A Software Architectural View of Usability Patterns

Xavier Ferre[(1)], Natalia Juristo[(1)], Ana M. Moreno[(1)] & M. Isabel Sánchez[(2)]

(1) Facultad de Informática, Universidad Politécnica de Madrid,
Boadilla del Monte, Spain
{xavier, natalia, ammoreno}@fi.upm.es

(2) Facultad de Informática, Universidad Carlos III de Madrid, Leganés, Spain
misanche@inf.uc3m.es

Abstract: Usability is one of the key quality attributes in software development. The content of this paper is part of the research conducted within the European Union IST STATUS project, related to the development of techniques and procedures for improving usability of software architecture designs. In this paper, we will focus on the possible improvement of usability at software design time. For this purpose, we have identified, from both the literature and the project's industrial partners' experience, what we have called usability patterns. The usability patterns represent twenty usability mechanisms, for example, undo, cancel, multiple-languages, etc., which improve final system usability and have an effect upon the design of the software system in which they are implemented. We also present possible design solutions for incorporating the respective usability mechanisms into a software system design. The design solutions have been obtained by means of an inductive process that guarantees that these solutions are possible, albeit not necessarily the only solutions.

Keywords: Software architecture, software design, software usability

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 (Bosch, 2000). The idea underlying this relationship is that a software architecture needs to be explicitly designed so that the final system satisfies specific quality attributes. The most widespread studies on this subject refer to quality attributes like performance or maintainability (Bengtsson et al., 2000).

Usability is not usually considered in software architecture, due to the widespread assumption between software developers that usability has to do only with the visible part of the user interface. This assumption is not true, since usability is strongly related to the interaction part of the system and it must be considered when designing the rest of the system, not just the user interface (Ferre et al., 2001). Decisions in software architecture can severely compromise the usability of the final system, for example if an "undo" mechanism is not devised when the software architecture is established, it will be very costly to incorporate afterwards. Therefore, it would be interesting to relate final system usability with the kind of decisions taken in the design of the software architecture. It is in this context that the STATUS (SofTware Architecture That supports USability)[i] project emerged, whose objective is to develop techniques and procedures to be incorporated during the design of a software system to achieve improvements in the usability of the system under construction. Traditionally, usability evaluation before implementation is directed towards user interface prototypes, and not to design specifications. Our approach aims to incorporate well-known usability heuristics into the design of the system before it is implemented, shaped in the form of patterns.

When no usability expertise is available in the software architecture team, usability requirements may not be considered as being related to software

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

architecture. This situation leads to designs with usability problems that would be much easier to correct if discovered in the software architecture design stage. Figure 1 shows how the usability evaluation can be handled in the software architecture design process, following the same process proposed in the software architecture field for other quality attributes, such as performance or maintainability.

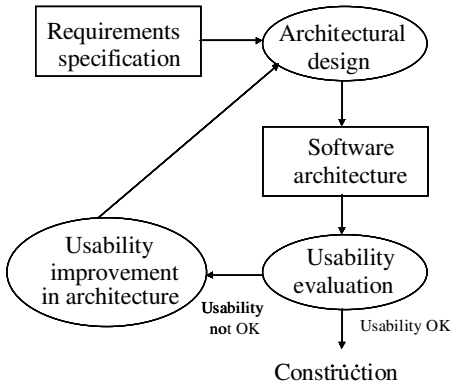


Figure 1: Architectural design method for usability proposed in STATUS

The design process starts with the construction of a model of the software architecture from a set of functional requirements. Some usability requirements that can be evaluated in design time must have already been established as well. Although software engineers will not design this preliminary model to be unreliable or under perform on purpose, most non-functional requirements are generally studied later on. Accordingly, the preliminary design derived is evaluated with respect to some quality attributes, usability in this particular case.

The evaluation of usability of an architectural design is far from being easy, since software architecture is represented by means of a set of design models, which cannot be tested directly with a user. One possibility is to dynamically simulate the architecture, and such approach is being pursued as one of the lines of research in the STATUS project. The techniques for this kind of evaluation are described in (Uchitel et al., 2003). Another possibility is based on a static perspective where usability properties are searched for in the design specifications. Techniques to evaluate a software architecture from a static perspective are described in (Folmer and Bosch, 2003).

