Model-based engineering of user interfaces to support cognitive load estimation in automotive applications

Giorgio Brajnik
Computer Science School
University of Manchester, UK
and
Dipartimento di Matematica e Informatica
Università di Udine, Italy
brajnik@uniud.it

Simon Harper
Computer Science School
University of Manchester, UK
simon.harper@manchester.ac.uk

ABSTRACT

In this brief position paper we argue that model-driven engineering practices could be adopted in the design and evaluation of automotive UI. We illustrate how UML state machine models can be used for automatic generation of executable prototypes of the UI and for computing graph-theoretic metrics that could bear upon cognitive load.

Categories and Subject Descriptors

H.1.2 [**Information Systems Applications**]: Models and Principles—*Human factors*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Prototyping, user-centered design, Graphical user interfaces*; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; D.2.m [**Software Engineering**]: Miscellaneous—*rapid prototyping*; D.2.2 [**Software Engineering**]: State diagrams—*UML state machines, statecharts*

General Terms

Software engineering, model-based user interfaces, model-driven engineering, interaction design, usability

1. INTRODUCTION

Bad usability critically affects embedded systems such as those on board of cars for two reasons. On the one hand, systems are costly to replace (higher costs in recalling, in disseminating, in reinstalling new versions); on the other hand, their context of use is critical, as bad usability in such user interfaces often leads to low safety.

Although advanced UIs can be conceived that reduce drivers' cognitive load, drivers will still have to interact with them. For example, [5] highlight several factors that can lead to drivers' distractions when using a GPS navigator and suggest remedies such as: while driving in familiar areas the level of detail of instructions provided by the navigator should be reduced, whereas it should increase with high traffic density, or bad weather, or when driving in unknown areas. Even with such adaptive navigators that optimize their output, drivers might need to interact with the navigator while driving: to turn off or on its voice, to change view on the map, to get an overview of the suggested route, to see the estimated time or distance to arrival, to locate some intermediate destination on the map, etc. Each of these use cases might require an attention switch that could be fatal, especially if the user interface requires observation and concentration.

To produce designs of user interface that are valid, there is no substitute of iteratively carrying out usability investigations. However, designers use static prototypes (sketches, storyboards, page mockups) or minimally interactive ones (clickable PDFs or slides), which lack most details regarding the dynamics of the user interface. In fact, because prototypes are manually built, not all the data nor all user actions are implemented and can be therefore investigated. As argued by [4], mixed-fidelity prototypes are often needed in order to perform a usability investigation that can reveal major usability problems. Almost always, because of cost, manually built prototypes show poor levels of fidelity in terms of richness of data and of interactivity, reducing the interactions that could be considered during the investigation. And this is despite user actions being an important focal point of designers ([1] offers an ample discussion) and crucial for usability investigations (cfr. "interaction cycle" by [6], goal-action-effect "triangle" [3]).

A different line of attack is based on using metrics. Over the years, metrics have been developed that can be applied to user interfaces with the aim of characterizing some property that bears upon usability. For example, [9] illustrate a number of graph-related metrics that can be used to highlight usability aspects of user interfaces. Using a transition network where states represent screens of the user interface and transitions represent user actions, these authors show that metrics that measure centrality in a graph, such as "betweenness" of states, do bear upon usability defects of devices like hospital infusion pumps, and that findings deriving from such metrics could be used to identify severe problems and hence to prompt for design modifications.

2. MODEL-DRIVEN ENGINEERING

The context of this research is model-driven engineering (MDE) methodologies [7, 8] for developing user interfaces. In general these approaches provide means for using models to direct the course of understanding design, construction, deployment, operation, main-

AutomotiveUI'13, October 27-30, Eindhoven, The Netherlands.
Adjunct Proceedings.

tenance and modification of software systems. They combine domain-specific modeling languages, that formalize aspects of the system that are specific to particular domains, with transformation engines and generators that are used to synthesize several types of artifacts, from source code to test cases, in order to achieve the process known as "correct-by-construction", as opposed to the more frequent "construct-by-correction" approach. CAMELEON is a reference framework for model-based approaches to the development of interactive context-sensitive systems. Development and maintenance of user interfaces occur through a layered architecture encompassing different models that separate the concerns [2], and corresponding transformation rules between models.

