Consistent Runtime Adaptation of User Interfaces

Anthony Anjorin
Paderborn University,
Germany
anthony.anjorin@ubp.de

Enes Yigitbas
Paderborn University,
Germany
enes@mail.upb.de

Hermann Kaindl
TU Wien,
Austria
kaindl@ict.tuwien.ac.at

Abstract

Modern *User Interfaces* (UIs) are increasingly expected to be *plastic*, in the sense that they retain a constant level of usability, even when subjected to context (platform, user, and environment) changes at runtime. Plasticity is often achieved by specifying suitable *runtime adaptations* of the UI as a response to changing contexts. While there exist numerous approaches to enabling dynamic UI adaptation at runtime, we argue that a series of consistency-related challenges arise during the adaptation process, which have not yet been adequately addressed. In this paper, therefore, we suggest a useful definition of *consistency* in this context, identify some important and open consistency-related challenges, and highlight solution strategies inspired by results and insights from research on *bidirectional transformations*.

1 Introduction and Motivation

User Interface (UI) development requires coping with a constantly increasing level of complexity. One of the reasons for this is the ubiquity of *mobile* UI platforms. Users running UIs on their mobile devices now expect *plasticity* [CCT01], i.e., a constant level of usability, even when experiencing dynamic changes in their environment (brightness, loudness). UIs (especially web-based UIs) are also often used by multiple, different people (age, preferred language), who all expect to experience a comparable level of usability. While such *context* changes can theoretically be handled by (i) developing multiple UIs at design time, (ii) monitoring context changes at runtime, and (iii) responding to changes by choosing the best UI variant, this naïve strategy typically suffers from a combinatorial explosion in practice. For this reason, viable approaches instead support adaptation at *runtime*, controlled with *adaptation rules* [YSSE17]. These approaches focus on runtime adaptation in the context of UI development; we argue that introducing an additional artefact (adaptation rules) gives rise to a series of consistency-related challenges, which have not yet been adequately addressed. In this paper, we provide a characterisation of UI runtime adaptation as a consistency problem, identify a series of consistency management tasks, and propose potential solution strategies inspired by research on *bidirectional transformations* [CFH+09].

Our running example is based on an adaptive e-mail client, which was developed at Paderborn University to perform an empirical experiment on better understanding how users react to automated runtime adaptation. Figure 1 depicts an illustrative sequence of context changes and how the UI adapts to the changed context in each case. Each state is a pair of the UI, depicted as a screenshot of the e-mail app, and the current context as experienced by the user. For our simplified example, the context is reduced to three components: (i) if the

In: J. Cheney, H-S. Ko (eds.): Proceedings of the Eighth International Workshop on Bidirectional Transformations (Bx 2019), Philadelphia, PA, USA, June 4, 2019, published at http://meilu.jpshuntong.com/url- 4 4 0 a2f2f 5 5 22d 2e f 2

