Graphic Toolkit for Adaptive Layouts in In-Vehicle User Interfaces

Renate Häuslschmid
Technische Universität München
Department of Informatics
Boltzmannstr. 3, Garching bei München
renate.haeuslschmid@mytum.de

Klaus Bengler
Technische Universität München
Institute of Ergonomics
Boltzmannstr. 15, Garching bei München
bengler@lfe.mw.tum.de

Cristina Olaverri-Monreal[*]
Technische Universität München
Institute of Ergonomics
Boltzmannstr. 15, Garching
olaverri@lfe.mw.tum.de

ABSTRACT

Currently, the processes used by many car manufacturers to adapt information from the head unit to the intended display platform are outdated and extremely cumbersome. The individual graphic elements are not designed for use in different contexts and many steps must be done manually to achieve a proper in-vehicle information visualization. Additionally, the amount of data that must be maintained is extremely large, resulting in strong restrictions in the variability of appearance and displayed information. An additional challenge is that multiple car brands belong to the same main company with each brand having a separate identity. Therefore, the graphical user interface (GUI) elements require resizing and recomposition which then reflects the brand's heterogeneous characteristics. To overcome these drawbacks we present a software solution to create and edit flexible, in-vehicle GUIs through reusable elements or widgets that adjust their size and composition to their environmental context in a dynamic and automatic manner. We have examined the quality of the tool through validation rules for each step and proposed calculation algorithms as a possible approach for a largely automated evaluation.

Categories and Subject Descriptors

D.2.1 [**Requirements/Specifications**]: [Elicitation methods, Methodologies, Tools]; D.2.2 [**Design Tools and Techniques**]: [Modules and interfaces]; D.2.5 [**Testing and Debugging**]: [Code inspections and walk-throughs.]

General Terms

Algorithms, Design, Human Factors, Verification

[*]Corresponding author

Keywords

Graphic elements, user interfaces, GUIs design

1. INTRODUCTION

Multi-platform applications and services conceived to run in mobile screen terminals in different screen sizes can create challenges for graphical user interface designers [1]. In an automotive context, information is depicted to the driver through visual displays that can be divided into the following three main categories:

1. Instrument panel: primarily provides information on driving speed and number of revolutions;
2. Head unit: offers the driver functions, which are directly required for driving, but which also provide additional information from infotainment or safety-related systems;
3. Head-up display: information related mostly to the navigation system and to warnings from driver assistance systems;

The small space available to display the relatively large amount of data presents an acute challenge, as the driver must be able to see the information immediately while still focusing on the task of driving itself. Additionally, as more and more in-vehicle systems begin to include further applications, such as those found in other mobile environments like smart phones or tablets, the information must be reorganized in digital screens located in new vehicular spaces in addition to the traditional spaces, such as the instrument panel or the center console [2]. Proper in-vehicle information location and visibility in different screen sizes could facilitate driver interaction with device controls, assuring a smoother automobile or information operation and reduction of distraction potential. Therefore, there is a need for a more flexible development of user interfaces (UI) to better support not only the user, but also different types of applications with reusable components that can automatically adapt to different in-vehicle displays.
Currently, processes used by many car manufacturers to adapt information from the head unit to the platform where it is intended to be displayed are outdated and extremely cumbersome. The individual graphic elements are not designed for use in different contexts and many steps must be executed manually, which often leads to errors and delays.