A basic tenet of *model-based user interfaces* is the ability to specify models that are expressive enough to explicitly represent the properties that are suitable for a given kind of analysis or processing. In our case, we are interested in finding out usability defects associated to the interaction structure and in computing metrics that are linked to cognitive load.

More specifically, with the UML-IDEA project (UML-based Interaction Design Approach) we use UML state machine models (also known as "statecharts") to represent the dynamics of a user interface, UML class diagrams to represent data manipulated by the user interface, and model annotations to associate data and widgets to states, so that one can automatically generate executable mixed-fidelity prototypes to be used in usability investigations. Furthermore, metrics could be computed on models so that properties of the interaction structure can be found.

Although several suggestions on how to use statecharts for modeling user interfaces are already known, UML-IDEA is the first approach that is based on a "model-to-code" transformation of UML state machines, class diagrams and XML annotations, so that the structure of the UI (containers and widgets) is automatically inferred by the system. This allows the designer to specify as detailed information as deemed appropriate, and still be able to generate executable prototypes. In addition, thanks to the orthogonality of the models, the entire mixed-fidelity prototype space can be easily explored by the designer.

At the moment, UML-IDEA encompasses a UI compiler that is capable of processing full UML state machine models, simplified class diagrams and annotations, and of generating executable prototypes in a HTML5/Javascript platform.

3. EXAMPLE

Consider the two models in figures 1 and 2, of cruise control devices on board of two common cars (left anonymous). It is obvious that one model is more complex than the other one: in 2 the user can "store" the desired speed and later on engage the controller so that the stored speed can be reached. In addition, the controller may automatically switch off when the actual speed of the car stays well beyond the desired speed for a certain time period. Models include only transitions that have some effect; though other actions can be performed (such as breaking when the system is in state `Standby`), they are not modeled as they do not affect the system.

One way to understand how that added complexity manifests itself in terms of usability or cognitive load relies on materializing the design into a prototype. Let's imagine that these are two alternative designs. By interacting with prototypes a designer could figure out what sequences of actions are needed to get to a certain state, if in a given state all available actions are needed and make sense, and if all the different states make sense for such an application. With such prototypes, suitably enriched with look and feel aspects, usability experiments could be run to estimate cognitive load and interferences of interactions with driving performance.

Because the chosen models emphasize interaction structure, models themselves provide no help in identifying usability problems that deal, for example, with affordances of controls or with perception of UI components. However, because of the independence of control, data and look and feel aspects, once these look and feel aspects are defined, it is relatively easy to change the UI logic and even the data and reuse them.

Because the state machine model specifies transitions that are associated to either user actions (*e.g.*, the user tapping on a widget) or autonomous actions (*e.g.*, the cruise controller switching off automatically), possible usability problems dealing with timing between relevant events could also be caught when using the prototypes.

As briefly mentioned above, models could be used also to compute graph-theoretic metrics that might be associated to cognitive load. Besides trivial metrics such as number of states, transitions and concurrent regions, other more complex scores can be used to benchmark a model and even to identify weak spots in a design.

To apply such kind of metrics, the UML state machine model has to be appropriately processed so that hierarchical states, hyper-transitions and concurrent regions are flattened[1]. On the resulting directed multi graph (the *interaction graph*), one could assess the following properties among others:

- *Connectivity*: an interaction graph that is not strongly connected has at least one state that cannot be reached from another one. This might mean a partial dead end for the user, whose consequences depend on the meaning of the isolated state. For example, in both models "S" and "A" the final state `Off` has no outgoing edges, but this is by design, and has no negative consequences on interaction.
- *Hinges and bridges*: these are states and transitions that, if removed, cause the interaction graph to become disconnected. Therefore they are states or actions that users need to be aware of, otherwise a set of system behaviors will not be available. For example, the `press` transition leaving state `standby` in model "A" is such a bridge. If the driver is not aware of such an action, then the whole cruise control system is useless.
- *Diameter*: it is an attribute of the entire graph, and is based on the shortest paths between all pairs of states. The larger it is and the more unbalanced the design is, with some pairs of states being far away. It can be used to gauge the potential complexity of a design. For example, in "A" the diameter is 3, while in "S" it is 2.
- *Centrality*: in general terms, a central state in the interaction graph is a state that is important (*e.g.*, because it can be easily reached from many other states). There are many notions of centrality that can be considered, including *eigenvector*, *pageRank*, *closeness* and *betweenness*. Betweenness of a state reflects the number of times that the state is included in a shortest path between two other states; similarly for a transition. High values of betweenness are associated to states or transitions that have to be passed through often. Table 1 shows betweenness scores for our example.

