1 INTRODUCTION

Usability principles are a set of good software practices, which make an application interact according to features and user expectations (Ferre et al. 2003). Within these principles we can include the functionality of Undo/Redo, which allows a user undoing or redoing an action performed by him. The inclusion of this functionality in a new or existing system is not a trivial process; one of the reasons for this assertion is that inclusion is usually performed in an advanced phase of the system development lifecycle (Ferre et al. 2003), when there is little time for same conclusion and key decisions have been already taken and designed.

These usability principles have taken the form of usability patterns, which have been designed with the objective of software development as a simple and predictable process (Ferre, Juristo & Moreno 2004); these patterns can be defined as mechanisms used during system design to provide a set of specific software usability features (Ferre et al. 2003). Some usability patterns defined in the state of the art are: Feedback, Undo/Redo, Cancel, Form/Field Validation, Wizard, User Profile, and Help (Juristo et al. 2005). The main obstacle to implement such patterns is the absence of a framework that contains entire including process of software artifact "usability functionality" in a new or existing application—from now on host application—with emphasis on architectural aspects, design, and performance associated with usability patterns. This means, patterns have to be applied ad hoc in each system. This implies that the cost of system development process will increase as a result of increased workload caused by each design. In other way, the implementation of usability features in an advanced stage of the project also make certain usability features left out of the development in order to reduce development effort.

The objective of this Chapter is developing a framework with more common usability patterns. We initially selected the Undo/Redo pattern, which provides the functionality needed for undoing and redoing user actions in a system. Undo/Redo is a pattern commonly used in the state of the art (Abowd & Dix 1991); this justifies our pattern selection to start building the framework.

There are other technical reasons supporting the decision of starting with the Undo pattern: this pattern shares much of its infrastructure (design, code) with other patterns—e.g., Redo and Cancel in the most obvious cases—but also applied to seemingly unrelated patterns, like Feedback and Wizard.

Some authors have defined alternatives to Undo, focused on particular domains, especially in the area of text editors (Qin & Sun 2001; Bates & Ryan 2000). Although these concepts can be exported to other domains, such alternatives are defined at a higher level, without an implementation that demonstrates their ability to be reusable in different types of systems; therefore, these alternatives do not solve the problem in a broad sense.

This chapter presents a new approach to the Undo/Redo functionality: this efficiently addresses a subset of cases (stateless operations model), demonstrating the importance of having a semi-automated solution for the inclusion of the new or existing system functionality.

The framework has been implemented as a service (SaaS), with a process to be orderly included into a host application. The framework is based on the ideas of Spring (2011) and Hibernate (2011); finally, the team emphasizes on the fact that the host application must receive a reduced and easy modification to include the functionality.

This paper is organized as follows. In Section 2 we describe the state of the art regarding the implementation of Undo. In Section 3 we present the problem to be solved, while in Section 4 we present the proposed solution. Section 5 includes a proof of concept of the draft framework. Finally, in Section 6 we briefly present the main contributions of our work.

2 STATE OF THE ART

The Undo/Redo functionality is a feature widely used and very important for a range of applications such as: word processors, spreadsheets, graphic editors, etc.; you can refer to Bates and Ryan (2000) and Baker and Storişteanu (2011); they have patented methods for implementing the Un-
do/Redo functionality for document editors in single-user environments.

There are specific solutions for text editors that support sharing functionality of Undo/Redo, as in Sun (2000), Chen and Sun (2011), and Yang, Gu, and Wu (2004). The reason for the number of solutions proposed for text editors is its relative simplicity. Conceptually, an editor is a container object with certain properties (form, position, etc.). Therefore, Undo is relatively easy to implement, storing state of host system in time units, e.g., i, i + 1, ..., i + n, then when it performs Undo commands, the container runs in reverse i + n, i + n-1, i.

A derivation of solutions for text editors is an alternative implementation of Undo/Redo for email systems (Brown & Patterson 2003). These solutions are implemented within the text editor that e-mail systems have.


In distributed environments, the solution should handle the complexity of the changes of shared data; this is done by means of a change history file (Berlage, & Genau 1993).

Several studies have provided information on the internal aspects of Undo, such as Mancini, Dix and Leviaudi (1996), which describe the features of the Undo process; even so, Berlage (1994) proposes the construction of a method based on Undo commands in graphical environments, Burke (2007) has worked on the concept of infrastructure (Korenstein 2003) and gives guidelines for a model of selective Undo.

Another aspect which has been considered is a model representation of users actions performed in Washizaki & Fukazawa (2002) this is a dynamic structure of the commands executed in historical form.

