HAL
archives-ouvertes.fr

Declarative Interaction Towards Evolutionary User Interface Prototyping

Cristian Bogdan

▶ To cite this version:

Cristian Bogdan. Declarative Interaction Towards Evolutionary User Interface Prototyping. 16th IFIP Conference on Human-Computer Interaction (INTERACT), Sep 2017, Bombay, India. pp.83-90, 10.1007/978-3-319-92081-8_8 . hal-01821422

HAL Id: hal-01821422
https://hal.inria.fr/hal-01821422

Submitted on 22 Jun 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Declarative Interaction Towards Evolutionary User Interface Prototyping

Cristian Bogdan

School of Computer Science and Communication (CSC), KTH Royal Institute of Technology (KTH), 10044 Stockholm, Sweden

cristi@kth.se

Abstract. This paper examines the potential of describing interactive systems in a declarative manner with concepts familiar to both developers and designers. Such declarative interaction descriptions will enable evolutionary prototyping processes. This new type of design and development processes that can emerge with declarative interaction is described along with benefits for human-centred system design. A few challenges are raised for future research in this area.

Keywords: prototyping, declarative, user interface, evolutionary, conflict

1 Introduction

Attempts were made to describe interactive systems declaratively for several decades, for example Model-Based UI Development (MBUID, e.g. [16]). Once a declarative description of an interactive system is available, there are several advantages: the system can be *analyzed* in regard to its associated usability, safety, human error (e.g. [7]) or other aspects, and it can be *transformed*, which includes the generation of code towards the running interactive product. These advantages stem from declarative models being relatively easy to process by computing systems, unlike procedural code

The downside of MBUID approaches is that they are often highly theoretical, aiming to drive the description of the interactive system from a very abstract model (e.g. the Tasks and Concepts level of the CAMELEON framework [5]), which is hard to understand by designers, users or even by developers. Furthermore, MBUID has very little support for user interface *prototyping* [1,15]: when a user interface is generated, it is hard for users and designers to adjust it and iterate with it for improvement according to their needs.

As declarative research approaches like MBUID do not constitute a solution for achieving a more declarative UI design and development practice, the approach we can take is to analyze the current practice and consider where procedural code dominates. For that analysis, one could employ e.g. the *Model-View-Controller* [14] approach to conceptualizing an interactive system. Important parts of the *Model* are

already declarative in current practice, at least in regard to describing how the data is structured, and what methods are available to process it. While the Model methods are most frequently implemented as procedural code, the declarative part of the model is often sufficient for describing the user interface in its other conceptual modules (*View* and *Controller*). Similarly, *View* templates are often described in a declarative manner, in languages such as HTML or variants of XML. In the quest proposed here to find fully or predominantly declarative representations of interactive systems, it is therefore the *Controller* where the procedural code still dominates. Since Controllers describe the "feel" of the user interface, the interaction itself, I use to refer this quest as "*Declarative Interaction*". It is important to note that once a declarative specification of an interactive system exists, a fully testable implementation can be produced. That is, if an online (running) prototype has been made in a declarative manner, with an approach that allows a declarative description of interaction, that prototype can be iterated into the running product. Therefore, declarative interaction can lead to *Evolutionary Prototyping* [1] processes.

In this paper I aim to address the issue of finding such *low-level declarative UI representations*, so that they are understandable for developers and end-users. A subsequent objective is to *support UI prototyping processes*, whereby developers, users, or customers have the freedom to iterate with the interactive prototype. This work is inspired from a large, long-term case study [2,3] that observed users, designers and voluntary developers, who were able to develop, maintain and extend predominantly declarative interactive systems for long periods of time.

