Improving software usability through architectural patterns

Natalia Juristo
School of Computing - Universidad Politécnica de Madrid, Spain

Marta Lopez
School of Computing - Universidad Complutense de Madrid, Spain

Ana M. Moreno
School of Computing - Universidad Politécnica de Madrid, Spain

M. Isabel Sánchez
School of Computing - Universidad Carlos III de Madrid, Spain

Abstract

This paper presents an approach for improving final software system usability by designing for usability, in particular by addressing usability issues in the software architecture. This approach differs from the traditional idea of measuring and improving usability once the system is complete. The work presented in this paper is part of the research conducted within the European Union - IST STATUS related to the development of techniques and procedures for supporting a forward-engineering approach to improve usability in software systems at the architectural level. In particular, we present the ongoing research about usability improvement by including architectural patterns that provide solutions for specific usability mechanisms.

1. Introduction

One reason why software architecture research is attracting growing interest is the direct relationship between architectural decisions and the fulfilment of certain quality requirements [1]. The goal is to assess software architecture for specific quality attributes and make decisions that improve these attributes. In short, a software architecture needs to be explicitly designed to satisfy specific quality attributes.

Moreover, usability is considered as just another quality attribute [2] and, therefore, we should also be able to design software architectures for usability as we do for other quality attributes.

The work presented in this paper is part of the insights and techniques developed in the STATUS project (SofTware Architectures That support USability)[1]. The goal of this project is to develop techniques and procedures to support a forward-engineering perspective to usability in software architectures, as opposed to the conventional backward-engineering alternative of measuring usability on a finished system and improving it once the system is practically complete.

In this paper, we will focus on presenting and discussing the ongoing STATUS research about architectural-level usability improvements.

For this purpose, section 2 shows the approach taken to decompose usability into levels of abstraction that are progressively closer to software architecture. These progressive levels are represented by the concepts of usability attributes, usability properties and usability patterns.

Then, section 3 shows how to incorporate the usability characteristics represented by the usability patterns into a generic software architecture. For this purpose, we will use the concept of architectural pattern, which specifies, in terms of components and their interrelationships, definite solutions for incorporating aspects that will improve final system usability into an architectural design.

Finally, section 4 presents future work to be done to complete and validate the approach taken in this paper.

2. Usability Decomposition: Attributes, Properties and Patterns

Software systems usability is usually evaluated on the finished system trying to assign values to the classical *usability attributes* [3] [4] [5]

- Learnability – how quickly and easily users can begin to do productive work with a system that is new to them, combined with the ease of remembering the way a system must be operated.
- Efficiency of use – the number of tasks per unit time that the user can perform using the system.

[1] STATUS project: EU funded project IST–2001–32298.

- Reliability – sometimes called "reliability in use", this refers to the error rate in using the system and the time it takes to recover from errors.
- Satisfaction – the subjective opinions that users form in using the system.

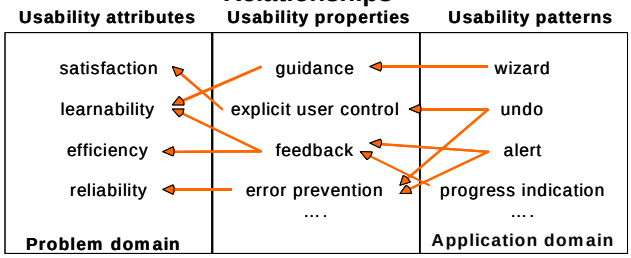
However, the level of these usability attributes is too high for us to be able to examine what mechanisms should be applied to a software architecture to improve these attributes. Therefore, the philosophy followed in STATUS was to decompose these attributes into two intermediate levels of concepts closer to the software solution: usability properties and usability patterns.

The first level involves relating the above-mentioned usability attributes to specific *usability properties* that determine the usability characteristics to be improved in a system. Usability properties can also be seen as the requirements of a software system for it to be usable (for example, provide feedback to the user, provide explicit user control, provide guidance to the user, etc). The second level was envisaged to identify specific mechanisms that might be incorporated into a software architecture to improve the usability of the final system. These mechanisms have been called *usability patterns* and they address some need specified by a usability property. Note that usability patterns do not provide any specific software solution to be incorporated into a software architecture, they just suggest some abstract mechanism that might be used to improve usability (for example, undos, alerts, command aggregations, wizards, etc.).

