Evaluating Application Prototypes in the Automobile

Automotive application developers often rely on computer simulations or driving simulators for testing their applications. A prototyping platform based on user-centered design principles allows early evaluations to take place in drivable cars.

Holger Hoffmann
and Jan Marco Leimeister
Kassel University

Pervasive automotive applications are becoming a key driver for innovation among car manufacturers and their suppliers. Customers want applications they know from their desktops and mobile devices also available in their cars.[1] For these pervasive applications to succeed, they must interact with drivers intuitively to avoid creating a distraction. Peter Mambrey and Volkmar Pipek have shown that differences in knowledge and perspective between users and developers aggravate this situation.[2] Users often don't know the possibilities arising from novel technologies and only become aware of an application once it's available. At that point, developers, who aren't fully aware of the application's future usage context, have already terminated their (traditional) working process. User-centered design can narrow these differences.

To obtain reasonable results from evaluations, developers must choose the kind of prototype and the evaluation setting consciously and wisely. Prototypes range from low-fidelity paper prototypes to fully functional high-fidelity systems. For complex systems, such as pervasive car applications, low-fidelity prototypes make it harder to distinguish application from prototype characteristics; hence, high-fidelity prototypes are preferred.[3] Evaluation settings range from laboratories to real-life situations, depending on what's being evaluated.[4] Most automotive software development projects use computer simulations[1] or driving simulators for user evaluations. However, driving simulators affect test subjects' perceptions and thus bias evaluation results.[5] As Andreas Riener noted, driving errors made using a simulator aren't hazardous, so subjects concentrate less on driving than when driving a real car, and because there are no consequences for breaking traffic laws, subjects don't follow the rules as strictly as when driving in reality.[5] In addition, the simulation's surreal characteristics—it lacks noises and vibrations, uses unrealistic scenery, and so on—negatively impact the validity of the subjects' measured behavior in the car.[6]

Although simulations are suitable in early studies, they leave a gap in the human-centered design cycle. Simulations cannot cover users' perceptions of using the application in a real car. To close this gap, developers should evaluate applications that have been found safe for use during computer simulations or in driving simulators, while driving a car either on a test track or in real traffic. However, a car's complex technical infrastructure and the pervasive nature of the applications complicate the integration of prototypes into an automobile:

- In today's cars, functions are distributed over multiple electronic control units (ECUs); for

Evaluation of HIMEPP as a Prototyping Environment

To evaluate whether the highly integrated modular embedded prototyping platform (HIMEPP) meets Scott Klemmer and his colleagues' requirements for a prototyping environment,[1] we collected feedback from 14 developers who used it to create at least one prototype. All developers had backgrounds in computer science, information systems, or engineering.

We introduced the developers to HIMEPP in small groups during three-hour workshops. After the developers had used all the prototypes, they completed posttask questionnaires. Using questions derived from Viswanath Venkatesh and his colleagues' Unified Theory of Acceptance and Use of Technology,[2] we asked developers about their expectancy and experience in terms of performance and effort as well as social influence and facilitating conditions when using HIMEPP and whether they intended to continue using it.

The feedback we collected shows that HIMEPP meets all three of the Klemmer requirements. First, by providing a Java-based software framework, developers can easily learn to work with HIMEPP. Ten developers found learning and working with HIMEPP easy. Second, preassembled base components cover all interfaces to the user and car. Using them requires little programming experience. Twelve developers stated that they had all the resources necessary and enough knowledge to use HIMEPP. Plus, HIMEPP's highly modular architecture and hardware infrastructure let them easily add and modify components. Using off-the-shelf hardware and software in the car's native environment, HIMEPP supports the rapid creation, evaluation, and modification of prototypes. Thirteen developers responded that creating prototypes using HIMEPP was more rapid than the tools and techniques they had before. Although six developers mentioned compatibility problems with other systems, all but one planned to continue using HIMEPP in the coming year.

REFERENCES

1. S.R. Klemmer et al., "Suede: A Wizard of Oz Prototyping Tool for Speech User Interfaces," *Proc. 13th Ann. ACM Symp. User Interface Software and Technology*, ACM Press, 2000, pp. 1–10.