Undo functionality by representing graphical models has been extensively developed by Berlage (1994) who distinguishes between linear Undo by using an archive and a non-linear one, which is represented by a tree graph. Some other branches can be opened according to user actions. Edwards and Mynatt (1998) also presents a graph structure similar to such one proposed by Berlage (1994), with tree branches representing a new set of actions taken by the user. Dix, Mancini & Leviaudi (1997) work on a cube graphics to represent the history of the actions taken by the user. Meanwhile Edwards et al. (2000) do a model of Undo actions in parallel threads. Milestoning and Rollback (reported by O’Brain & Shapiro 2004) have used a register which temporarily stores the actions; this model has been widely used for its simplicity.

All of these alternative representations of the Undo feature are valid, but is not a simple task to implement, and to create a new branch of action and the union of two existing branches is not a trivial task, as it should know all possible users’ paths, and consequently may be advisable to generate a time-ordered linear structure; this structure may be a queue, which is easy to deploy and manage.

Historically, some authors have used the "Command" pattern to represent Undo/Redo (Buschmann et al. 1996; Fayard & Shumidt 1997; Mesherer 1998); this serves to maintain a list of commands executed by the user, but it is not enough to create a framework easy to add to existing systems. As detailed below, using a service model with a process of inclusion can be more flexible, in order to integrate the functionality of Undo/Redo into an application. This allows a greater degree of complexity in the functionality of Undo/Redo, based on different configurations that provide the service, adapting to the requirements of the host application. Undo/Redo functionality has been also associated with exception handling mechanisms to reverse actions that fail (Shinnar et al. 2004); this model is only invoked after a failure.

Some other authors have worked on patents, as the method for the construction of a process of Undo/Redo in a system (Keane et al. 1996); curiously, such a paper presents the opposite of an undo process.


The biggest problem with the aforementioned projects is the difficulty to take the software development processes outside the domain of the text editor. The only notable exception is a pattern called Memento design level (Gamma et al. 1994). This model recovers an object to a previous state and provides an independent mechanism that can be easily integrated into a system; the main drawback is that this pattern is not easy to build on an existing system. In addition, Memento only restores an object to a previous state and other options that the Undo pattern should include are not considered.

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 the Undo functionality in host application.

3 PROBLEM

For software applications, Undo/Redo functionality can be a primary or is a desirable feature. If an application is included in the first type, core functionality, a designer includes functionality in the core of the application; these applications—e.g., word processors, spreadsheets, email managers, instant messaging managers, etc.—cannot be conceived without the existence of Undo/Redo and the user expects the existence of such functionality.

On the other hand, applications enrolled in the second category, desirable functionality, should include Undo/Redo functionality to be complete from the usability point of view; in this category we include management applications such as database systems, data loading user interface, and others. In the second category, system users may enter data and may
generate input errors, generating the need for correcting these errors; in this way the functionality is a desirable feature.

Focusing on this second category, in this Chapter we attempt to answer the following questions; (a) Is it possible to develop an infrastructure to include Undo/Redo functionality in a simple and quickly way inside an existing or new application? (b) Is it possible to define a process for detecting the features of Undo/Redo needed by an application?

4 PROPOSED SOLUTION

In this Section we detail several approaches to implement the Undo/Redo functionality, we compared them, and we state a position on each. Such approaches are: *Ad Hoc*, Patterns, Software as a Service (SaaS), Frameworks, and Processes.

4.1 *Ad Hoc*

Developer includes the Undo functionality when he/she is writing the application code. Neither code reuse nor practices learned from previous development are considered. One advantage of this alternative is that the application has all of the required features; the downside of is approach is that with every application the code needs to be rewritten, causing a higher rate of error, given that the experience of other applications is not transferred. Also, the functionality of Undo/Redo for every application always needs to be adapted.

4.2 *Patterns*

This alternative includes Undo/Redo functionality with a proven structure; all code is included in the host application. The use of patterns is a substantial advance over the *Ad Hoc* alternative, because experience has been systematized in a scheme that can be used in almost any application making the necessary adjustments. One disadvantage of this alternative is including all complexity of Undo/Redo functionality; this implies a considerable increase in the number of code lines, with the consequent tendency to add errors. This alternative is feasible in cases where Undo/Redo functionality is a core application element, e.g., a word processor.

4.3 *Software as a Service (SaaS)*

With this approach, the host application invokes Undo/Redo functionality through an external service. All functionality is encapsulated inside a service. This solution has the advantage of not increasing the code in the host application, thus, reducing the amount of coding errors. It also keeps complexity required for each application. One disadvantage of this solution is the method of inclusion inside the host application; the developer includes service, based in his/her own experience with the Undo/Redo functionality.