In this paper, we will focus on the latter, the static perspective, studying possible usability improvements that can be made at design time. Our approach is based on identifying the usability properties (such as guidance, explicit user control, and so on), that are needed for the software product being developed, and providing software developers with design solutions that address such usability properties. The design solutions proposed are shaped in the form of patterns, which express common usability heuristics of the HCI field. In this way we express usability knowledge by using the terms and concepts which are employed at the software architecture design stage. For this purpose, section 2 shows the approach taken to decompose usability into several 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, possible solutions for incorporating aspects that will improve final system usability into the design.

Finally, section 4 briefly discusses the use of the approach taken in this paper.

2 Usability Attributes, Properties and Patterns

Software systems usability is usually evaluated on the finished system, trying to assign values to the classical *usability attributes* (Constantine and Lockwood, 1999; Nielsen, 1993; Shackel, 1991):

- 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.
- 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.

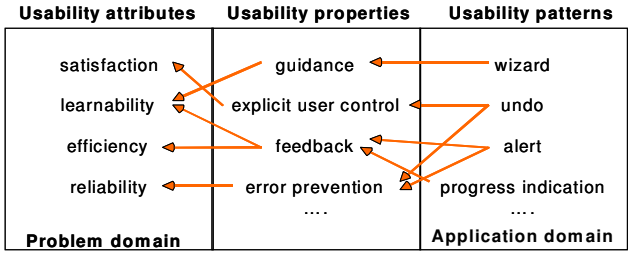
However, the level of these usability attributes is too high for us to be able to examine what mechanisms should be applied to a software architecture to improve them. Therefore, the philosophy followed in STATUS was to decompose these attributes into two intermediate levels of concepts closer to the 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 to be satisfied by 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, provide undos, alerts, command aggregations, wizards, etc.).

The procedure followed to identify the relationship between usability attributes, properties and patterns is detailed in (Andrés et al., 2002). We took a top-down approach from usability attributes (defined in the literature), through usability properties (derived, on the one hand, from heuristics and guidelines given in the literature for improving usability and, on the other, from the experience of the project's industrial partners), to finally identify usability patterns.

A subset of the above-mentioned relationship is outlined in Table 1, showing 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" (Perzel and Kane, 1999). The usability aspects dealt with by these patterns refer basically to user interfaces, which is why these patterns are also called user interface patterns (Cascade, 1997) or interaction design patterns (Tidwell, 1998). As indicated by authors like Welie and Troetteberg (Welie and Troetteberg, 2000), 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 (Cascade, 1997; Tidwell, 1998; Welie and Troetteberg, 2000): feedback, wizard, provide the user with all information needed in the same window, mark required fields when filling a form, etc.

The differences between the usability patterns proposed in this paper 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 outlined in this paper address the mechanisms to be considered in a software architecture, dealing with usability aspects in the early stages of the development process. For example, the solution proposed by Welie (Welie and Troetteberg, 2000) 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". From a different perspective, our solution for this same pattern covers the components to be added to a software architecture and the relationships among these components.

The second column in Table 2 lists the usability patterns proposed in the STATUS project. The first column of the table shows the usability properties related to each pattern.

Table 2: List of usability patterns

Usability Property	**Usability Pattern**
NATURAL MAPPING	
CONSISTENCY (functional, interface, evolutionary)	
ACCESSIBILITY (internationalisation)	Different languages
CONSISTENCY, ACCESSIBILITY (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 logging
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 the system 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.

3 Architectural Usability Patterns

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 (Gamma et al., 1995). These patterns show a solution to a problem, which has been obtained from its use in different applications. Note, nevertheless, that a design pattern should not be seen as a unique or original solution, but as a possible solution.

Besides the idea of usability pattern, we also use the concept of architectural pattern in the STATUS project. 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 system 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.

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 Table 3 shows.

3.1 Procedure for outputting architectural patterns for usability

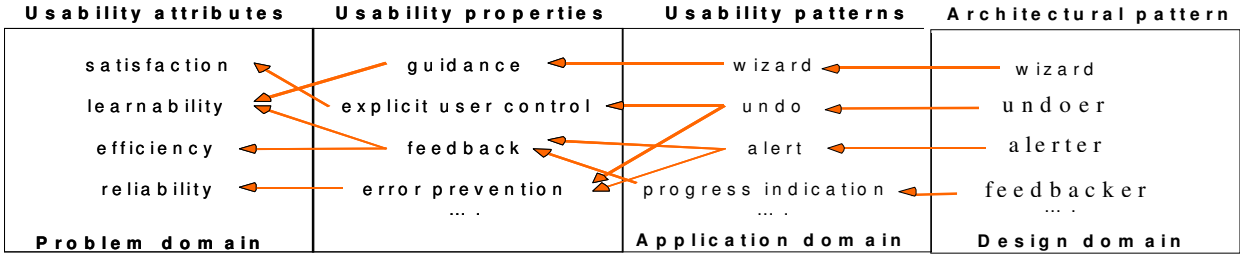
In the following, we describe the procedure followed to identify the architectural patterns that design the proposed usability patterns. This procedure is composed of two parts:

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 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.
2. The application of the resulting architectural patterns from step 1 to several developments to validate their feasibility.