The procedure followed to identify the relationship between usability attributes, properties and patterns is detailed in Usability Attributes Affected by Software Architecture [6]. We took a top-down approach from usability attributes (identified in the literature), through usability properties (derived from heuristics and guidelines given in the literature to developers for improving usability), to finally identify usability patterns. Accordingly, usability patterns are the final links in the chain, and they provide examples of how to achieve some usability requirements. Nevertheless, they are not the central axis of our approach, which provides the users of the research results with a procedure for developing new usability patterns according to the context of the applications to which the results are applied.

A subset of the above-mentioned relationship is outlined in Table 1. It shows how usability properties relate patterns to usability attributes in a qualitative sense (an arrow indicates that a property positively affects an attribute, that is, improves that attribute). For example, the "wizard" pattern improves learnability: the wizard pattern uses the concept of "guidance" to take the user through a complex task one step at a time; "guidance" improves the learnability usability attribute. Usability patterns may address one or more of the usability properties and usability properties may improve one or more usability attributes.

Table 1. Attribute, Property & Pattern Relationships



The concept of usability pattern has already been used in the literature. This concept can be generally defined as "a description of solutions that improve usability attributes" [7]. The usability aspects dealt with by these patterns refer basically to user interfaces, which is why these patterns are also called user interface patterns. [8] or interaction design patterns [9]. As indicated by authors like Welie and Troetteberg [10], although several pattern collections exist, an accepted set of such patterns has not emerged. There appears to be a lack of consensus about the format and focus of user interface patterns.

Possible examples of some user interface patterns are:

- Feedback
- Wizard
- Provide the user with all information needed in the same window
- Mark required fields when filling a form
- You are here
- Grid Layout.

The differences between the usability patterns proposed in our work and the classic usability or interface patterns existing in the literature lie basically in that the classic patterns of usability are based on the improvement of the application interface, which means that these patterns are implemented mainly during the interface design phase and generally affect low-level components like pseudo-code. On the other hand, the usability patterns in our work relate the mechanisms to be considered in a software architecture, addressing usability aspects in the early stages of the development process. For example, the solution proposed by Welie [10] for the feedback pattern is based on "provide a valid indication of progress. Progress is typically the time remaining until completion, the number of units processed or the percentage of work done. Progress can be shown using a widget such as a progress bar. The progress bar must have a label stating the relative progress or the unit in which is measured". Whereas, as we will see later, we consider a progress indication pattern and provide a solution based on the components to be added to a software architecture and the

relationships among these components in order to provide this mechanism.

The second column in Table 2 shows the list of usability patterns that we propose. The first column of the table shows the usability properties related to each pattern.

Table 2. List of usability patterns

Usability Property	Usability Patterns
NATURAL MAPPING	
CONSISTENCY (functional, interface, evolutionary)	
ACCESSIBILITY (internationalisation)	Different languages
CONSISTENCY, ACCSESIBILITY (multichannel, disabilities)	Different access methods
FEEDBACK	Alert
ERROR MANAGEMENT, FEEDBACK	Status indication
EXPLICIT USER CONTROL, ADAPTABILITY (user expertise)	Shortcuts (key and tasks)
ERROR MANAGEMENT (error prevention)	Form/field validation
ERROR MANAGEMENT (error correction),	Undo
GUIDANCE, ERROR MANAGEMENT	Context-sensitive help
GUIDANCE, ERROR MANAGEMENT	Wizard
GUIDANCE, ERROR MANAGEMENT	Standard help
GUIDANCE, ERROR MANAGEMENT	Tour
MINIMISE COGNITIVE LOAD, ADAPTABILITY, ERROR MANAGEMENT (error prevention)	Workflow model
ERROR MANAGEMENT (error correction)	History l ogging
GUIDANCE, ERROR MANAGEMENT (error prevention)	Provision of views
ADAPTABILITY (user preferences)	User profile
ERROR MANAGEMENT, EXPLICIT USER CONTROL	Cancel
EXPLICIT USER CONTROL	Multi-tasking
MINIMISE COGNITIVE LOAD ERROR MANAGEMENT (error prevention)	Commands aggregation
EXPLICIT USER CONTROL	Action f or multiple objects
MINIMISE COGNITIVE LOAD, ERROR MANAGEMENT (error prevention)	Reuse information