2 Potential of Declarative Interaction

2.1 The traditional UI design and development process

In order to illustrate the potential of declarative interaction for achieving a more flexible and human centred User Interface design and development process, it is important to visit the traditional process and emphasize its aspects relevant to this paper: various stages of prototyping and the transition between design and development. The top of Figure 1 illustrates this current process of designing and developing interactive systems). A few (cheap) paper prototypes are produced, and based on designer judgment and user feedback, some ideas continue to the online prototyping phase. After a number of formative iterations, one of the online prototypes is delivered ("Online Delivered" in the figure) for the developers to implement. Variants of the *Model-View-Controller* [14] approach are often used, whereby the non-interactive *Model* (backend, business rules) is manipulated via a user interface programmed in the target technology. The interactive behavior, described by the *Controller*, is, for the largest part, implemented procedurally. The produced interactive system is iterated with, based on feedback from users and designers.

In this traditional approach, designers do not produce artifacts that can be directly used by developers, since most elements of an online (running) interactive system prototype refer to the *View* component, but they need to be rebuilt at the developer

side. There are also power issues in this traditional arrangement: developers are often more powerful than designers and even managers [6], since their work is the most expensive and their work object (software) is resistant to changes. There exist therefore technology 'viscosity' issues: one cannot iterate fast because iterations with non-prototype software are expensive. Usability problems found at later stages are difficult to address.

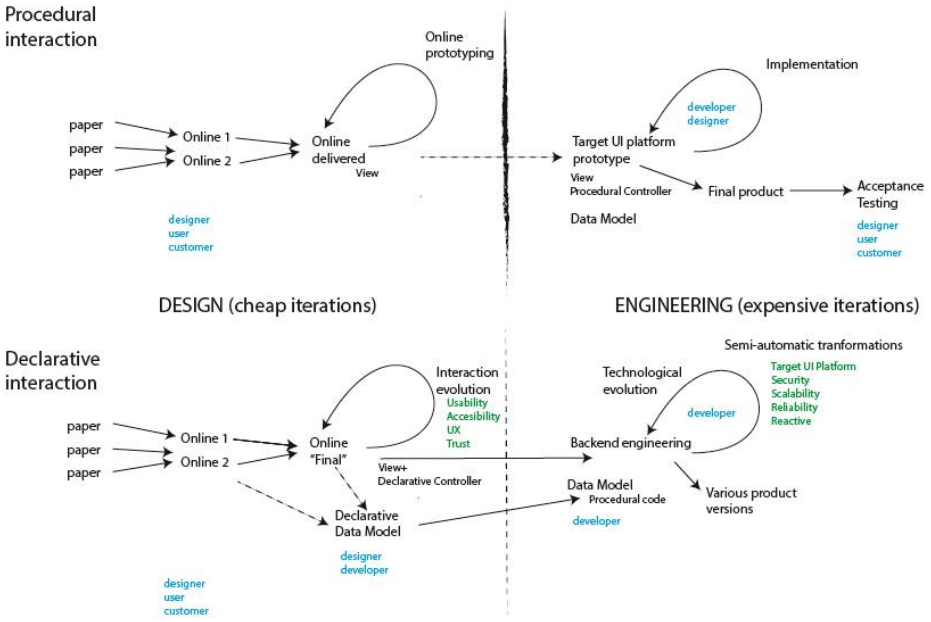

Fig. 1. Traditional interactive system design and development process (top) and the new process facilitated by Declarative Interaction (bottom).

2.2 Example of a declarative User Interface approach

One approach to defining a declarative interaction (*Controller*) takes advantage of declarative constraints specified for the *Model-View* data correspondence (binding) and declares on which UI events these constraints should be updated. Figure 2 depicts a fully declarative user interface, where two *Views* of a factory production Lines containing production Tasks (diagram view at the top and table view at the bottom) show data from the same *Model*. The *Views* are connected to the *Model* by way of declarative SQL-like queries (studied with designers in [3]) shown in blue.

A simple declarative interaction can be illustrated in regard to the user editing in textboxes. For example the designer can choose to update e.g. the task customer name (the table left column) at each user keystroke, and the task duration (the table middle column) only when the user presses Enter or Tab. The declarative controller consists of specifying on which UI event the *View* should be synchronized with the *Model*,

because the widget relation to the model is already set, similar to the drag-and-drop case.