4.4 *Framework*

For framework definition, authors use a set of routines encapsulated inside an API (Application Program Interface), with a certain order of invocation. In this case, the strategy can be combined with patterns and SaaS.

4.5 *Process*

This strategy also encompasses the use of a framework, including a set of steps to detect the features of Undo/Redo necessary for a host application and a set of post implementation tasks. This alternative can be applied wherever we enlist our solution.

We selected a set of desirable features for the implementation of a solution in a system, and we compared them in Table 1.

Table 1. Comparison. (Scale= NC not meet the condition, CP partially meets, CT fully meets the condition)

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Repeatable</th>
<th>Scalable</th>
<th>Auditable</th>
<th>Perfectible</th>
<th>Transferable</th>
<th>Integrable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad Hoc</td>
<td>NC</td>
<td>NC</td>
<td>CP</td>
<td>CP</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>Patterns</td>
<td>CT</td>
<td>CP</td>
<td>CT</td>
<td>CT</td>
<td>CP</td>
<td></td>
</tr>
<tr>
<td>SaaS</td>
<td>CT</td>
<td>CT</td>
<td>NC</td>
<td>NC</td>
<td>CT</td>
<td>CP</td>
</tr>
<tr>
<td>Framework</td>
<td>CT</td>
<td>CT</td>
<td>CP</td>
<td>CP</td>
<td>CT</td>
<td>CT</td>
</tr>
<tr>
<td>Process</td>
<td>CT</td>
<td>CT</td>
<td>CT</td>
<td>CT</td>
<td>CT</td>
<td></td>
</tr>
</tbody>
</table>

As you can see, the implementation of a process is the most promising alternative for the Undo/Redo. Construction of the Undo/Redo functionality is hidden by the service. In this case each designer can include complexity in his application and the service will be reused by different applications, whereby the construction effort is justified.

The service has a set of screens for configuration and management, where data should be entered as in an application for resource administration, and authorized users may load the usability requirements for the implementation and use for future historical analysis.

A method for defining the data to be included in a possible service invocation is critical in order to realize successfully the inclusion of the service in the host application. The phases of the proposed process, input, and output are shown in Table 2 and the program flow can be seen in Figure 1.

5 STEPS, PROBLEMS AND POTENTIAL SOLUTIONS

This section presents methodological phases defining the inputs, outputs, and potential problems that may occur and the proposed solutions.

5.1 *Phase 1: Usability Analysis (E1-AU)*

*Input:* System Requirements (E1-AU-RS)

*Procedure:* Usability Requirements Detection (E1-AU-AU).

*Output:* Usability Requirements (E1-AU-RU).
Table 2. Phases of the proposed process (part 1 of 2).

<table>
<thead>
<tr>
<th>STEPS</th>
<th>NAME</th>
<th>INPUT</th>
<th>TASK</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Usability Analysis (E1-AU)</td>
<td>System Requirements (E1-AU-RS)</td>
<td>Document</td>
<td>Usability Requirements Detection (E1-AU-AU)</td>
<td>Usability Requirements (E1-AU-RU)</td>
</tr>
<tr>
<td></td>
<td>From general requirements, designer detect usability request</td>
<td></td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td>2. Logical Units Change Detection (E2-ULC)</td>
<td>Usability Requirements (E1-AU-RU)</td>
<td>Document</td>
<td>ULC Detection (E2-ULC-DE)</td>
<td>ULC Definition (E2-ULC-DI)</td>
</tr>
<tr>
<td></td>
<td>Define ULC relevant for Undo</td>
<td></td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td></td>
<td>Detect circumstances data cannot be turned back due to data consistency</td>
<td></td>
<td></td>
<td>PNR Millstones</td>
</tr>
<tr>
<td>4. Usability Test without Service (E4-PUO)</td>
<td>Host System</td>
<td>Software</td>
<td>Usability Tests (E4-PUO-U)</td>
<td>Usability Report (E4-PUO-AU)</td>
</tr>
<tr>
<td></td>
<td>Usability test on Host System</td>
<td></td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td>5. Stress Test without Service (E5-PS)</td>
<td>Host System</td>
<td>Software</td>
<td>Stress Test (E5-PS-EPS)</td>
<td>Stress Test Report (E5-PS-APS)</td>
</tr>
<tr>
<td></td>
<td>Stress test on Host System</td>
<td></td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td>6. Service Configuration (E6-CS)</td>
<td>ULC Definition + PNR Definition + Access Rights (E2-ULC-DI) (E3-PNR-DF)</td>
<td>Software + Document</td>
<td>Service Configuration (E6-CS-CS)</td>
<td>Service’s configured (E6-CS-SC)</td>
</tr>
<tr>
<td></td>
<td>Load data configuration</td>
<td></td>
<td></td>
<td>Software</td>
</tr>
<tr>
<td>7. Service’s Inclusion Method (E7-MIS)</td>
<td>Usability Requirements (E1-AU-RU)</td>
<td>Document</td>
<td>Inclusion Assessment Methods (E7-MIS-EMI)</td>
<td>Inclusion Method (E7-MIS-MI)</td>
</tr>
<tr>
<td></td>
<td>A partir de las ULC detectadas se analiza el mejor método para incluir el servicio en el sistema anfitrión</td>
<td></td>
<td></td>
<td>Document</td>
</tr>
</tbody>
</table>