Assuming that user actions require different levels of accuracy and attention, one could attach weights to transitions and observe the analytic consequences of such assumptions. For example, in model "S" certain actions require more attention than others: a `uplong` action means to pull up the lever and keep it there to continuously accelerate the car; when the lever is left the current speed becomes the reference value for the controller. In the met-

[1]This is a preliminary process that in UML-IDEA is needed also for generating prototypes.

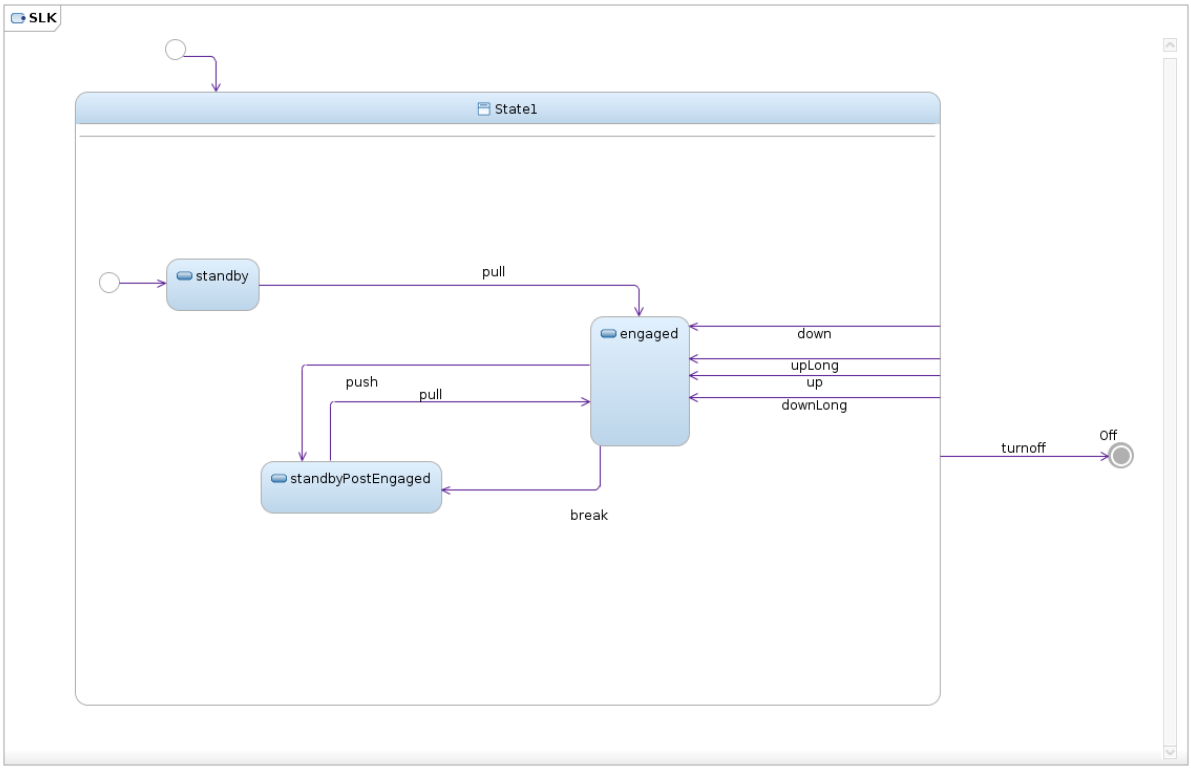

Figure 1: UML state machine model of the cruise control on board of car model "S". Arrows correspond to user actions.