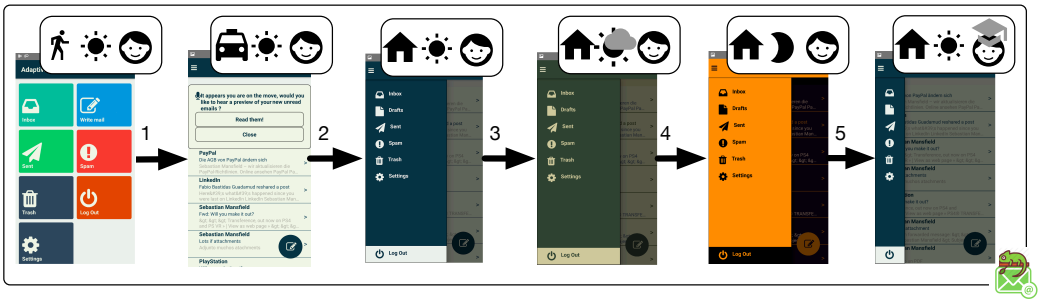

Figure 1: Consistent Context and UI Changes

user is on the move, in a moving vehicle, or immobile (and probably at home), (ii) if the brightness level is high (sunny), low (cloudy), or very low (night-time), and finally, (iii) if the user is a novice or experienced user, based on a threshold value of usage time. The first state (leftmost in Fig. 1) represents a novice user on the move and experiencing high brightness levels. The corresponding UI uses a grid layout to simplify haptic interaction. In response to a context change (depicted in Fig. 1 as labelled arrows – in this case with Label 1) leading to a state where the user is now in a moving vehicle, the UI switches its modality to audio-based interaction, offering to read new e-mails aloud and enabling control of the app via audio commands. When the user is immobile for some time (and can be assumed to be seated in a building – see Label 2), the UI responds by reverting to standard haptic-based modality and additionally uses a list of icons instead of a grid for more efficient screen space usage. The next two context changes (Labels 3 and 4), represent changes in brightness level to low brightness and night-time. The UI responds to low brightness levels by dimming the screen and using sepia tones instead of white/black, and to night-time by inverting the colour scheme. The final context change (Label 5) is triggered when the user passes a certain usage-time threshold. The UI assumes that the user must now be accustomed enough to the icons and saves screen space by removing the explanatory labels for each icon.

With the increase in interest in the development and design of cyber-physical systems, we also mention another possible scenario: In a *smart home environment*, an alternative strategy to address low brightness levels (Label 3), could be to actively modify the context. In this case, for example, the smart home could be requested to increase the brightness levels of the lamps in the current room. A further example for context manipulation could be reacting to a sharp increase in mistakes and corrections (repeated text deletion) of the user. This "frustration level" could be possibly addressed again by increasing brightness levels in the room, by playing soothing music, or by opening the windows to let in fresh air. The point here is that there are cases where adjusting the context of the user in response to changes in the UI could be a desirable strategy for consistency restoration.

2 Consistent Runtime UI Adaptation: Schematic Overview and Challenges

A schematic overview of a typical solution architecture (based on [YSSE17]) that supports runtime UI adaptation is depicted in Fig. 2. We distinguish two main layers: a *modelling layer* that is technology independent, and a *system layer* representing a particular choice of technologies making up a suitable execution platform. A *model space* comprises all possible models of a certain "type" (context, UI), as well as all possible changes (called *deltas*) leading from one model to another. Model spaces can be specified using metamodels and constraints, graph grammars, or some other dedicated modelling language. At the modelling layer, we identify a *context model space*, a *UI model space*, and collection of correspondence models (referred to as *corrs* in the following) that capture the semantic overlap between context and UI models (which context model elements correspond to which UI model elements). Given all possible *triples* consisting of a context and UI model connected by a corr, *adaptation logic* provided by an adaptation engineer is used to identify a *consistent* subset of all possible triples. A triple of context, UI, and corr is thus consistent if and only if it is a member of this consistent subset determined (in some way) by the adaptation logic. Adaptation logic can be expressed as a set of constraints, a set of rules, or a program in some suitable domain-specific language.

Context models represent (a simplification of) the actual context experienced at runtime. At the system layer, this context has to be observed in some manner. For our running example, the specification of the context model space (a context metamodel) is used to generate required *sensor services* for the Android platform. These sensor services not only observe the context, but also notify a *synchroniser* when the context changes. The synchroniser is generated from (or configured with) the specified adaptation logic. For our running example, the synchroniser

