State of the art

# Interface generation

## Automated design tools

#### Janus

Let’s talk about Janus. Software developed in 96.

## Specification-based tools

#### Its

The ITS system was developed in the early 90’s by the IBM research and development department. It was successfully used to develop several large applications like the information kiosks for Seville Expo 92.

The ITS architecture divides applications in four layers. The action layer implements the application’s back-end computations. The dialog layer defines the content of the user interface independent of its style, much like an abstract user interface model. Content specifies the objects included in each frame of the interface, the flow of control among frames, and what actions are associated with each object. The style rule layer defines how the dialog is presented to the user in terms of appearance and interaction techniques. Finally the style program layer implements the primitive toolkit objects that are composed by the rule layer into complete interaction techniques.

Before ITS there were mainly two types of layered architectures that provided the required flexibility in application development. User Interface Management Systems (UIMS) and toolkits. UIMS separate the business layer from the interface. Back-end computations are separated from the dialog control and style. Style, however, is often treated in a single interface layer. Toolkits separate style from the application. Dialog control remains in the back-end while the implementation of interaction techniques is hidden in a code library.

The four layers in ITS present a series of advantages that separate this tool from its predecessors. Like in previous UIMS there is a separation from back-end computations and the interface itself. By separating the action layer from the dialog allows actions to be reused in different applications.

Splitting the interface into separate layers for style-independent dialog, rule base and toolkit also gives some benefits. First, the dialog remains independent of style. A dialog can be mapped into any different style simply by firing the appropriate rule. Second, interface designers control style rather than application programmers. The rule layer represents the selection criteria for all interaction techniques.

Each layer in ITS architecture corresponds to one of four roles in application development: application programmer, application expert, style expert and style programmer. An application expert is familiar with the domain of the application. The application expert typically is neither trained in software development or part of an information systems department. In ITS, the application expert is the author of the dialog. A style expert may be a graphic artist or a human factors engineer. Rules give them direct control over style in ITS.

ITS is a specification-based system. The main difference between these tools from automated-design tools like JANUS is that the modeling language is open whereas in automated-design tools are closed. By lifting this limitation for the developers the final result can have a higher level of quality although it’s dependent from the capabilities of the developer himself.

Even though ITS is a specification-based system, this doesn’t mean that developers have to specify every feature of every individual window. Developers are forced to specify the content of dialogs which is equivalent to the abstract user interface and this is, as been proved to be by experience, the most difficult model to generate by the automated-design tools. The style rules layer or the concrete user interface model doesn’t have to be totally specified. This doesn’t mean that ITS generates this model but that developers can reuse rule sets from libraries that contain the abstract to concrete mapping for significant portions of the interface specifications.

#### Gadget

Recent work is beginning to reveal that numerical optimization can play a role on modern approaches for generating interfaces and displays. GADGET is a toolkit for optimization-based approaches to interface and display generation.

Although optimization-based techniques appear to offer several potential advantages most programmers are intimidated or uncomfortable by the math required to program an optimization. Although optimization toolkits are available (cite), they typically require substantial specialized knowledge because they have mostly been designed for physics simulations and other traditional optimization problems.

GADGET provides a set of abstractions for many optimization concepts along with a set of mechanisms to help programmers quickly create optimizations, including an efficient lazy evaluation framework, a powerful and configurable optimization structure, and a library of reusable components.

A programmer creating an optimization using the GADGET toolkit needs to supply three essential components: an initializer, iterations and evaluations. The initializer creates the initial solutions to be optimized. This might be based on an existing algorithm, or done randomly. Iterations are responsible to transform one potential solution into another, typically using models that are at least partially random. Finally, evaluations are used to judge the different notions of goodness in a solution.

There’s a standard framework to abstract the concepts and constructs behind evaluations. GADGET allows programmers to focus on creating evaluations to measure criteria that are important to a problem. GADGET then combines these evaluations and uses them to between possible solutions to a problem. This process is divided into five stages.

First the framework presents each evaluation with the current potential solution, which is called the prior solution. Each evaluation returns an array of double values representing its interpretation of the prior solution. This collection of arrays of double values is called the prior result.

On the second stage the framework uses an iteration object to modify the prior solution and create a new one, called the post solution.

Third, the framework presents the post solution to each evaluation. Each individual evaluation returns interpretations that are then combined to create a post result.

In the fourth step, the framework uses a method to compare the prior result with the post result. This is possible by requiring each evaluation to be capable of comparing two arrays of double values that it has created and providing a double value in the range of -1 to 1, where -1 indicates the evaluation has a strong preference for the prior solution, 0 indicates that the evaluation is indifferent and 1 indicates that the evaluation has a strong preference for the post solution.

Finally, the result of this comparison indicates whether the framework should go with the post solution or revert to the prior solution. To choose between them, the values retrieved from the fourth step are multiplied by the weight associated with its respective evaluation and then summed. If the sum is greater than 0, the framework prefers the post solution, else it reverts to the prior solution.

#### Supple

User interfaces delivered with today’s software are usually created in a one-size-fits-all manner by making assumptions about the needs, abilities and preferences of their users and the characteristics of the device they’re running. The objective of Supple is to automatically generate personalized adaptive user interfaces at run time in order to be fit for every user or device.

#### IdealXML

Interface Development Environment for Applications specified in usiXML (IdealXML). This is a solution to integrate experience (represented in patterns) into model-based user interface development, using the usiXML notation. Its main objective is to provide a single editor where designers can use and transfer design knowledge that has emerged from experience. They can use patterns to profit from their own experience or from others. [Cite IdealXML]

IdealXML manages a pattern repository using a hierarchical structure. At the top of this structure there are some different models related with the MB-UIDE like the domain model, task model, presentation model and mapping model.