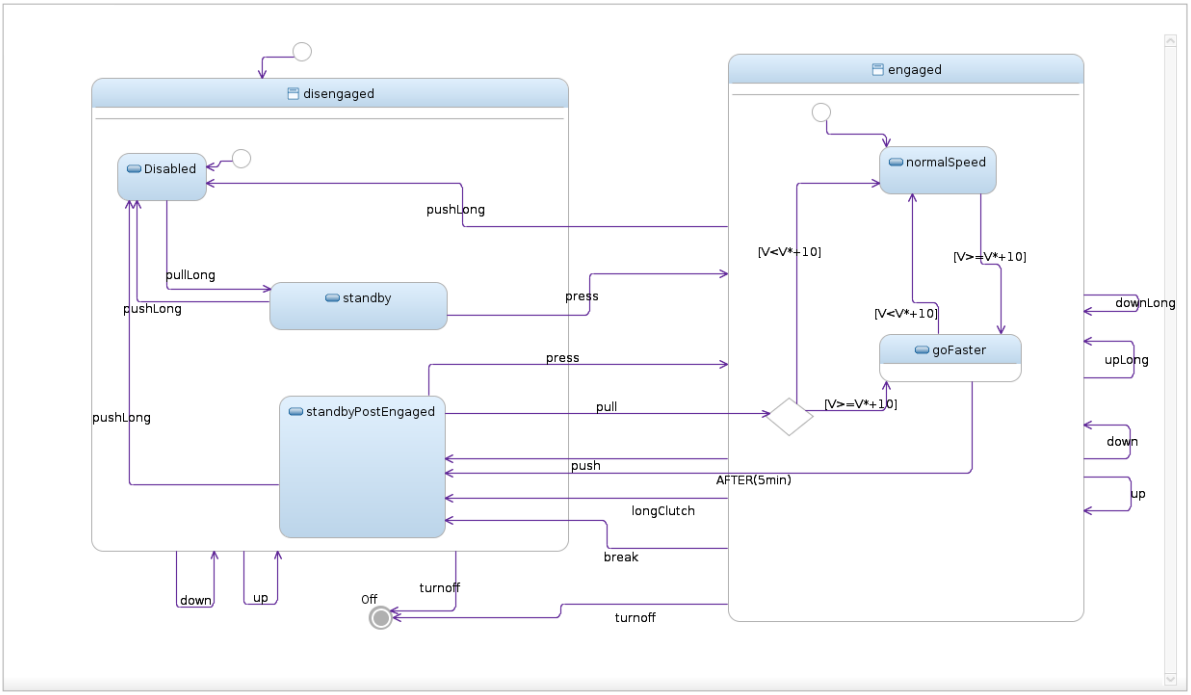

Figure 2: UML state machine model of the cruise control on board of car model "A". Some arrows correspond to user actions, while others are autonomous ones.

	Transition	Source state	Betweenness
	downlong(3)	Standby	0.00
	uplong(3)	StandbyPostEngaged	0.00
	downlong(3)	Engaged	0.00
	uplong(3)	Standby	0.00
	downlong(3)	StandbyPostEngaged	0.00
	uplong(3)	Engaged	0.00
	pull(1)	StandbyPostEngaged	0.33
	up(1)	StandbyPostEngaged	0.33
	down(1)	StandbyPostEngaged	0.33
	pull(1)	Standby	0.67
	up(1)	Standby	0.67
	down(1)	Standby	0.67
	break(1)	Engaged	1.00
	push(1)	Engaged	1.00

	Transition	Source state	Betweenness
	pushLong(3)	GoFaster	0.00
	downLong(3)	Standby	0.00
	upLong(3)	GoFaster	0.00
	downLong(3)	NormalSpeed	0.00
	pushLong(3)	NormalSpeed	0.00
	press(3)	StandbyPostEngaged	0.00
	...	...	0.00
	AFTER(5min)(1)	GoFaster	* 0.83
	break(1)	GoFaster	0.83
	push(1)	GoFaster	0.83
	pull[V<V*-10](1)	StandbyPostEngaged	1.00
	[V<V*-10](1)	GoFaster	* 1.00
	pushLong(1)	Standby	1.00
	pull[V<V*-10](1)	StandbyPostEngaged	1.00
	pull[V>=V*-10](1)	StandbyPostEngaged	1.00
	push(1)	NormalSpeed	2.17
	break(1)	NormalSpeed	2.17
	[V>=V*-10](1)	NormalSpeed	* 3.00
	pushLong(1)	StandbyPostEngaged	4.83
	press(3)	Standby	6.00
	pullLong(1)	Disabled	7.00