It should be noted that the properties of Natural Mapping and Consistency cannot be arranged around specific usability patterns. The reason is that these properties require the performance of different tasks and activities throughout the entire development process rather than the application of particular solutions at the architectural level. For example, the provision of natural mapping between the user tasks and the tasks to be implemented in the system calls for software requirements to be elicited during the analysis process bearing in mind this objective and they must be designed according to these requirements. The same goes for consistency, which involves different activities throughout the lengthy development process of the original system or new versions.

At this point, we should refer to the work of Bass, John and Kates [11], who use the concept of usability scenario, where "a scenario describes an interaction that some stakeholder (e.g. user, developer, system administrator) has with the system under consideration from a usability viewpoint". These scenarios are related to some properties and usability patterns considered in our approach. Table 3 shows a comparison of their and our approaches, through the relationships between our patterns and their scenarios. These relationships are:

- Content: achieving a particular usability pattern implies achieving a particular scenario. For example, properly provide the *Provision of views* mechanism implies "Make views accessible".
- Instantiation: a usability pattern is a special case of a scenario. For example, *Standard Help* is a case of "Help".
- Similarity: a pattern and a scenario are considered similarly in both approaches, for example, *Cancel*.
- Generality: a scenario is a special case of a pattern. For example, "Novice interfaces for users in unfamiliar contexts" is a special case of "provide a *Workflow model"*.

Some of the scenarios have not been considered in our approach:

- "Account human needs and capabilities when interacting, keep coherence through multiple views, define upgrades similar to previous ones, provide easily modifiable test points for evaluation and design interfaces" are the results of specific actions to be taken during the development process and are not in keeping with the definition of usability pattern considered in our work. So, the issues referred to by these scenarios need to be dealt with within the whole development process, not specifically in terms of architecture. The STATUS project has a workpackage that deals with modifications in the development process to improve final system usability.
- "Minimize user recovery work due to system errors" refers to errors made by the software system and not by the users. Traditionally [3][4], usability efficiency deals with the prevention of and recovery from user, not system, errors.
- "Allow searching by different criteria" and "Provide alternative secure mechanisms" are specific requirements and not really usability patterns as they are considered in our work.

Table 3. Usability patterns / scenario relationship

Usability Patterns	Relationship	Scenarios
Different languages	similarity	Support international use
Different access methods	generality	Maintain device independence
Alert	generality	Verify resources before beginning an operation
Status indication	similarity generality	Present system state Predicting task duration
Shortcuts (key and tasks)		
Form/field validation	similarity	Checking for errors
Undo	similarity	Undo
Context-sensitive help	instantiation	Provide good help
Wizard	instantiation	Provide good help
Standard help	instantiation	Provide good help
Tour	instantiation	Provide good help
Workflow model	generality	Novice interfaces for user in unfamiliar contexts
History logging		
Provision of views	content similarity content	Make views accessible Provide reasonable set of views Quick navigation into a view
User profile		
Cancel	similarity	Cancel
Multi-tasking	content content	Use applications concurrently Allow to quick switch back and forth between different tasks
Commands aggregation	similarity	Aggregate commands
Action for multiple objects	similarity	Aggregate data
Reuse information	similarity	Reusing information

It might be interesting to note the similarities and differences between the two approaches, for example, the generality or specificity levels of the usability mechanisms employed by the two approaches. In this respect, it would be worthwhile discussing at the workshop the strengths and weaknesses of using usability mechanisms with differing detail levels from the viewpoint of practitioners. Note, however, that both approaches will agree with the idea of relating the different aspects of usability to the architecture through architectural patterns. These patterns will show how the scenario (Bass et al.'s approach) or the usability pattern (STATUS approach) can be represented at an architectural level.