Additionally, the amount of data that must be maintained is extremely large, resulting in strong restrictions in the variability of appearance and information.
An additional challenge is the fact that multiple car brands belong to the same main company. For example, the Volkswagen Group sells automobiles under the following brands Audi, Bentley, Bugatti, Lamborghini, Porsche, SEAT, Škoda and Volkswagen [3]. As each brand has a separate identity, the GUI elements do not only require a resizing that fits in the varied displays, but also a complete recomposition which reflects the brands heterogeneous characteristics. Thus, there is an urgent need for a centralized GUI development that decreases the complexity degree of managing all possible variants for different displays and brand characteristics [4]. Such an approach would help to create user interface objects that are adapted to a certain environment. Consequently, we present in this paper a software solution for creating and editing flexible, in-vehicle graphical user interfaces through reusable elements or widgets. These user interface elements adjust their size and composition to their environmental context in a dynamic and automatic manner, allowing thus for flexible composition of elements within the limited in-vehicle screens.
For the development of user interfaces for head units, the automotive industry currently resorts to graphical mock ups and documentation containing descriptions written mostly in natural language, as well as guidelines and graphical widget toolkits. These documents are then split into hierarchical object structures, object behaviors, logic descriptions and content. The graphics of each head unit variant are then manually defined and documented. As a consequence, composition variations must be done manually and posteriorly analyzed with regard to their effects. This procedure involves not only a tremendous amount of work, but also produces inconsistent output.
To overcome the drawbacks related to the manual procedures, our toolkit is based on automation and failure management in early development phases. The remainder of this paper is organized as follows:
The next section considers related work in the areas of adaptive, flexible user interfaces. Section 3 presents a detailed description of the methodology followed to implement the approach presented in this study. Section 4 reports on the tool evaluation. Finally, Section 5 concludes the paper.

2. RELATED WORK

Several authors have already stated the difficulty of specifying, implementing and testing in-vehicle Human Machine Interfaces (HMI), due to the complexity of their static and dynamic properties. According to [5] these interfaces can include 2000 graphical different views and more than 2500 system states to describe the system behavior. Consequently, the emerging need for more component-based user interfaces development tools has been highlighted in various research works [6]. The authors in [7, 8] presented a system to automatically adjust windows sizes and locations depending on the system environment and to automatically alter the layout of content-filled documents. A further approach to adapt interfaces to small screens through the definition of abstract widgets, deciding the new value in run time, was suggested in [9]. However, these adjustments do not consider a potential recomposition of the GUI elements.
A method to transfer and integrate mobile applications on infotainment system displays was proposed in [10] using HTML as main language for the layout description of the elements and guaranteeing a high graphical quality and reusability. Additional adaptation approaches were presented in [11, 12] based on the idea of zooming or using a user-adaptable hierarchical layout methodology, controlled by additional user actions. These solutions are, however, not applicable in an automotive context, as manipulating in-vehicle devices using this kind of technology would divert the driver's attention from the road.
The reutilization of graphical components to support not only an specific application but also other types of applications has been the focus of several other studies [13, 14]. In this context, a review of current technology related the dynamic compositional adaptation of software has been presented in [15].
A related work that additionally considered small mobile screen sizes was presented in [1]. The authors proposed a method that divided the screen space for components at runtime, taking into consideration the terminal screen size and current user interface components, thus allowing constant information density, context visibility and better suitability for small screens. The user interface elements used a growth potential and an aspect ratio to adjust the expansion of components. On large displays, the GUI was optimized by enlarging the components. However, when the size of the display was small, the information was reduced by minimizing or hiding less important components.
We propose a novel approach to automatically adapt the layout of the information displayed in a head unit to the platform where it is intended to be displayed. Our system is based on algorithms that use independent fixed or variable values or formulas to guarantee maximum flexibility in element size and position, instead of relying on attributes and fixed layout patterns. Our toolkit allows the creation of all head unit views through the composition of standard objects. In a sensitive environment such as a head unit display, information cannot always be reduced. Therefore, our software solution is able to manage the brand and display variants which allows for the creation, verification, and publishing of each variant as a single and independent building block. Additionally, through our approach we guarantee flexibility and scalability through adaptation to changing layout requirements as specified in [16, 17]. We also ensure an independence of the content, layout and logic. This increases flexibility through enormous variation possibilities enabled by a graphic kit.

3. DEVELOPMENT PROCESS

The modular building kit enables the universal use of various modules in several combinations. The tool design process was iterative and user-centered. Consequently, the implemented system contains all necessary features for the user friendly design of the head units GUI elements independently of the technical background of the user.

3.1 Requirements Analysis

In order to define the software requirements for the implementation of the toolkit, we firstly performed an extensive development process and system layout analysis. Next, data related to user groups was collected through qualitative techniques such as observation and interviews in order to determine potential user types. According to the user types, mental model diagrams and use cases were developed, a list of functions and components consequently derived and finally a requirements specification was agreed upon.