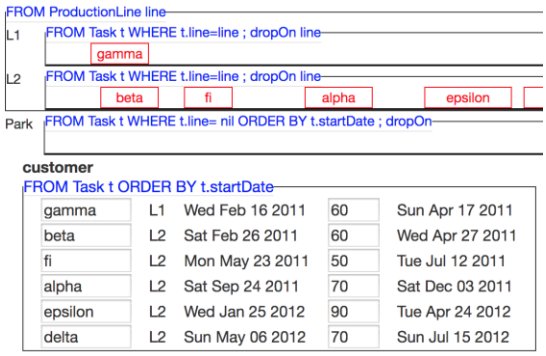

Fig. 2. Fully declarative user interface featuring advanced interaction like drag and drop

A more advanced example of declaring interaction is drag and drop, described by its data semantics, for example a Task object can be dropped on a Line object. This information is sufficient to generate a functioning UI, because the meaning of the drop action is already governed by the *View-Model* connection declared for the respective container widget representing the Line. The widget representing the Line in turn contains widgets that represent Tasks. Therefore, in order to define drag and drop interaction declaratively, it is sufficient for the designer (possibly with developer help) to declare what *Model* objects are being dragged and on which object can they be dropped.

It is interesting to reflect that the declarative annotations that achieve the interaction exemplified above require very little code comparing to the procedural controller needed for achieving similar interaction. If we take the drag and drop example, such a procedural controller would need to define a drag and/or drop handler for each involved widget. Such a handler should do a translation between the graphical domain (widgets) to the abstract domain (*Model* objects), and then invoke the appropriate Model methods to implement the drag and drop semantics.

2.3 Towards a new, evolutionary UI design and development process based on declarative interaction

Based on such declarative controller definition techniques, the alternative proposed here to the traditional UI design and development process is the Declarative Interaction process, illustrated at the bottom of Figure 1 along with its potential emerging evolutions. Declarative interaction was inspired to the author through field observations and own experience [2,3]. Early versions of declarative controllers were demonstrated in [3] and refined in [9,11,12,13]. We have also shown a combination with MBUID approaches in [10].

The specific aspect of this process is that the designer, with input from users and customers, can work not just on the *View*, but also on the declarative *Controller*, and on the declarative parts of the *Model* as they are represented as annotations of the *View* on which the designer usually works. The developer plays a role in this process, by helping in structuring a convenient data *Model*, and possibly also to help express more complex interaction. However, the designer drives the iterations, changing the UI look and feel (*View* and *Controller*) while the developer only helps when relevant aspects of the data Model require adjustment. Early on, or in more advanced phases, online prototypes can be directly made using the target UI technology.

Once a final UI is decided upon, it can be tested and validated with users in regard to its usability and acceptance. After that engineers can take over the prototype and optimize it for Computer Science concerns (e.g. Security, Reliability, Scalability) using semi-automatic tools. Automatic processing (similar to model transformations) and analysis are possible thanks to the declarative nature of the interactive system representation. Since most of the design side uses such declarative representations, the results of the design phase are *transferable* to the engineering side, thus achieving *evolutionary prototyping*. In the engineering side, procedural code can be added. It is interesting to note that in this case developers work mostly on the system backend, which is why their work is termed “backend engineering”.

3 Declarative Interaction and Conflicting UI properties

Design is a balancing act, a suite of tradeoffs that are made along the way between the needs and desires of various users and the institutions they may represent. Therefore conflicting UI design concerns are bound to occur. One way to address such conflicting concerns is the Participatory Design approach [8] of keeping the users involved at all stages of design, therefor ensuring that various conflicting qualities that users, designers and developers require are balanced in an acceptable way.