The following section shows how we have developed design solutions that can be used to incorporate the usability mechanisms specified by the usability patterns into a software system. These design solutions are the architectural patterns mentioned above.

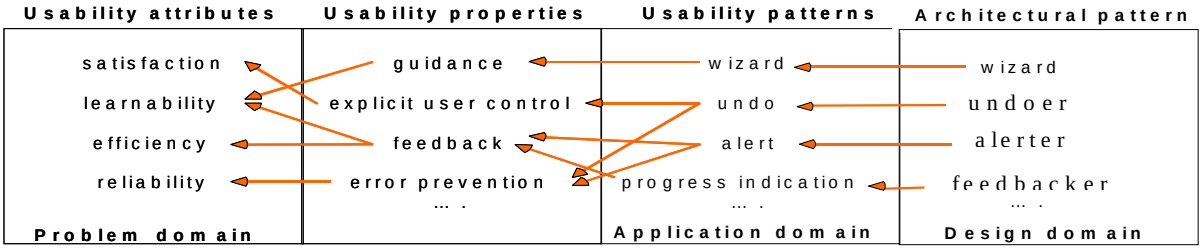
3. Architectural Patterns for Usability Support

The most widely used concept of pattern in software development is the design pattern, and it is used particularly in the object-oriented paradigm. In this context, a design pattern is a description of classes and objects that work together to solve a particular problem [11]. These patterns show a solution to a problem, which has been obtained from its use in different applications. Note, nevertheless, that a design pattern can be seen as a unique or original solution.

Besides the idea of usability pattern, we also used the concept of architectural pattern. Given that we have defined a usability pattern as a mechanism to be applied to the design of a system architecture in order to address a particular usability property, an architectural pattern will determine how this usability pattern is converted into software architecture. In other words, what effect the consideration of a usability pattern will have on the components of the software architecture. Abstracting the definition of design pattern, an architectural pattern can be defined as a description of the components of a design and the communication between these components to provide a solution for a usability pattern. Like design patterns, architectural patterns will reflect a possible solution to a problem, the implementation of a usability pattern, although this will be a unique solution in each case.

Therefore, the architectural pattern is the last chain in the usability attribute, property and pattern chain that connects software system usability with software system architecture. Accordingly, another column can be added to Table 1, as shown in Table 4.

Table 4. Usability attributes/properties/pattern and architectural pattern relationships



3.1. Procedure for outputting architectural patterns for usability

In the following, we describe the procedure followed to identify the architectural patterns that design the proposed usability patterns. This procedure is composed of two parts:

1. Application of a process of induction to abstract the architectural patterns from particular designs for several projects developed by both researchers and practitioners. For this purpose, we took the following steps:
   1.1. We asked designers to build the design models for several systems without including usability patterns.
   1.2. For each usability pattern, we asked designers to modify their earlier designs to include the functionality corresponding to the pattern under consideration.
   1.3. For each usability pattern, we abstracted the respective architectural pattern from the modifications made by the developers to the design.
   This process was carried out on two applications: restaurant orders and tables management and ride control and maintenance at an amusement park.
2. Application of the architectural patterns resulting from the previous step to several developments to validate their feasibility.

To illustrate this process of induction, below we show part of the induction of one of the architectural patterns on the restaurant orders and table management application, specifically the pattern related to the usability pattern *Progress Indication*.

The sequence diagram shown in Figure 1 and the class diagram shown in Figure 2 show part of the design of this application, specifically the part related to the entry of the menu requested by the restaurant customer. This part was designed without taking into account the usability property on feedback. As we can see from the diagrams, the system user is not receiving any information on what the software system is doing. Figure 3 and Figure 4 show the sequence and class diagrams, respectively, now considering the inclusion of the usability pattern for *Status Indication* on this same functionality. We can see how the inclusion of an object of the Feedbacker class provides the user with information on system operation.