3.2 Central Information Display System Analysis

From a functional point of view, a Central Information Display (CID) view consists of GUI elements or widgets that provide information that can be manipulated by the user, such as a window or a text box. From a technical point of view, these widgets consist of layout, content, and a specific logic. A CID view is built through Layout Templates (LT) in the Extensible Markup Language (XML) and, together with data and logic, transferred to widget nodes. At the lowest hierarchic level stand the containers into which the content is inserted. We analyzed the relationship between the interface elements in order to set up the requirements for the tool as detailed below:

- In the view structure, links between an LT and the contained LT through the widget are struck, the child LT is linked. However, the kinship occurs only between the Layout Templates, while the widgets are not expressly concerned. This relationship is already defined during the LT creation, at a time when the content is still completely excluded.

- Kinships are set as properties of the LT and handed on to all variants. If the structure of the LT variants differs with regard to the elements a child LT might be embedded in only a Layout Template's variant. In any case, the embedded version of the LTs must be mapped to the same brand and the same display.

- Kinships are set as properties of the LT and handed on to all variants. If the variants of one Layout Template differ regarding the embedded containers, and a child Layout Template is embedded in these containers, the structure of the view depends on the structure of the Layout Templates. In any case, the embedded version of the LTs must be mapped to the same automobile brand and the same specific display.

- The Layout Template parameters summarize CID layout versions that have common global properties.

- A container specifies defined types such as text, graphic, or widgets which correlate with the planned content and contribute to certain Layout Templates by means of their exact size definition (height and width) and position (absolute or relative spacing). Variants contain all information of the actual composition and calculation of standard layout elements specifically for one or more runtime environments. Variants can differ greatly from one another, for example with regard to embedded containers.

We structured each CID view and its components in a hierarchical manner in order to later subdivide them into singular elements to produce several UI versions. This allowed for the union of the basic containers to each other, which served as the basic elementary toolkit of the editor. Composed by different layout patterns, these containers could build higher level components and finally make an entire presentation similar to a construction kit possible.

This set of elements and their combinations could then be computed in a parameterized manner with different dimensions and distribution algorithms in accordance with the variations, thus allowing for flexible and automatic element adaptation in size and appearance through information specified in different style sheets. As a consequence, the dynamics of base elements and layout managers allow for the dynamics of the entire figure. Changes to individual items or components can be shown through the version management and analyzed due to the hierarchical structure of the kit.
In order to ensure the visual building blocks characteristics of the widget, we determined the Layout Template's parameters and the specific combinations, achieving thus all views containing this LT. To accomplish a user-friendly environment, all dependencies, inheritance and differences between the parameters and values were therefore made visible to the user.

3.3 System Design

Figure 1 shows an activity diagram illustrating the creation process of a new CID screen design layout. Figure 2 shows the designed user interfaces, derived from the activity diagram.