To illustrate this process, we show below how the *reusing information* pattern was abstracted from a restaurant orders and tables management application.

The sequence diagram shown in Figure 2 and the class diagram shown in Figure 3 show part of the design of this application, specifically the part related to the entry of the menu requested by the restaurant customer. Figure 4 and Figure 5 show the sequence and class diagrams, respectively, now considering the inclusion of the *reusing information* usability pattern for this same functionality. We can see how the inclusion of the *reuser* class provides the possibility of repeating a previous operation.

Table 3 - Usability attributes/properties/pattern and architectural pattern relationships



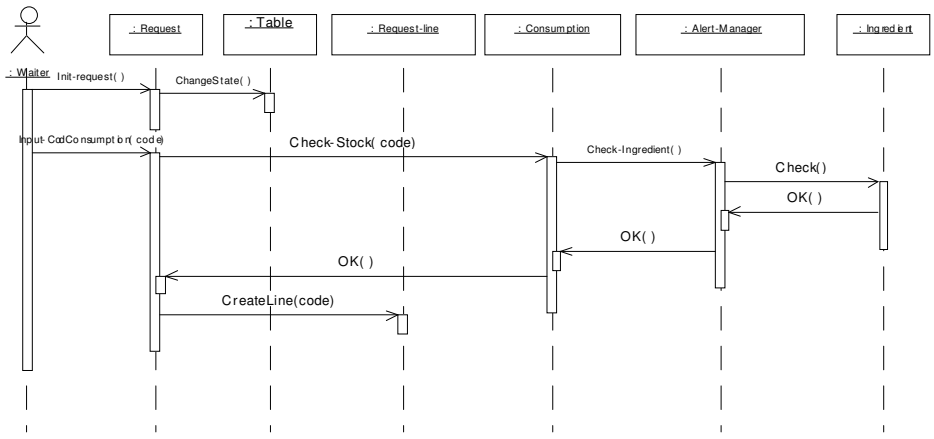


Figure 2: Interaction diagram without the reusing information pattern

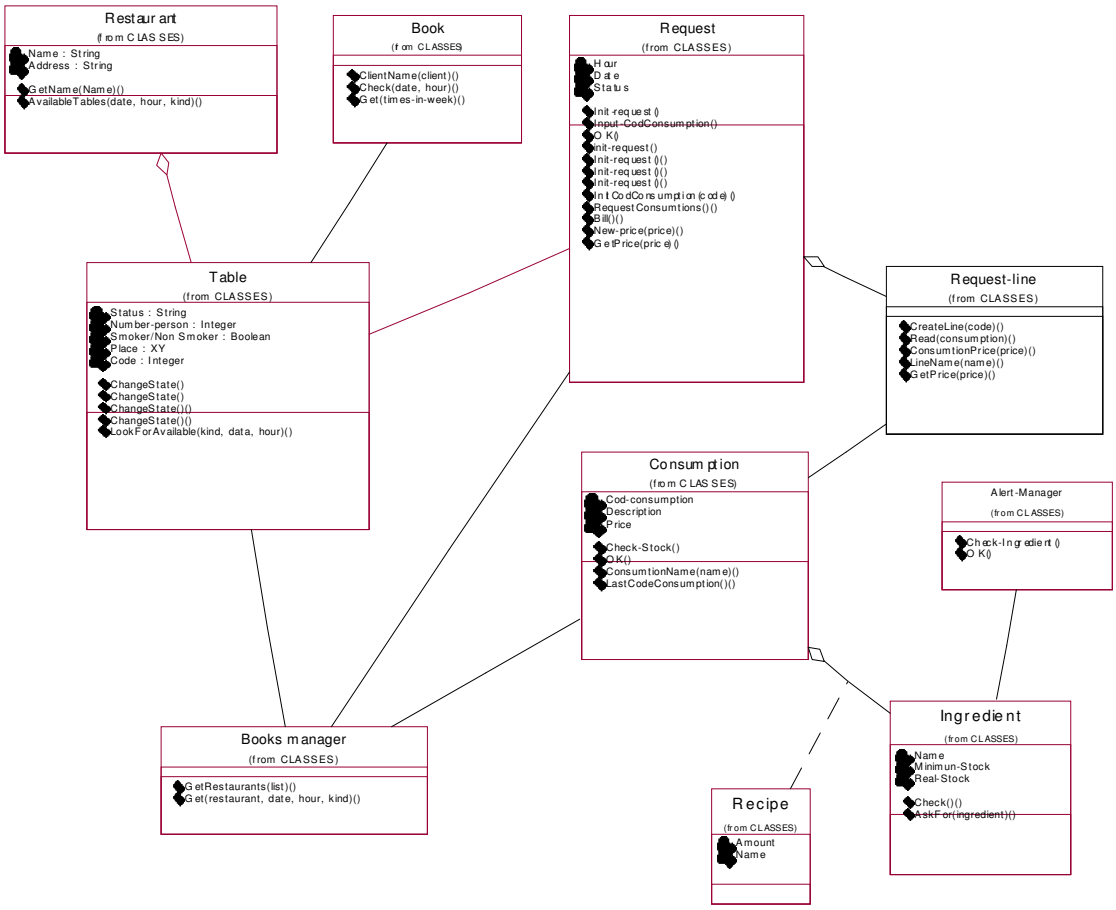