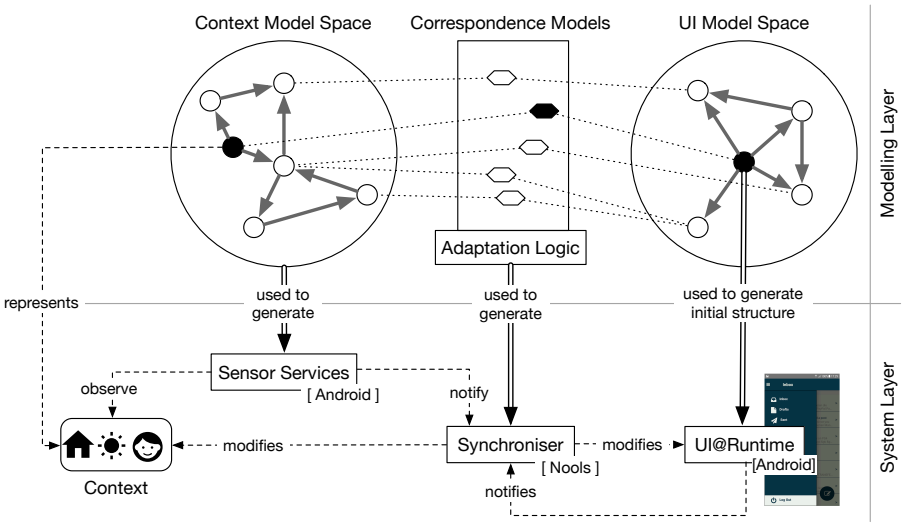

Figure 2: Schematic Overview of Solution Architecture

is realised using *Nools*,[1] a Rete-based rule engine for Javascript. The responsibility of the synchroniser is to react to context and UI changes and restore consistency (if necessary) by modifying either the context (if this is possible) or the UI (or even both). Although it is theoretically possible to have solutions where the UI can be arbitrarily changed at runtime, realistic systems will probably use well-known UI frameworks such as what is offered by the Android or iOS platforms, or web-based UI frameworks such as Angular/AngularJS. The typical approach will be for a UI designer to specify the *initial structure* of the UI on the model level, and then to generate code from this UI model representing the basic structure of the UI. Although some changes (layout, navigation, colours) will be supported, the UI will typically never radically diverge from this initial structure. Creating, e.g., completely new UI widgets on-the-fly is intriguing, but does not reflect the current practice. In the following, we identify and discuss three main groups of consistency-related challenges of runtime UI adaptation:

(C1) Conformance of the System Layer: Especially for an iterative, incremental development of the involved metamodels, adaptation logic, and the required generators, it cannot be assumed that the generated system (services, UI and synchroniser) always behaves in *conformance* to the adaptation logic specified on the modelling layer. Automating such a conformance check requires a trusted execution engine on the modelling layer, and a means of reverse engineering models representing the current state of the running system.

(C2) Compatibility of the Initial Structure of the UI and Adaptation Logic: Another check is if the starting state of the UI, chosen by a UI designer, is still compatible with the adaptation logic, specified by an adaptation engineer. Even if both artefacts are developed by the same person, improvements to the fixed parts of the UI can still violate assumptions made in the adaptation logic.

(C3) Checking for Desirable Properties of the Adaptation Logic: Finally, a series of model analyses can be performed on the adaptation logic. (i) Under which circumstances can the adaptation logic lead to non-deterministic runtime behaviour? While this can be desirable, there are also cases where it is clearly not. (ii) Do adaptations always terminate? This is especially relevant when the context can also be modified. (iii) Are there "dead" parts of the adaptation logic that can never be applicable for a chosen initial UI model?

3 Limitations of Existing Approaches to Runtime UI Adaptation

In the following, we briefly describe the main groups of existing approaches to runtime UI adaptation and discuss their limitations regarding *consistency* management and our identified challenges.

A widespread technique to support UI plasticity is *rule-based* UI adaptation [MPAA16, YSSE17]. The basic idea is to express the adaptation logic as a set of rules characterising how the UI is to be changed as a reaction to the changing context of use. In most cases, rule-based UI adaptation approaches rely on a *rule engine* such as Nools[2] or Drools[3] to implement the synchroniser. To the best of our knowledge, existing rule-based approaches

[1] `http://noolsjs.com/`
[2] http://noolsjs.com/
[3] https://www.drools.org/

do not address challenges (C1) or (C2), and only partially address (C3) by annotating adaptation rules with priority levels used to resolve conflicts between adaptation rules. Priorities, however, only shift the responsibility to the developer and the manual task of determining the correct set of priorities for an evolving rule set is an error-prone process that has to be repeated from scratch for every change.