Definición del mejor método para inyectar el servicio en el sistema.
### Table 2. Phases of the proposed process (part 2 of 2).

<table>
<thead>
<tr>
<th>STEPS</th>
<th>INPUT</th>
<th>TASK</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>8. Service Inclusion (E8-IS)</td>
<td>Host System + Inclusion Method (E7-MIS-MI)</td>
<td>Coding a Program (E8-IS-PRF)</td>
<td>Host System + Service's Invocation (E8-IS-IS)</td>
</tr>
<tr>
<td></td>
<td>Software + Document</td>
<td></td>
<td>Software</td>
</tr>
<tr>
<td></td>
<td>Include service in host system.</td>
<td></td>
<td>Service's included</td>
</tr>
<tr>
<td>9. Usability Test (E9-PU)</td>
<td>Host System + Service's Invocation + Usability Acceptance (E4-PUO) (E8-IS-IS)</td>
<td>Usability Test (E9-PU-U)</td>
<td>Usability Acceptance (E9-PU-AU)</td>
</tr>
<tr>
<td></td>
<td>Software</td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td></td>
<td>Usability test with service inclusion</td>
<td></td>
<td>Usability test report with service</td>
</tr>
<tr>
<td>10. Stress Test (E10-PS)</td>
<td>Host System + Service's Invocation + Stress Test Report (E5-PS-APS) (E8-IS)</td>
<td>Stress Test (E10-PS-EPS)</td>
<td>Stress Test Acceptance (E10-PS-APS)</td>
</tr>
<tr>
<td></td>
<td>Software</td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td></td>
<td>Stress test with service inclusion</td>
<td></td>
<td>Stress test report with service</td>
</tr>
<tr>
<td></td>
<td>Document + Document</td>
<td></td>
<td>Document</td>
</tr>
<tr>
<td></td>
<td>Last evaluation</td>
<td></td>
<td>General Acceptance</td>
</tr>
</tbody>
</table>

**Detail:** From general requirements, the designer detects usability requests related to Undo/Redo functionality. This process may require additional sessions with user to collect information.

**Risk:** User may don’t know all usability requirements related to the Undo/Redo functionality.

**Solution:** The designer should read system functional requirements, and detect usability requirements related to Undo/Redo. If requirements cannot be detected, conduct meetings with the user.

### 5.2 Phase 2: Logical Units Change Detection (E2-ULC)

**Input:** Usability Requirements (E1-AU-RU)

**Procedure:** ULC Detection (E2-ULC-DE)

**Output:** ULC Definition (E2-ULC-DI).

**Logical Unit Definition Change (ULC):** This concept refers to a ULC data set that should be treated as a unit for coherence of system information, e.g. telephone field is composed of the same country code, area code and telephone number. These data can be entered in separate fields, but for purpose of Undo/Redo process, this should work as a unit, if it’s necessary return to previous state service returns information about all related fields.

**Detail:** Define ULC relevant for Undo, and detect data sets to be labeled as ULC.

**Risk:** Designers can make mistakes in ULC process detection.

**Solution:** Perform an iterative requirement process.

### 5.3 Phase 3: Points No-Return detection points Detection (PNR) (E3-PNR)

**Input:** ULC Design + RS (E2-ULC-DI) (E1-AU-RU)

**Procedure:** PNR Detection (E3-PNR-DE)

**Output:** PNR Definition (E3-PNR-DF)

**Definition of Don’t Back Points:** The PNR are system stage with particular domain aspects or practical limitations, system cannot go back, these are usually in multiuser environments where changing done by one user invalid changes done by other.