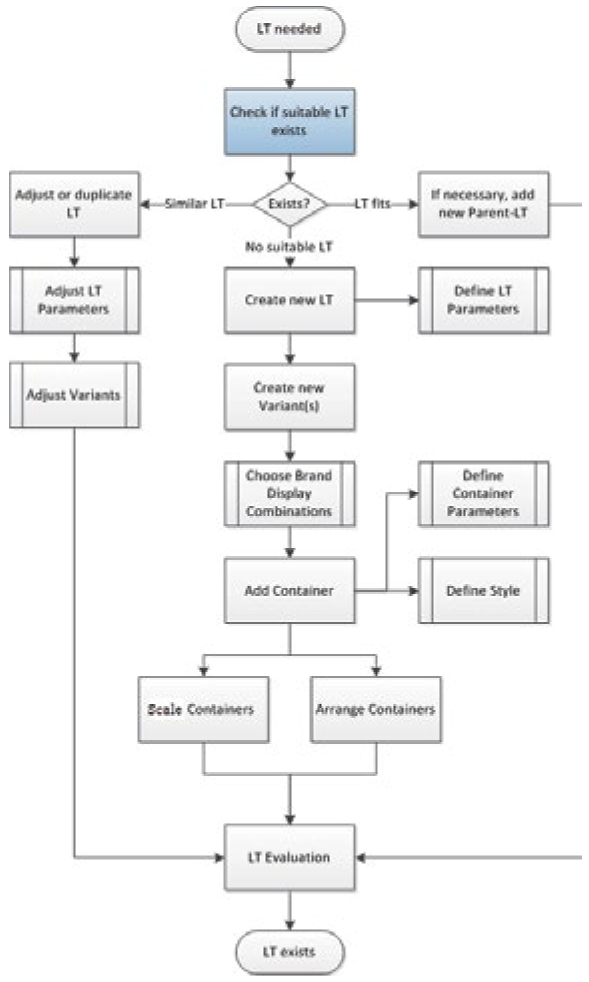

Figure 1: Different development processes to implement a unique layout template

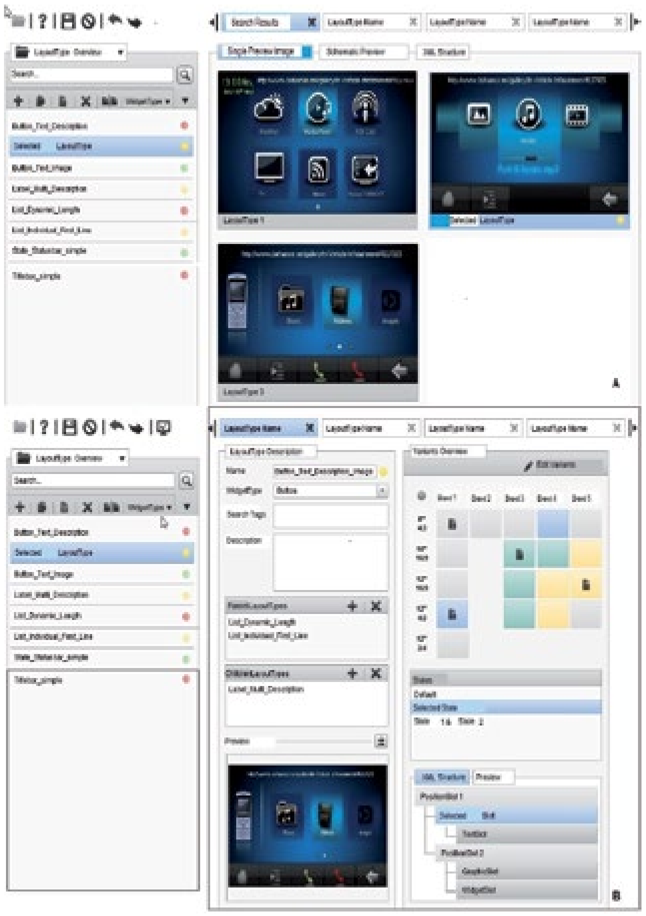

Figure 2: User Interface showing two different modes. A depicts the search mode for layout templates or variants. B depicts the layout template mode.

Our tool enables the creation and adjustment of Layout Templates. When the user gets a new screen design and a matching Layout Template (LT) exists, it can be duplicated and adjusted to the specific requirements of the target display. If the Layout Template meets all requirements, the new parent and child Layout Templates can be added to the user context. In any case, after all changes the LT needs to be verified. According to the requirement analysis results, the system was designed to target three different kinds of users: designers, layout developers and system developers. Each user group has different demands that concern not only the functionality but also the GUI representation of the in-vehicle information. Based on the conducted analysis, the system was designed through a top-down and bottom-up approach which allowed the creation of data flow diagrams as the first step of the system design phase. The necessary functions enlightened by the mental models obtained in the requirement analysis phase were clustered to system components and modules in order to develop Entity-Relationship-Models (ERM), which are required for the system architecture development.
Based on these levels of functions clustering, the front-end was incrementally designed according to the process described in [18]. The physical dialog structure, the system tree, and the navigation were laid out according to the main use cases and tasks. We then placed the previously defined functions into this framework. The resulting design concept was subsequently evaluated with users and usability professionals and optimized accordingly.