In contrast to rule-based approaches, *optimisation-based* approaches [Aki14, BMB+11] treat the runtime adaptation of UIs as an optimisation problem. The basic idea is to formulate an objective function that characterises the consistency of the UI and context models (e.g., via constraints in form of a cost function [Aki14] or properties [BMB+11]). Inconsistencies are then resolved by optimising this objective function, e.g., using genetic algorithms [BMB+11]. While optimisation-based approaches provide a better solution for (C3) than rule-based approaches, it is arguable if detailed analyses can be replaced by black-box, often unconfigurable optimisation. As with rule-based approaches, (C1) and (C2) remain completely unaddressed.

Another group of approaches (e.g., [Tro14]) focuses on the modelling layer and makes a *models at runtime* assumption, i.e., that there is no (generated) system layer, eliminating (C1). Trollmann [Tro14] focuses on supporting (C3), providing a formal approach based on *graph diagrams* for a rule-based specification of the adaptation logic. Analyses for conflict detection and other inconsistencies are presented, potentially addressing (C2) as well (at least on the modelling layer). While the approach of Trollmann essentially applies bx[4] to UI runtime adaptation, it is also (at least for the foreseeable future) of limited applicability as it ignores the conformance of the system layer and thus all pragmatic, generative approaches to UI development (C1). Finally, there are also hybrid approaches [RTKP18] that strive to combine approaches to design and runtime adaptation.

4 Applying Bx to Runtime UI Adaptation: Potential and Open Challenges

As partly demonstrated by Trollmann [Tro14], a bx language can be used to implement the adaptation logic required for supporting runtime UI adaptation (see Fig. 2). This is advantageous for the following reasons:

1. Bx languages are designed to address multiple consistency management scenarios including consistency checking, forward and backward transformations, and incremental synchronisation, using a *single program* (in this case the adaptation logic). In combination with a reverse engineering approach, (C1) can thus be addressed by checking if a pair of context and UI models (representing the current context and runtime UI on the modelling layer) is consistent. (C2) can be addressed by backward transforming the initial UI model, i.e., attempting to extend it to a consistent triple. If this fails, then the initial UI model is not in the domain of the synchroniser and, in this sense, contradicts the adaptation logic.

2. Concerning (C3), bx languages are designed to obey a certain set of bx laws (depending on the language) with the goal of guaranteeing *desirable* behaviour. Some rule-based bx languages such as Triple Graph Grammars (TGGs) [Sch94] also provide static analyses techniques that can be used to guarantee additional properties such as determinism or confluence.

There are, however, also open challenges for bx posed by the application domain of runtime UI adaptation:

1. In general, there has been a primary focus of most bx approaches on model synchronisation. While this is certainly important, (C1) indicates that conformance checking of a generated system is an equally crucial task in practice. *Scalable* support for consistency checking is, however, still to be improved for most bx languages [Leb16], even for TGGs.

2. Bx languages should be able to *tolerate* inconsistencies, as users in a UI runtime adaptation setting can *refuse* certain adaptations and might prefer to continue using the UI without restoring consistency. This has already been identified as an open bx challenge by Stevens [Ste14]; runtime UI adaptation provides yet another practical setting where this is important.

3. Finally, a development framework for setting up and realising complex architectures such as illustrated by our schematic overview (Fig. 2) is missing in practice. The choice of bx language, as well as generators, adapters, and a suitable modelling framework should be kept flexible and exchangeable through the usage of standard component interfaces. The current work on a bx benchmarking framework can be viewed as preliminary work in this direction [ADJ+17].

[4] Graph diagrams are a generalisation of Triple Graph Grammars (TGGs) to multiple models [TA15].