Figure 3: Class diagram without the reusing information pattern

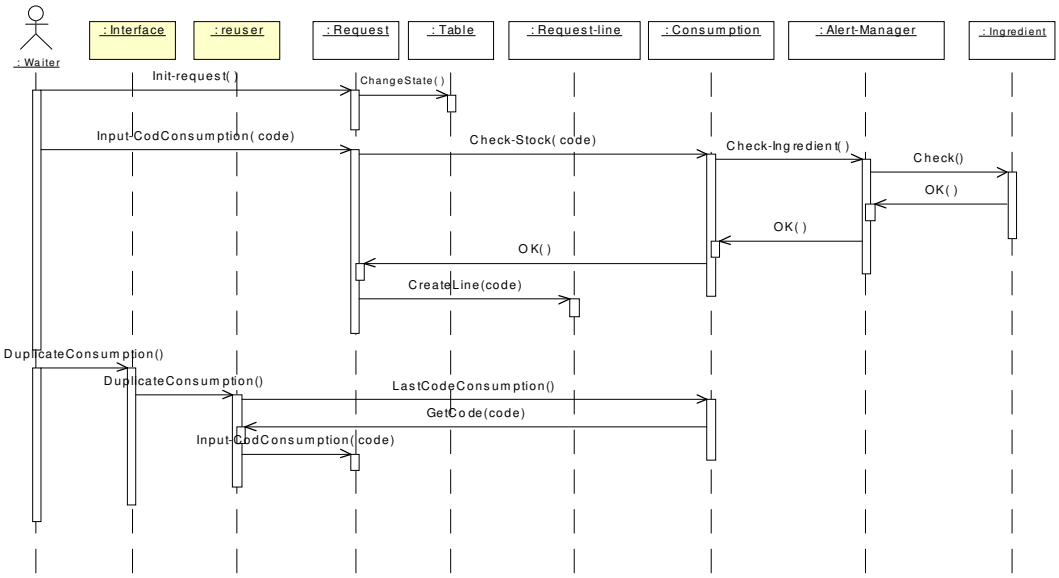


Figure 4: Interaction diagram with the reusing information pattern

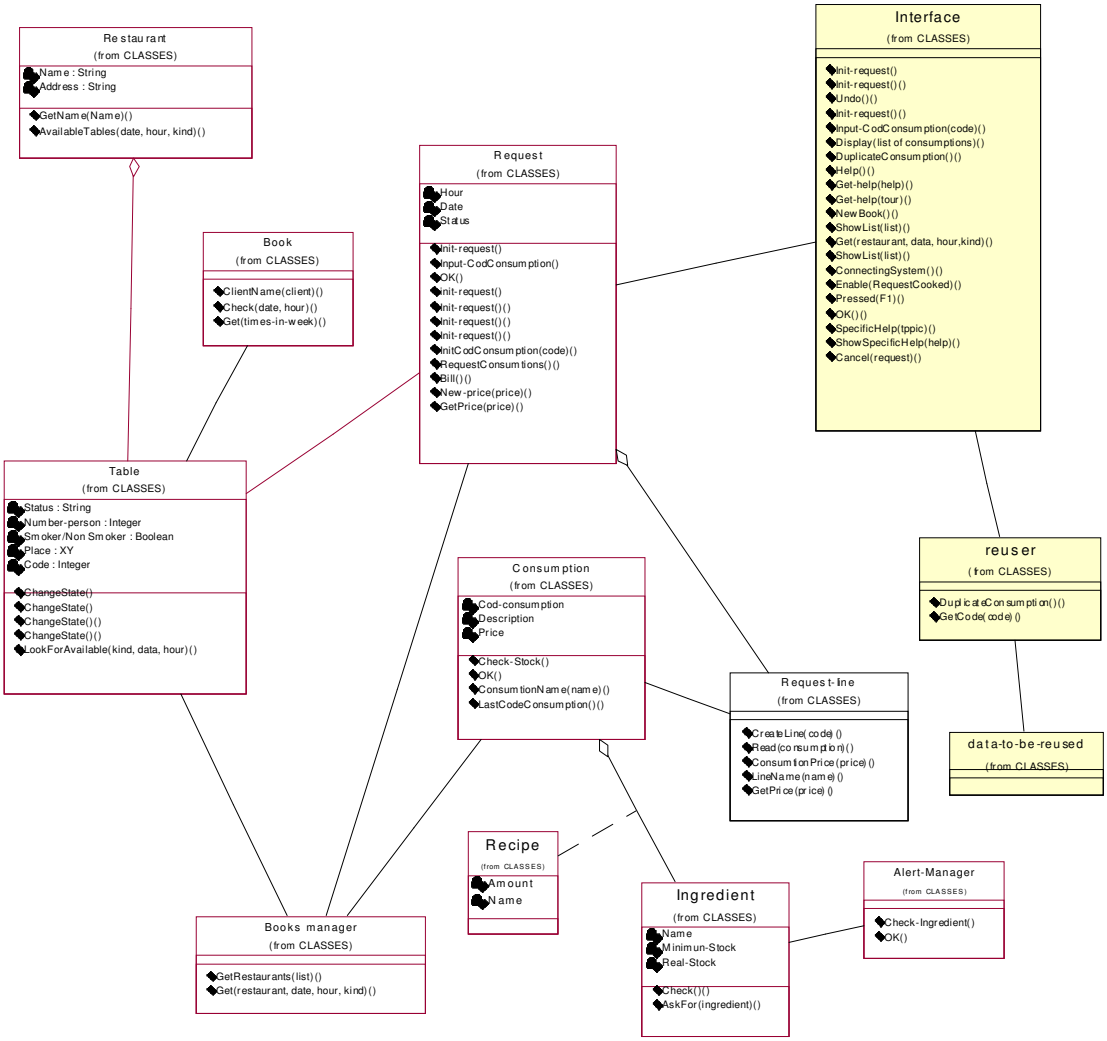


Figure 5: Class diagram with the reusing information pattern

From this design and others created by other developers, we have abstracted a general design solution as shown in Figure 6.

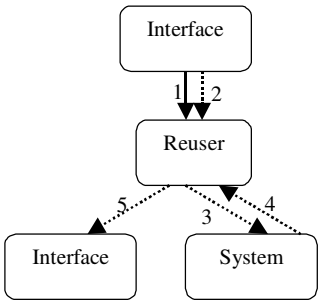


Figure 6: Generic solution for the reusing information pattern

Likewise, we have applied this inductive process to the other usability patterns to develop the respective architectural patterns. This process is detailed in (Juristo et al., 2003).

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 software systems, these recommendations will be described in an architectural pattern catalogue. Each pattern in this catalogue has to be described according to the following fields:

- **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 by means of the following elements:
    - **Diagram** - A figure that represents the components of the architecture and their relations. Numbered arrows between the different components will represent the relations. 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 relations (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 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 have got worse. Initially, this feature will be completed with information coming from other 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 the results of 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 Figure 7, we show how the architectural pattern *reusing information* is described. The other architectural patterns that provide design solutions for the usability patterns proposed in the STATUS project have been described similarly.