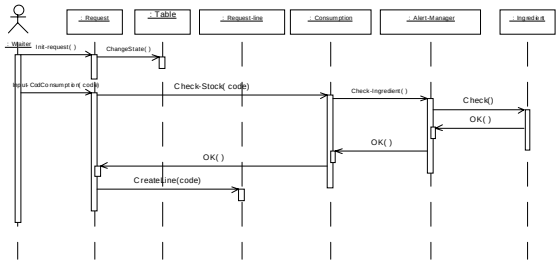


Figure 1. Interaction diagram without usability pattern

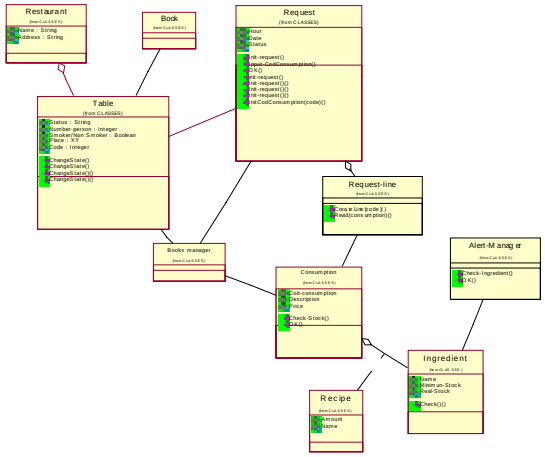


Figure 2. Class diagram without usability pattern

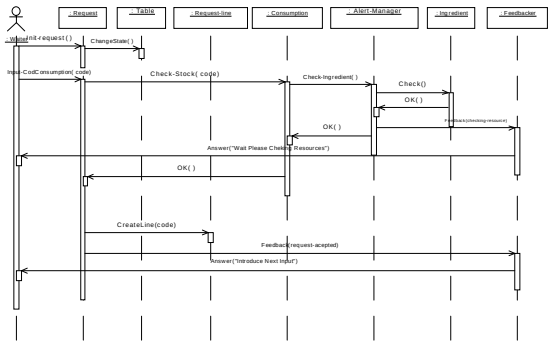


Figure 3. Interaction diagram with usability pattern

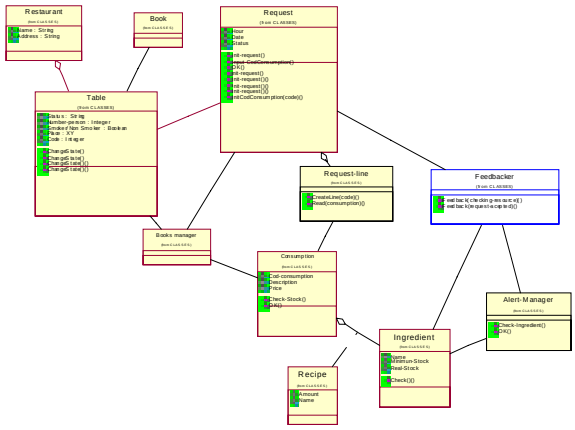


Figure 4. Class diagram with usability pattern

From this design and others for the same system and the other application created by other developers, we have abstracted a general design solution as shown in Figure 5.

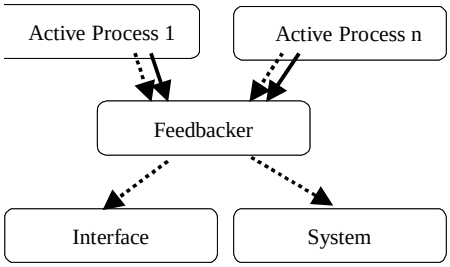


Figure 5. Generic solution for the Feedbacker pattern

Likewise, we have applied this inductive process to the other usability patterns to develop the respective architectural patterns. For details of this process, see Techniques and Patterns for Architecture-Level Usability Improvements [13]. Table 5 summarises the architectural patterns defined together with their underlying usability patterns.