3.4 System Components

Figure 3 shows the main system components in form of a user work flow in which the functions are clustered into system features and interface components. These functions are additionally connected according to their accessibility in the system. The data on which the system features are based is reflected in the system components specification below.

- Search: search Layout Templates and variants.
- Relationship Control: visualizes parent-child structures and LT connections.
- Style Control: creates, modifies and deletes styles. Lists stylesheets for specific brands.
- Constants Control: creates, modifies and deletes constants.
- State Preview: allows the nested display of Layout Templates as a state.
- LT Control: management of all LTs (creation, edition, deletion and view).
- LT Overview: all available LTs for selection.
- LT Comparison: detailed view of two LTs, editing or transferring of properties.
- LT Properties: provides details about the LTs properties and variants.
- LT Parameters: subcomponent of the LT detail definition properties containing the LT-specific parameters.
- Variants Control: all LT variants and creation of new ones.
- Variants Overview: all LT variants.
- Variants Comparison: two Layout Templates variants, editing of parameters.
- LT Canvas: part component of the variant detail definition: creates graphic layout.
- Container Parameters: part of the variant properties. They define the position and size of the elements contained.
- Layout Pattern: detailed definition of a variant.
- Toolkit: elements (containers) for the construction of LTs.
- Formula editor: makes possible for the user to insert the formulas to calculate the containers size.

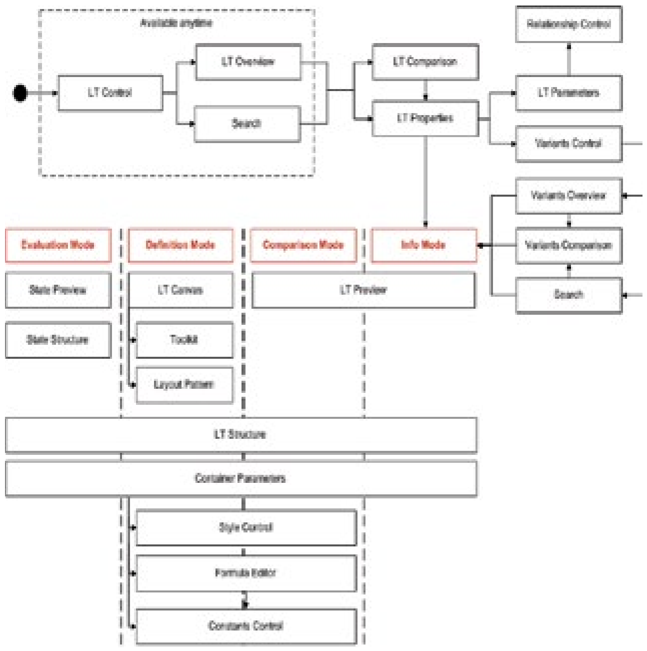

Figure 3: Main system components

4. CODE INSPECTION

To examine the quality of the tool implementation, we performed a formative evaluation obtaining feedback in the early design phase with the main goal being to improve design. Therefore, we provided the system with features in order to evaluate the Layout Templates. The evaluation was intended to be initiated by the user, producing feedback about the quality of the tested Layout Templates. In the case of issues/errors , the procedure was disrupted and the issues were displayed to the user.
We established verification rules which must be met for the further and successful summative evaluation. In addition, these rules were transferred into calculation algorithms, executed by a defined tree map algorithm relying on the approach by [1]. The tree map algorithms calculated the single containers of the Layout Template positions and sizes. A view of Layout Templates was defined for a variant or a combination of brands and displays that produced a huge variety of possible structures for different variants. This diversity can be highly complex as shown in the example in Figure 4. Therefore, we additionally proposed calculation algorithms and a possible approach for a largely automated computation of variants. To test the functionality of the tool, we divided the Layout Templates into the following categories: referenced objects; such as styles, constants and formulas; internal compatibility and external compatibility.