Declarative Interaction supports the resolution of UI requirement conflicts by supporting Participatory Design through (1) facilitating equal-footing communication between users, designers and developers and (2) encouraging iterations until the late stages of the product design and implementation, thanks to its Evolutionary Prototyping nature.

Even in situations where Participatory Design is not suitable, the balanced power relation facilitated by Declarative Interaction between designers and developers is likely to achieve, through iteration, a good balance between the interactive system properties championed by designers and those guarded by developers. Especially the Interaction Evolution (Figure 1 left) iterations are also accessible to managers, letting them bring their own concerns. Therefore, *iteration* and the *balance of power* are the general process qualities that allow Declarative Interaction to support the resolution of conflicting UI design concerns.

4 A Case of Declarative Interaction

A European Non-Governmental Organization (NGO) with almost 100 locations developed their own systems for over 20 years: member database, document archives, summer course participant selection and management, virtual job fair, etc. All systems are tailor-made for the NGO rules and needs, ensuring greater user understanding and usability compared to general-purpose systems, if such systems are available at all. Users of such systems are 1000s of members and students of participant universities, creating 10000s of new data objects per year.

A major role in this success is played by the Makumba framework [2,3]. It was designed with learning in mind, so that members have to learn two small declarative programming languages (a SQL variant for data retrieval, and HTML for data formatting). More advanced members can continue their learning path and "career" within the NGO by using Java for more complex application logic. Furthermore, a few production Makumba systems exist that use declarative SQL code for most of their application logic, including authentication and authorization, leaving just a few functions to be implemented in procedural Java code.

Reflecting on this long process, I believe that much of the success of Makumba in NGOs is due to the declarative nature of its languages. Declarative code is often small: if a declarative language suitable for a specific programming problem exists, the code will typically be more compact than the procedural correspondent. For an NGO this means less code to track and maintain. Declarative code is often intuitive to read, reducing the initial threshold that junior NGO volunteers have to face before they can contribute with code of their own. Once one makes the code work during development, declarative code is reliable to run, reducing the NGO maintenance costs. Because declarative code can be analyzed and transformed into other representations or technologies, it reduces technology lock-in for the NGO.

Another Makumba success factor stems from its facilitation of evolutionary prototyping, being therefore an early incarnation of Declarative Interaction. Systems are typically prototyped in HTML or directly in Makumba first, and are iteratively refined to become the production system. Systems are often prototyped starting from the user interface (and much of the system code is the user interface), which is intuitive and motivating for the developers, as they can work directly with the artifact that their fellow NGO members will use.

The first generation of Makumba technologies, based on Java Server Pages (JSP) and pre-Web 2.0 user interaction, is by now outdated. Kis [11,12,13] has explored the possibilities of combining data from a multitude of APIs into one interactive application, rather than data from a single relational database, thus using data as a "prototyping material". This has resulted in Endev [12], a Javascript framework that is, however, not production-ready. One major reason is that, unlike Makumba, Endev does not optimize the number of queries sent to its data sources (a very complex problem when there are multiple data sources).

The current approach in modernizing Makumba recognizes that most organizations have one single main data source, and that the Makumba approach of binding data to elements of a HTML user interface has been taken by many other technologies such

as AngularJS[1]. However, Angular still requires a lot of procedural Javascript code to be written, which breaks the principle of declarative development. Therefore the current approach is to develop 'plugins' for Angular and other suitable technologies to replace the current Makumba JSP layer, while keeping the declarative engines of Makumba: optimal SQL query combination, SQL query inlining for code re-use, authentication and authorization, etc. This approach is well under way and since Angular does much of the data binding that Makumba-JSP did, the resulting Angular 'plugin' is quite small and easy to maintain.

While the problem of Declarative Interaction has not yet been fully explored with Makumba, a number of experimental prototypes exist that allow fully declarative description of complex interaction such as Drag and drop, or simpler interaction such as form fill-in with UI update as the user types. One of the next steps considered is to describe declaratively various UI design pattern libraries (e.g. [4]) or other exemplary systems.