Table 5. List of usability and architectural patterns

Usability Patterns	Architectural Pattern
Different languages	Language-recogniser
Different access methods	Device- recogniser
Alert	Alerter
Status Indication	Feedbacker
Shortcuts (key and tasks)	Shortcutter
Form/field validation	Checker
Undo	Undoer
Context-sensitive help	Sensitive-helper
Standard help	Standard helper
Tour	Guided helper
Workflow model	Filter
History logging	Logger
Provision of views	Viewer
User profile	Profiler
Cancel	Canceler
Multi-tasking	Dispatcher
Commands aggregation	Aggregator
Action for multiple objects	Multi-executer
Reuse information	Reuser

3.2.Description of the Architectural Patterns

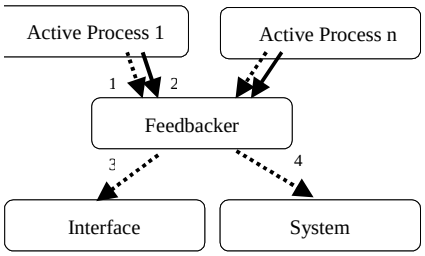
Since the ultimate aim of this work is to provide a set of architectural recommendations to improve the usability of the software systems, these recommendations will be described in an architectural pattern catalogue. Each pattern in this catalogue has to be described according to following elements:

- **Pattern Name** - Patterns must have suggestive names that give an idea of the problem they address and the solution in a word or two.
- **Problem** – This describes when to apply the pattern and in which context. In the case of architectural patterns, the problem will refer to a specific usability pattern to be materialised.
- **Solution** – This describes the elements that make up the architecture, their relationships, responsibilities, etc. The solution does not describe a definite design, as a pattern can be seen as a template that can be applied in many different situations. Particularly, the solution for a specific pattern will be specified from :
  - **Graphical representation** - A figure that represents the components of the architecture and their iterations. Numbered arrows between the different components will represent the iterations. The arrows with solid lines specify the data flow, while the dotted lines represent the control flow between the components.
  - **Participants** – A description of the components that take part in the proposed solution and the iterations (represented by arrows) to determine how they are to assume their responsibilities.
- **Usability benefits** - Description of which usability aspects (usability properties) can be improved by including the right pattern.
- **Usability rationale** - A reasonable argumentation for the impact of pattern application on usability, that is, what usability attributes have been improved, and which ones may get worse. Initially, this feature will be completed with information coming from others authors or from the experience of the consortium members. However, once the patterns have been applied to real applications, this field will be filled in with empirical experience.
- **Consequences** - Impact of the pattern on other quality attributes, like flexibility, portability, maintainability, etc. As for the above feature, this one will be filled in with the results of empirical experience.

- **Related patterns** - Which architectural patterns are closely related to this one, and what differences there are.
- **Implementation of the pattern in OO** - The architectural patterns provided are patterns that can be applied in any development paradigm. However, as these patterns have been obtained and refined for OO applications, we will provide guides tending to address pattern application in this field. Basically, we will describe the classes deriving from the pattern's main components. These guides are illustrated in the example shown in the following section.
- **Example** of the application of the pattern in question.

In the following, we show how the architectural pattern "Feedbacker" is described:

- Pattern Name: Feedbacker
- Problem: The user should be provided with information pertaining to the current state of the system.
- Solution:
  - Graphical representation:



  - Participants:
    - Active-process i: this module has been represented more than once, because there may be several processes running simultaneously that request feedback (1), and it will be each active process that sends the information that it wants to be fed back (1) to the Feedbacker.
    - Feedbacker: this module is responsible for receiving the request and data (1) (2) that indicate the type of feedback requested and the data to be fed back from each active process. Additionally, it must know the recipient of this feedback and will send this feedback either to another part of the system (4) and/or to the interface (3) to inform the user. [10] specifies some guidelines on how to display this feedback on the user interface, for example, how often it should be refreshed or where the particular information should be placed. These details should be taken into account during low-level design.
    - Interface: the interface is responsible for receiving the feedback and displaying it to the user (3).
    - System: this component is optional and represents other parts of the system that should be informed of the feedback (4).
- Usability benefits: giving an indication of the system's status provides feedback to the user about what the system is currently doing, and what the result of any action they take will be.
- Usability rationale: providing feedback gives the user information about what the system is working on and whether the application is still processing or has died. So, this pattern raises *satisfaction*.
- Consequences:
  - This pattern prevents additional *system load* by avoiding retries from users [10].
  - This pattern improves system *maintainability* because it channels the feedback information better as compared with when the feedback exists but is indiscriminately emitted by any other system module.
- Related patterns:
- OO implementation: This architectural pattern will give rise to a Feedbacker class specialised in informing the user and the system of what is going on. This means that all the classes that want to report something to the system must report to the feedback manager, Feedbacker, so that this manager can properly distribute this information either inside or outside the system.
- Example: This section would detail one of the examples used to get this pattern, for example, the example shown in Figure 4 and Figure 5.