4.1 Internal Compatibility

The tool assesses the properties of the Layout Template (such as kinship, name, widgettype, etc) and ensures that the Layout Template contains a minimum of information and values. Through verification rules, transferred into axioms and a procedural program code, circular references and excessive or insufficient space in the formulas were shown to the user.
The space condition rules defined are presented exemplary

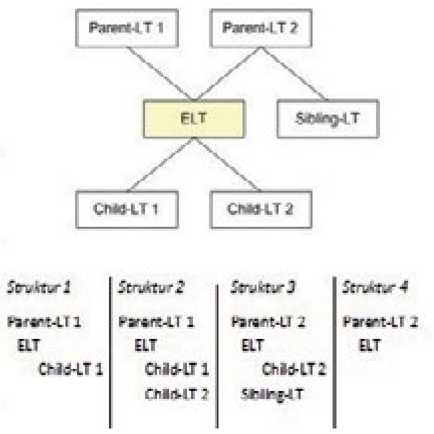

Figure 4: Different Layout Templates combinations depending on the brand or screen characteristics. ELT denotes evaluated Layout Template.

below by the axioms relating the widths of a parent container with its child containers. The axioms compare the size of a certain container with the size of the embedded containers when the embedded containers are positioned in a row, one next to another. Other axioms apply if the containers are positioned freely or in a column layout.

$$\alpha = \{Slot||Total - LT\}, n = Length(cslot[])$$
$$cslot = \{directchildcontainers\}/\{childcontainersofchildren\}$$

The width is calculated for the three different cases. Axiom (1) denotes the calculation for fixed values, auto and formulas. Axiom (2) is to be applied in case of a variable value.

$$width(\alpha) = \sum_{i=0}^{n}\{width(cslot[i]) + margin\_left(cslot[i])\} \tag{1}$$

$$\begin{aligned} width\_min(\alpha) \leq \sum_{i=0}^{n}\{width(cslot[i]) + \\ margin\_left(cslot[i])\} \leq width\_max(\alpha) \end{aligned} \tag{2}$$

The height is calculated through the following axioms for fixed values and formulas (3), auto (4) and variable values (5).

$$\begin{aligned} height(\alpha) = height(cslot[0...n]) + \\ margin\_top(cslot[0...n]) \end{aligned} \tag{3}$$

$$\begin{aligned} height(\alpha) = max(height(cslot[0...n]) + \\ margin\_top(cslot[0...n])) \end{aligned} \tag{4}$$

$$\begin{aligned} height\_min(\alpha) \leq height(cslot[0...n]) + \\ margin\_top(cslot[0...n]) \leq height\_max(\alpha) \end{aligned} \tag{5}$$

4.2 External Compatibility

The LT is assessed in the context of other Layout Templates. We verified the space conditions between the Layout Template and its parents and children. For example, the next section illustrates two verification rules, potential error cases and methods for determining the path and evaluating the space distribution.
As a rule to verify the external compatibility test, we determined that the height and width of the top containers of the Layout Templates must be equal to the height or width of the container of the parent LT where the Layout Template was embedded. This rule must be applied to every binding of one LT to another. If a problem between the tested Layout Template and the parent, or between the Layout Template and a child arises, it is always assigned to the tested Layout Template.

4.2.1 *Error cases for the external compatibility test*

We defined the following error cases for the described rule:

- a Layout Template provided insufficient space for the contained child LTs;
- a Layout Template provided more space than the child LTs occupy;
- a parent Layout Template provided insufficient space for the tested LT;

4.2.2 *Path discovery for different views containing a LT*

Each Layout Template has one or more variations which represent one or more brand-display combinations. Additionally, each variant has one or more states, such as active or inactive. Additionally, a LT has parent and child LTs, and therefore different sets of parent-child combinations. As this is the case for any Layout Template, the embedding of several LTs to build up a view resulted in approximately 10000 paths to test. As a consequence, we defined an algorithm to determine the correct path and to improve the performance of the discovery processing time.

4.2.3 *Evaluation algorithm for a view containing a LT*

We considered two cases: (1) the internal compatibility of the LTS with the parent LTs, as well as (2) the internal compatibility of the LTs children with the LT. In both cases, the test was run according to the same pattern: an LT was inserted into an assigned parent LT. For this purpose, the algorithm calculated the total size of the LTs and the size of the slot of the parent LTs, in which the LT had to be inserted.