5 Declarative Interaction: Challenges for the future

There are many ways to achieve declarative interaction, the Makumba approach (using declarative queries to connect the UI to the data model) is just one. *Exploring alternative declarative interaction approaches* is thus one important future challenge.

Is any user interaction possible to describe declaratively based on simple abstractions, familiar to designers and users? *Addressing declaratively a wide palette of interaction patterns* is probably the biggest challenge faces by researchers in the field.

While the impact on the human centered software engineering processes has already been considered, *assessing further the impact on process* is another important challenge. This includes *considering further ways to empower the designers and users,* but also to *improve their communication with developers.*

Most of this work was conceptualized within the classic Model-View-Controller paradigm. MVC is known to be sensitive to cross-cutting concerns, and an investigation of declarative interaction in relation to such concerns is therefore important. Alternatively, future work can consider other UI development approaches for declarative interaction.

References

1. Beaudouin-Lafon, M. and Mackay, W., 2002. Prototyping Tools and Techniques. The Human Computer Interaction Handbook. J.A. Jacko and A. Sears, eds., p. 1006–1031.
2. Bogdan, C., 2003. IT Design for Amateur Communities. PhD dissertation. KTH Royal Institute of Technology
3. Bogdan, C. and Mayer, R., 2009. Makumba: the Role of Technology or the Sustainability of Amateur Programming Practice and Community. Proceedings of the 4th international

[1] https://angularjs.org

conference on Communities and technologies (C&T). New York, New York, USA: ACM Press, p. 205–214.
4. Boxes and Arrows, Implementing a Pattern Library in the Real World: A Yahoo! Case Study, http://boxesandarrows.com/implementing-a-pattern-library-in-the-real-world-a-yahoo-case-study, accessed August 2017
5. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J., 2003. A Unifying Reference Framework for multi-target user interfaces. Interacting with Computers, Vol. 15, No. 3, p. 289–308
6. Cooper, A. 1999. The inmates are running the asylum, Sams Publishing
7. Fahssi, R., Célia Martinie, Philippe A. Palanque, 2015. Enhanced Task Modelling for Systematic Identification and Explicit Representation of Human Errors. INTERACT 2015 192-212
8. Greenbaum, J. & Kyng, M. (Eds.) Design at Work, pp. 169 – 196. Hillsdale, New Jersey: Laurence Erlbaum Associates.
9. Kis, F. and Bogdan, C., 2013. Lightweight Low-Level Query-Centric User Interface Modeling. 2013 46th Hawaii International Conference on System Sciences (HICSS). Washington, DC, USA: IEEE, p. 440–449.
10. Kis, F., Bogdan, C., Kaindl, H., and Falb, J., 2014. Towards Fully Declarative High-Level Interaction Models: An Approach Facilitating Automated GUI Generation. 2014 47th Hawaii International Conference on System Sciences (HICSS). Washington, DC, USA: IEEE, p. 412–421.
11. Kis, F. and Bogdan, C., 2015. Generating Interactive Prototypes from Query Annotated Discourse Models. I-Com. Journal of Interactive Media, Vol. 14, No. 3, p. 205–219.
12. Kis, F. and Bogdan, C., 2016. Declarative Setup-free Web Application Prototyping Combining Local and Cloud Datastores. 2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE Computer Society.
13. Kis, F. 2016. Prototyping with data. Opportunistic Development of Data-Driven Interactive Applications. PhD thesis. KTH Royal Institute of Technology 2016
14. Krasner, G.E. and Pope, S.T., 1988. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of object oriented programming, Vol. 1, No. 3, p. 26–49.
15. Lim, Y.-K., Stolterman, E., and Tenenberg, J., 2008. The anatomy of prototypes. ACM Transactions on Computer-Human Interaction, Vol. 15, No. 2, p. 1–27.
16. Patternó, F., 2000. Model-based design and evaluation of interactive applications, Springer-Verlag