2. V. Venkatesh et al., "User Acceptance of Information Technology: Toward a Unified View," *MIS Quarterly*, vol. 27, no. 3, 2003, pp. 425–478.

example, the indicator lights use up to eight ECUs.[6]

- Even for the automaker, ECUs are often a "black box" because different suppliers build them to meet an interface specification.[7]
- Multiple hardware and software revisions of one ECU exist for any given car model, each conforming to the interface definition, but working differently.[7]

Consequently, integrating a working prototype into the embedded infrastructure is time-consuming and expensive, and it depends on expertise often found only with ECU domain experts. This reduces the possibilities for other developers to assess user requirements, weigh design options, and conduct user evaluations. Prototyping tools can help solve this dilemma by helping speed up design cycles, saving time and money. This makes it easier for other developers to create new applications and facilitate the gathering of user feedback throughout the process.[8]

A Rapid Prototyping Environment for Automotive Applications

The highly integrated modular embedded prototyping platform (HIMEPP) closes the gap in the user-centered design of pervasive applications for the automobile by allowing user evaluations in a real environment. HIMEPP's design goals follow Scott Klemmer and his colleagues' argument[9] that effective prototyping tools

- must be easy to learn,
- require little programming expertise, and
- support the rapid creation, evaluation, and modification of prototypes.

(See the "Evaluation of HIMEPP as a Prototyping Environment" sidebar.)

HIMEPP includes a hardware platform installed in a stock car and a software framework for development support. The hardware integration lets developers implement their prototypes on a standard computer, which is less complex and has lower development costs than embedded prototypes. The software framework supports prototype implementations by providing scaffolding and preassembled software components. Both reduce development time and require less domain expertise, opening evaluations to developers with limited programming skills. HIMEPP lets developers run multiple iterations of a prototype, a core concern voiced by Björn Hartmann and his colleagues.[10]

Hardware Platform and Automobile Interfaces

HIMEPP's hardware platform is based on a standard Intel-compatible computer designed for the automotive environment. It differs from a standard computer only in its dimensions—that is, it easily fits into a car's glove compartment—and the extended operational temperature range.

Developing applications that appear to run natively in the car requires users to interact with the prototype through the user interfaces found in the car. In the Audi A4, we used as a prototyping environment; the visual output uses the display in the middle of the dashboard and sound output uses the car's speaker system. Whereas the prototype's sound output is reached using the car stereo's auxiliary input line, graphical output requires a more complex setup. To convey a natural feeling during evaluations, regular in-car applications must remain available. We achieve this using a Y-switch that lets users select either the head unit (for regular applications) or our system (for the prototype) for the visual output source. To further enhance the illusion of the prototype as an integrated standard application, our setup lets users switch the video input through the Audi user interface's standard application selection method, the haptic controller.

Other interfaces available to developers include the buttons, switches, and levers on the dashboard and steering wheel. Because these components are connected to the car's central controller-area network (CAN) bus, we added a CAN interface card to the prototyping platform. We also included a microphone for speech recognition to allow speech-controlled pervasive applications. Because car microphones don't fit the standard computer input, we integrated an off-the-shelf microphone hidden near the center rearview mirror.

To allow developers to create high-fidelity prototypes of mobile applications, we also added a GPS receiver for location-aware applications (such as point-of-interest routing); a General Packet Radio Service/Universal Mobile Telecommunications System (GPRS/UMTS) modem and a Wi-Fi network card for mobile data transfer; and a cable-bound network card to link with other computers in the car—for example, to remotely control or debug prototypes.