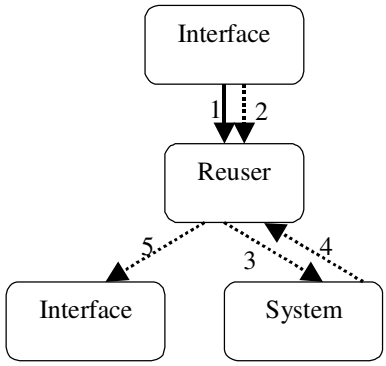
4 Discussion

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 how to determine 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. 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 then be used to implement usability improvement solutions for the detected shortcomings.

- **Pattern Name**: Reusing Information.
- **Usability Mechanism**: This pattern enables the user to move data from one part of a system to another. So users should be provided with automatic (e.g., data propagation) or manual (e.g., cut and paste) data transports between different parts of a system.
- **Solution**:
  - Diagram:



  - Participants:
    - Interface: collects the data to be processed by the reuser pattern and finally displays the operation results (if the user needs to see the result). Interface sends the data to be processed (1) and the function requested by the interface (2), i.e. copy, paste, move, etc., to Reuser. Also, once the reuser pattern has been applied the results of the requested function will be displayed on the interface (5), unless the requested function was "copy".
    - Reuser: is the module that gathers the information provided by the interface and manipulates these data according to the requested function (copy, paste, move, etc.). Reuser receives the data to be manipulated as well as the function to be executed (1) (2). If Reuser does not store the data to be manipulated internally, it has to send these data to the system (3), as happens, for instance, with the Copy function. Also if Reuser does not store the data internally, it has to ask for these data from the part of the system where they are stored (4) as happens with the paste or move functions.
    - System: this component is optional and is only necessary when the Reuser module does not store the data internally.
- **Usability benefits:** The reuse of data in an application as well as across different applications minimises users' cognitive load and also inputs fewer errors into the process. It also improves the adaptability of the application or applications that enable data reuse.
- **Usability rationale:** By preventing the error input by users, the application of this pattern improves system reliability. User efficiency is also improved. Additionally, by building a more adaptable system, the satisfaction of the end user is improved too.
- **Related patterns:**
  - System performance will be better if the information to be reused is stored in the Reuser module rather than in another part of the system, because this reduces the system interaction level.
- **Pattern Implementation in OO:** Interface generates some classes. Reuser generates one or more "Reuser" classes, furnished with the manipulation methods (copy, paste, move, etc.) and "Data-to-be-reused" classes, which store the data to be manipulated in the class or through a link to another one. In this case, it was decided to store the data outside the reuser class to respect the encapsulation principle.
- **Example**: *This section would detail one of the examples used to get this pattern, for example, the design shown in Figure 4 and Figure 5 along with the corresponding explanations.*

Figure 7: Example of architectural pattern: reusing information

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. At the time of writing, one of the industrial partners was applying the patterns in a real project. As soon as the system has been developed, the classical usability evaluations will be run to check the improvements in the final usability of the system constructed.

References

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

Bengtsson P., Lassing N., Bosch J. and van Vliet H. Analyzing software architecture for modifiability. *Journal of Systems and Software*, 2000.

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

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

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

Ferre X., Juristo N., Windl H., Constantine L. Usability Basics for Software Developers. *IEEE Software*, vol 18 (1), January/February 2001. pp. 22-29.

Folmer E., Bosch J. Usability patterns in Software Architecture. Proc. of HCI-International'2003, Crete, June 2003.

Gamma E., Helm R, Johnson R, Vlissides J. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

Juristo N., López M., Moreno A., Sánchez M. *Techniques and Patterns for Architecture-Level Usability Improvements. Deliverable 3.4. STATUS project*, April 2003. http://www.ls.fi.upm.es/status.

Nielsen J. *Usability Engineering*. AP Professional, 1993.

Perzel D., Kane D. Usability Patterns for Applications of the World Wide Web. *Proc. of PloP'99.*

Shackel B. Usability – context, framework, design and evaluation *in* B. Shackel and S. Richardson (eds.) *Human Factors for Informatics Usability*. Cambridge University Press, 1991. pp. 21-38

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

Uchitel S., Chatley R., Kramer J. and Magee J. LTSA-MSC: Tool Support for Behaviour Model Elaboration Using Implied Scenarios. *Proceedings of TACAS ' 03*Warsaw April 2003.

Welie M., Troetteberg H. Interaction Patterns in User Interfaces. *Proc. of PloP'00.*