References

[ADJ+17] Anthony Anjorin, Zinovy Diskin, Frédéric Jouault, Hsiang-Shang Ko, Erhan Leblebici, and Bernhard Westfechtel. BenchmarX Reloaded: A Practical Benchmark Framework for Bidirectional Transformations. In *Proceedings of the 6th International Workshop on Bidirectional Transformations, BX@ETAPS 2017*, pages 15–30, 2017.

[Aki14] Pierre A. Akiki. *Engineering Adaptive Model-Driven User Interfaces for Enterprise Applications*. PhD thesis, Open University, UK, 2014.

[BMB+11] Arnaud Blouin, Brice Morin, Olivier Beaudoux, Grégory Nain, Patrick Albers, and Jean-Marc Jézéquel. Combining Aspect-Oriented mMdeling with Property-based Reasoning to Improve User Interface Adaptation. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing System, EICS 2011, Pisa, Italy, June 13-16, 2011*, pages 85–94, 2011.

[CCT01] Gaëlle Calvary, Joëlle Coutaz, and David Thevenin. A Unifying Reference Framework for the Development of Plastic User Interfaces. In *Engineering for Human-Computer Interaction, 8th IFIP International Conference, EHCI 2001, Toronto, Canada, May 11-13, 2001, Revised Papers*, pages 173–192, 2001.

[CFH+09] Krzysztof Czarnecki, John Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James Terwilliger. Bidirectional Transformations: A Cross-Discipline Perspective. In *ICMT 2009*, volume 5563 of *LNCS*, pages 260–283. Springer, 2009.

[Leb16] Erhan Leblebici. Towards a Graph Grammar-Based Approach to Inter-Model Consistency Checks with Traceability Support. In Anthony Anjorin and Jeremy Gibbons, editors, *Proceedings of the 5th International Workshop on Bidirectional Transformations, Bx 2016, co-located with The European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 8, 2016.*, volume 1571 of *CEUR Workshop Proceedings*, pages 35–39. CEUR-WS.org, 2016.

[MPAA16] Raúl Miñón, Fabio Paternò, Myriam Arrue, and Julio Abascal. Integrating Adaptation Rules for People with Special Needs in Model-based UI Development Process. *Universal Access in the Information Society*, 15(1):153–168, 2016.

[RTKP18] Thomas Rathfux, Jasmin Thöner, Hermann Kaindl, and Roman Popp. Combining design-time generation of web-pages with responsive design for improving low-vision accessibility. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2018, Paris, France, June 19-22, 2018*, pages 10:1–10:7, 2018.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In Ernst W. Mayr, Gunther Schmidt, and Gottfried Tinhofer, editors, *Graph-Theoretic Concepts in Computer Science, 20th International Workshop, WG '94, Herrsching, Germany, June 16-18, 1994, Proceedings*, volume 903 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 1994.

[Ste14] Perdita Stevens. Bidirectionally Tolerating Inconsistency: Partial Transformations. In Stefania Gnesi and Arend Rensink, editors, *Proceedings of 17th Int. Conf. on Fundamental Approaches to Software Engineering, FASE 2014*, volume 8411 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2014.

[TA15] Frank Trollmann and Sahin Albayrak. Extending Model to Model Transformation Results from Triple Graph Grammars to Multiple Models. In *Theory and Practice of Model Transformations - 8th International Conference, ICMT 2015, Held as Part of STAF 2015, L'Aquila, Italy, July 20-21, 2015. Proceedings*, pages 214–229, 2015.

[Tro14] Frank Trollmann. *Detecting Adaptation Conflicts at Run Time using Models@run.time*. PhD thesis, Technical University Berlin, Germany, 2014.

[YSSE17] Enes Yigitbas, Hagen Stahl, Stefan Sauer, and Gregor Engels. Self-adaptive UIs: Integrated Model-Driven Development of UIs and Their Adaptations. In *Modelling Foundations and Applications - 13th European Conference, ECMFA 2017, Held as Part of STAF 2017, Marburg, Germany, July 19-20, 2017, Proceedings*, pages 126–141, 2017.