4. Future Work

Now that we have satisfactory solutions for the architectural patterns, the next step is to apply these to different designs. The aim is to check the feasibility of the solution provided by each pattern and derive and refine recommendations for its application by practitioners. This task is part two of the procedure for outputting architectural patterns described in section 3.2.

After generating the architectural patterns we propose to present a set of practical guides that provide practitioners with information on:

- How to select an architectural pattern, for example, from the usability attributes that are to be enhanced in each design and the impact on the other quality attributes.

- How to use an architectural pattern for inclusion in a given design.

The effort to improve software architecture with regard to usability presented in this paper is related to another important part of STATUS, which is the assessment of this architecture with respect to usability. This assessment is being conducted in two ways in the project: a scenario-based architectural assessment and a simulation-based architectural assessment (two papers addressing this research have also been submitted to this ICSE workshop). This evaluation will yield the set of shortcomings that a given software architecture has with respect to certain usability attributes or parameters. The architectural patterns could, therefore, be used to implement usability improvement solutions for the detected shortcomings.

However, the idea of architectural patterns can also be used independently of the architecture evaluation, as they provide design solutions for certain usability requirements (any *usability properties* included in the requirements specification). The consideration of these usability requirements at the start of development and later in design, by means of architectural patterns, is expected to provide improvements in final system usability.

We have to take into account that the final software system usability has to be validated and measured when the system in question has been built and is operational. Therefore, we will have to wait until these results have been applied to real projects to get empirical data to properly verify the as yet intuitive benefits that the use of architectural patterns can provide for software systems usability. Half of the STATUS project time has been allocated to validating the ideas of this paper and the remainder of the research with the industrial partners. This validation will kick off in March 2003 and run until June 2004.

5. References

[1]. J Bosch. Design and Use of Software Architectures: Adopting and Evolving a Product Line Approach, Pearson Education, Addison-Wesley, 2000.

[2] X. Ferré, N. Juristo, H. Windl, L. Constantine. Usability Basics for Software Developers. *IEEE Software*, vol 18 (11), p. 22-30

[3] L. L. Constantine, L. A. D. Lockwood. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Addison-Wesley, New York, NY, 1999

[4] J. Nielsen. *Usability Engineering*. AP Professional, 1993.

[5] B. Shackel. "Usability – context, framework, design and evaluation". In *Human Factors for Informatics Usability*. pp 21-38. Ed. by B. Shackel and S. Richardson. Cambridge University Press, 1991.

[6] A. Andrés, J. Bosch, A Charalampos, R. Chatley, X. Ferre, E. Forlmer, N. Juristo, J. Magee, S. Menegos, A. Moreno. *Usability attributes affected by software architecture*. Deliverable 2. STATUS project, June 2002. Http://www.ls.fi.upm.es/status

[7] Perzel,, D Kane D. (1999) *Usability Patterns for Applications o the World Wide Web*. PloP'99

[8] G Cascade. *Notes on a Pattern Language for Interactive Usability*, Proceedings of the Computer Human Interface Conference of the ACM, Atlanta, Georgia, 1997.

[9] J Tidwell. *Interaction Design Patterns.* Pattern Languages of Programming 1998, Washington University Technical Report TR 98-25.

[10] M. Welie, H Troetteberg. *Interaction Patterns in User Interfaces*. PloP'00.

[11] E Gamma, R Helm, R Johnson, J Glissades. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1998.

[12] L Bass, E Bonie, J Kates. *Achieving Usability Through Software Architecture*. Technical Report. CMU/SEI-2001-TR-005, March 2001.

[13] N. Juristo, M. López, A. Moreno, M Sánchez. *Techniques and Patterns for Architecture-Level Usability Improvements*. Deliverable 3.4. STATUS project. Http://www.ls.fi.upm.es/status.