**Detail:** Detect circumstances where data cannot be turned back due to data consistency.
Figure 1. Proposed program flow.
5.4 Phase 4: Usability Test without Service (E4-PUO)
Input: Host System
Procedure: Usability Tests (E4-PUO-U)
Output: Usability Report (E4-PUO-AU)
Detail: The aim of the usability test is to identify complexity No-Return detection points in the user interface, recommended method CPM-GOMS (Carrol 2003).
Risk: Unsatisfactory evaluation.
Solution: This has no effect on methodology strategy.

5.5 Phase 5: Stress Test without Service (E5-PS)
Input: Host System
Procedure: Stress Test Execution (E5-PS-EPS)
Output: Stress Test Report (E5-PS-APS)
Detail: Purpose stress test is to evaluate the system’s response with simultaneous users accessing it.
Risk: Unsatisfactory evaluation.
Solution: This has no effect on methodology strategy.

5.6 Phase 6: Service Configuration (E6-CS)
Procedure: Service Configuration (E6-CS-CS)
Output: Service’s configured (E6-CS-SC)
Detail: Load service configuration.

5.7 Phase 7: Service’s Inclusion Method (E7-MIS)
Input: Usability Requirements (E1-AU-RU)
Procedure: Inclusion Assessment Methods (E7-MIS-EMI)
Output: Inclusion Method (E7-MIS-MI)
Detail: Select the best way to include the service. This phase can be performed in parallel with E6-CS.

5.8 Phase 8: Service Inclusion (E8-IS)
Input: Host System + Inclusion Method (E7-MIS-MI)
Procedure: Coding a Program (E8-IS-PRF)
Output: Host System + Service’s Invocation (E8-IS-IS)
Detail: Include the service in host system.

5.9 Phase 9: Usability Test (E9-PU)
Input: Host System + Service’s Invocation + Usability Acceptance (E4-PUO-AU) (E8-IS-IS)
Procedure: Usability Test (E9-PU-U)
Output: Usability Acceptance (E9-PU-AU)
Detail: Evaluate user interface again with the implemented service. Same condition used in E4-PUO.
Risk: Worse result than E4-PUO-AU.
Solution: Review inclusion method.

5.10 Phase 10: Stress Test with Service (E10-PS)
Input: Host System + Service’s Invocation + Stress Test Report (E5-PS-APS) (E8-MS-IS)
Procedure: Stress Test (E10-PS-EPS)
Output: Stress Test Acceptance (E10-PS-APS)
Detail: Run stress test in host system with service.
Risk: Worse result than E5-PS-APS.
Solution: Review inclusion method.

5.11 Phase 11: Evaluación (E11-EV)
Procedure: Evaluation (E11-EV-R)
Output: Service Inclusion Acceptance (E11-EV-AIS)
Detail: Last evaluation and report generation.

6 CONCLUSIONS

In this Chapter we proposed a micro framework for including the Undo/Redo functionality and a method to include in a software application that Undo/Redo is to disable functionality, but not core functionality, such as a word processing. The most outstanding feature of this micro framework is type information stored in order to undo the operations done by the users, instead of memory object states or commands executed by the system.

Undo/Redo service has some significant advantages over other models presented; first, the simple inclusion in existing software, the effort compared to other models is significantly lower. Second, the independence provided by the service in relation to the host application, the service can create sophisticated models for each domain with Undo complexity encapsulated in a service. Future lines of research are being contemplated: (a) compiler creation, (b) automatic detection of fields susceptible of Undo process, (c) automatic detection of fields susceptible of Redo process.

AKNOWLEDGEMENT

The research reported in this chapter has been partially granted by Research Projects 33B066 and 33A105, Department of Productive and Technological Development, National University of Lanus.
REFERENCES


Berlage, T. & Genau, A. 1993. From Undo to Multi-User Applications. GNRCICS.


Burke, S. 2007. UNDO infrastructure. PN: 7.207.034 US.


Gamma, E., R. Helm, R. Johnson, & J. Vlissides. 1994. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.


Juristo, N; Moreno, A; Sanchez-Segura, M & Davis, A. 2005. Gathering Usability Information through Elicitation Patterns. GrISE Madrid Politechnic University.

Keane, P. & Mitchell, K. 1996. Method of and system for providing application programs with an UNDO/redo function. PN:5.481.710 US.


Li, C. 2006. UNDO/redo algorithm for a computer program. PN: 7.003.695 US.


Nakajima, S. & Wash, B. 1997. Multiple level UNDO/redo mechanism. PN: 5.659.747 US.


Shinnar, A; Tarditi, D; Plesko, M. & Steensgaard, B. 2004. Integrating support for undo with exception handling. Microsoft Research.


Sun, C. 2000. Undo any operation at time in group editors. School of Computing and Information Technology, Griffith University Australia.