Table 1: Centrality of states and some (weighted) transitions in model S (top) and model A (bottom).

rics analysis of the model, we attached a weight of 1 to all actions except for `uplong` and `downlong`, whose weight was set to 3. In other words we are assuming that these two actions are 3 times more difficult.

Because betweenness figures depend on the notion of shortest path in the graph, which in turn is based on the "cost" of followed transitions, the resulting scores depend on such an assumption. For the kind of analysis that we are discussing here, however, choosing any pair of increasing positive values would lead to similar results.

One thing that we can notice in model "S" is that "difficult" actions have a 0 betweenness score, meaning that they are not in the way of the driver who wants to use the controller. And, vice versa, actions that have a relatively high score are simple ones.

Compare these data with the scores obtained from model "A" (Table 1). Notice that several actions with weight 3 have a low betweenness score, which is good. However, the `press` action (to initially engage the controller the driver has to radially press the lever, which requires careful control of the hand movement) has a centrality score of 6, indicating that it is a transition that should be often followed. Transitions marked with "*" are autonomous ones, that do not require a driver action. Notice that the autonomous transition `[V>=V*-10]` (which occurs when the actual speed exceeds the set one by 10 or more km/h) has a relatively high centrality, meaning that it could occur often. Therefore appropriate indicators should be used in the user interface to notify the driver of such a change (such as turning on or off a particular symbol in the dashboard, or changing its color.

4. CONCLUSIONS

We attempted to show that model-driven engineering practices could be beneficial for automotive user interfaces. Such an approach could be adopted to quickly generate running prototypes so that usability of the UI can be assessed. A second usage is in applying graph-theoretic metrics that can be used to benchmark a design, to compare 2 or more designs, to estimate cognitive load associated to interaction, and even to spot possible weak points.

Currently we are working on in-vehicle devices that require user interaction, such as infotainment systems and GPS navigators. We are collecting evidence that this rapid prototyping approach coupled with the ability of using metrics is effective in spotting weak areas of a design and supports informed rapid changes so that a designer can easily explore a large design space. We are also exploring the space of available metrics and validating promising ones.

5. REFERENCES

[1] P. Dourish. *Where the action is: the foundations of embodied interaction*. The MIT Press, 2001.

[2] J. M. C. Fonseca. Model-based ui xg final report. `http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/`, May 2010.

[3] R. Hokyoung and A. Monk. Interaction unit analysis: A new interaction design framework. *Human-Computer Interaction*, 24(4):367–407, 2009.

[4] M. McCurdy, C. Connors, G. Pyrzak, B. Kanefsky, and A. Vera. Breaking the fidelity barrier: an examination of our current characterization of prototypes and an example of a mixed-fidelity success. In *CHI 2006*, pages 1233–1242, New York, NY, 2006. ACM, ACM Press.

[5] D. Münter, A. Kötterizsch, T. Islinger, T. Köhler, C. Wolff, and J. Ziegler. Improving navigation support by taking care of drivers' situational needs. In *Proc. of the 4th Int. Conf. on Automotive User Interfaces and Interactive Vehicular Applications*, pages 131–138. ACM Press, 2012.

[6] D. Norman. *The psychology of everyday things*. Basic Books, 1988.

[7] D. Schmidt. Model-driven engineering. *Computer*, pages 25–31, February 2006.

[8] J. Siegel. Developing in OMG's model-driven architecture. Object Management Group White Paper, Nov. 2001. `http://www.omg.org/cgi-bin/doc?omg/00-11-05.pdf`.

[9] H. Thimbleby and P. Oladimeji. Social network analysis and interactive device design analysis. In *Proc. of Engineering Interactive Computing Systems 2009*, pages 91–100. ACM Press, 2009.