5. CONCLUSION AND FUTURE WORK

In this paper, we have presented a software solution for creating and editing flexible in-vehicle graphical user interfaces (GUI) through reusable elements or widgets. The solution is based on variants, through the separation of content, layout and logic, and allows for a centralized, dynamic creation of user interfaces which can be adapted to different vehicle brands and displays. The embedded mechanisms used to verify the applicability and reliability of our tool confirmed the gains in flexibility of the layout design. However, future steps still need to be taken to extend and improve the tool considering additional usage cases extending as well the layouttypes functionality and evaluation in a final XML format.

6. REFERENCES

[1] H. Keränen and J. Plomp, "Adaptive runtime layout of hierarchical ui components," in *Proceedings of the second Nordic conference on Human-computer interaction.* ACM, 2002, pp. 251–254.

[2] C. Olaverri-Monreal, C. Lehsing, N. Trübswetter, C. A. Schepp, and K. J. Bengler, "In-vehicle displays: Information priorizitation and visualization while driving," in *Intelligent Vehicles Symposium (IV13).* IEEE, 2013.

[3] Volkswagenag.com, "Volkswagen aktiengesellschaft annual report 2008," 2009.

[4] T. Fleischmann, "Model based hmi specification in an automotive context," *Human Interface and the Management of Information. Methods, Techniques and Tools in Information Design*, pp. 31–39, 2007.

[5] S. Hess, A. Gross, A. Maier, M. Orfgen, and G. Meixner, "Standardizing model-based in-vehicle infotainment development in the german automotive industry," in *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications.* ACM, 2012, pp. 59–66.

[6] E. Whitehead Jr, J. Robbins, N. Medvidovic, and R. Taylor, "Software architecture: Foundation of a software component marketplace," in *Proceedings of the First International Workshop on Architectures for Software Systems*, 1995, pp. 276–282.

[7] K. Leong, R. Love, and H. Tsuji, "Display window layout system that automatically accommodates changes in display resolution, font size and national language," Apr. 30 1996, uS Patent 5,513,342.

[8] B. Ross, M. Schackwitz, and K. Young, "Desktop publishing software for automatically changing the layout of content-filled documents," Feb. 15 2000, uS Patent 6,026,417.

[9] J. Eisenstein, J. Vanderdonckt, and A. Puerta, "Applying model-based techniques to the development of uis for mobile computers," in *Proceedings of the 6th international conference on Intelligent user interfaces.* ACM, 2001, pp. 69–76.

[10] F. Hüger, "User interface transfer for driver information systems: a survey and an improved approach," *Automotive UI*, vol. 11, 2011.

[11] K. Perlin and J. Meyer, "Nested user interface components," in *Proceedings of the 12th annual ACM symposium on User interface software and technology.* ACM, 1999, pp. 11–18.

[12] E. Kandogan and B. Shneiderman, "Elastic windows: improved spatial layout and rapid multiple window operations," in *Proceedings of the workshop on Advanced visual interfaces.* ACM, 1996, pp. 29–38.

[13] R. Taylor, N. Medvidovic, K. Anderson, E. Whitehead Jr, J. Robbins, K. Nies, P. Oreizy, and D. Dubrow, "A component-and message-based architectural style for gui software," *Software Engineering, IEEE Transactions on*, vol. 22, no. 6, pp. 390–406, 1996.

[14] J. Greenfield and K. Short, "Software factories: assembling applications with patterns, models, frameworks and tools," in *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2003, pp. 16–27.

[15] P. McKinley, S. Sadjadi, E. Kasten, and B. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56–64, 2004.

[16] C. Kerer and E. Kirda, "Layout, content and logic separation in web engineering," *Web Engineering*, pp. 135–147, 2001.

[17] C. Olaverri-Monreal, K.-J. Bengler, M. Breisinger, and C. Draxler, "Markup languages and menu structure transformation during the internationalisation process of driver information systems," *Localisation Focus*, p. 4, 2010.

[18] J. Tidwell, *Designing interfaces*. O'Reilly Media, Incorporated, 2010.