaluated system is aligned with the formal description detailed above, architect could use SaaS described in [4]; in another way, probably the architect needs to use any of specific domain's implementations of Undo detailed in section 2 (Background).

4 Conclusions

In this paper we have proposed a formal description to detect a sub set of Undo functionality and an alternative to implement this usability functionality in a system. The most salient feature of this framework is the type of information it stores to be able to undo the user operations: input data instead of in-memory object states or commands executed by the system. This lessens the impact of building the framework into the target application a great deal.

Building an Undo Service has some significant advantages with respect to Undo models presented. First of all the simplicity of inclusion in a host application under construction or existing, can be seen in the proof of concept. Second, the independence of service in relation to the host application allows the same architectural model to provide answers to different applications in different domains. Construction of a service allows Undo to be a complex application, with the possibility of including analysis for process improvement, as described in the next paragraph it is possible to detect patterns of invocation of Undo in different applications.

Further work is going to bring: (a) creation of a pre-compiler, (b) automatic detection of fields to store (c) extension of the framework to other platforms.

Acknowledgements. The research reported in this paper has been partially funded by grants UNLa-SCyT-33A167 and UNLa-SCyT-33B112 of the National University of Lanus (Argentine) and by grants TIN2008-00555 and HD2008-00046 of the Spanish Ministry of Science and Innovation (Spain).

References

1. Ferre, X., Juristo, N., Moreno, A.: Framework for Integrating Usability Practices into the Software Process. Madrid Polit. University (2004)
2. Ferre, X., Juristo, N., Moreno, A., Sanchez, I.: A Software Architectural View of Usability Patterns. In: 2nd Workshop on Software and Usability Cross-Pollination (INTERACT 2003), Zurich, Switzerland (2003)

3. Merlino, H., Dieste, O., Pesado, P., García-Martínez, R.: Service Oriented Architecture for Undo Functionality. In: Proceedings 6th International Conference on Research and Practical Issues of Enterprise Information Systems (2012)
4. Merlino, H., Dieste, O., Pesado, H., García-Martínez, R.: Software as a Service: Undo. In: Proceedings 24th International Conference on Software Engineering and Knowledge Engineering (SEKE 2012), pp. 328–332 (2012) ISBN 978-1-891706-31-8
5. Abowd, G., Dix, A.: Giving UNDO attention. University of York (1991)
6. Qin, X., Sun, C.: Efficient Recovery algorithm in Real-Time and Fault-Tolerant Collaborative Editing Systems. School of computing and Information Technology Griffith University Australia (2001)
7. Bates, C., Ryan, M.: Method and system for UNDOing edits with selected portion of electronic documents. PN: 6.108.668 US (2000)
8. Merlino, H., Pesado, P., Dieste, O., García-Martínez, R.: Inclusion Process of UNDO/REDO Service in Host Applications. In: Software Engineering, Methods, Modeling and Teaching, Edited by Pontificia Universidad Católica de Peru. JIISIC 2012, Lima, Peru, vol. II (2011)
9. Baker, B., Storisteanu, A.: Text edits system with enhanced UNDO user interface. PN: 6.185.591 US (2001)
10. Sun, C.: Undo any operation at time in group editors. School of Computing and Information Technology, Griffith University Australia (2000)
11. Chen, D., Sun, C.: Undoing Any Operation in Collaborative Graphics Editing Systems. School of Computing and Information Technology, Griffith University Australia (2001)
12. Yang, J., Gu, N., Wu, X.: A Documento mark Based Method Supporting Group Undo. Department of Computing and Information Technology, Fudan University (2004)
13. Abrams, S., Oppenheim, D.: Method and apparatus for combining UNDO and redo contexts in a distributed access environment. PN: 6.192.378 US (2001)
14. Berlage, T., Genau, A.: From Undo to Multi-User Applications. German National Research Center for Computer Science (1993)
15. Mancini, R., Dix, A., Levialdi, S.: Reflections on UNDO. University of Rome (1996)
16. Washizaki, H., Fukazawa, Y.: Dynamic Hierarchical Undo Facility in a Fine-Grained Component Environment. Department of Information and Computer Science. Waswda University, Japan (2002)
17. Keane, P., Mitchell, K.: Method of and system for providing application programs with an UNDO/redo function. PN:5.481.710 US (1996)
18. Nakajima, S., Wash, B.: Multiple levels UNDO/redo mechanism. PN: 5.659.747 US (1997)
19. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison- Wesley (1994)
20. Binildas, C.A., Malhar, B., Vincenzo, C.: Service Oriented Architecture with Java. Packt Publishing, Birmingham – Mumbai (2008)
21. Endrei, M., Ang, J., Arsanjani, A., Chua, S., Comte, P., Krogdahl, P., Luo, L., Newling, T.: Patterns: Service-Oriented Architecture and Web Services. IBM, Redbooks (2004)