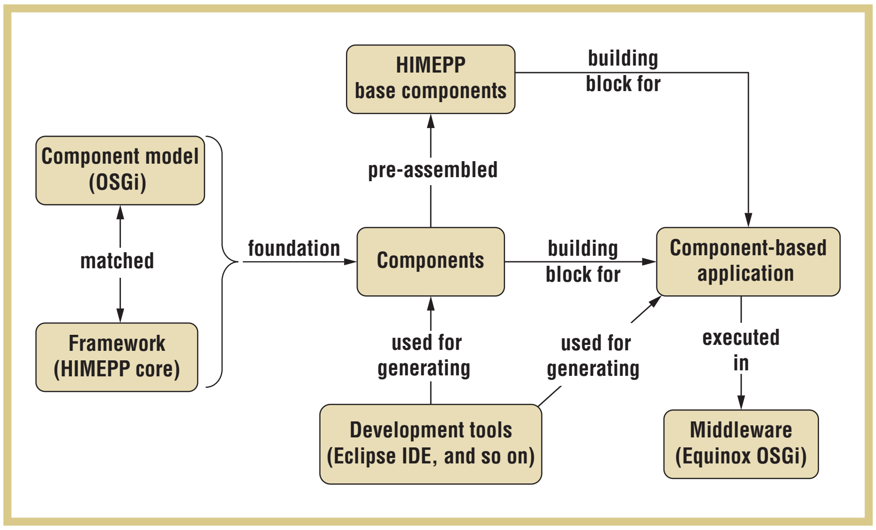


Figure 1. Links between elements of a component-oriented architecture (adapted from Volker Gruhn and Andreas Thiel's work[13]) show how the architecture's different design elements are combined to create a modular prototyping environment.

Software Architecture

Whereas the hardware platform's goal is integrating standard hardware into the car, the software platform allows developers to integrate their applications into the car's usage concept. To this end, the platform provides access to human-car interfaces and car data interfaces.

The user-centered approach focuses on iterative implementation of functionality while repeatedly evaluating the design and implementation.[11] Hence, our software platform needs to make software functions easily exchangeable so developers can evaluate alternatives and make design revisions. Furthermore, because automotive software is developed outside the automobile, developers must be able to test the system outside a car, simulating user interfaces and car-related data.

Using a component-oriented approach for the software platform's architecture satisfies both requirements.[12] Figure 1 shows the design elements of a component-oriented software architecture. The figure also illustrates the corresponding HIMEPP elements and pre-assembled base components added for reuse in projects.

Platform Design Elements

Volker Gruhn and Andreas Thiel describe the component model with a matching framework and middleware as the core of component-oriented architectures.[13] Supplements are development tools that ease the creation of components and component-based prototypes.

Components are the architecture's name-giving elements. In Clemens Szyperski's frequently used definition, *components* are reusable, encapsulated logical units that implement certain functionality made available through public interfaces.[12] A *component-based application* uses the public interfaces of one or more components to create a more complex application.

The *component model* defines the syntactic and semantic standard for implementing a component-based application, covering interface definitions, naming conventions, and intercomponent connectivity.[12] The components' runtime environment, the *middleware*, is closely related to the component model. Because HIMEPP simplifies rapid prototyping of pervasive applications that users can evaluate, the component model must be chosen accordingly. The OSGi component model, adding a module system and service platform to the widely used Java programming language, is a suitable foundation for an easy-to-handle prototyping platform. Programming in

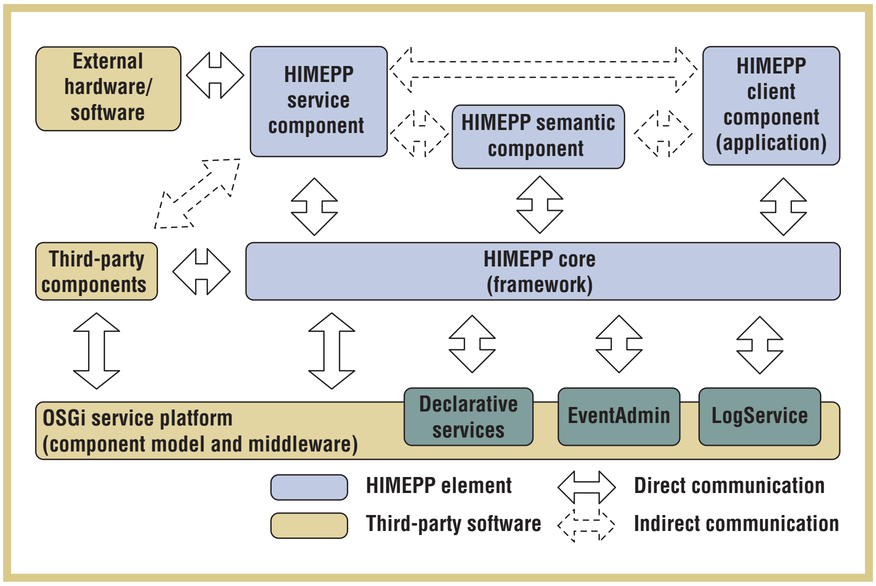


Figure 2. Communication and component interplay within the highly integrated modular embedded prototyping platform (HIMEPP). Based on the broadly used OSGi service platform, the HIMEPP framework and components enable domain-specific development and allow for integrating prototypes using standard hard- and software into the car's infrastructure.

Java improves the developers' learning curve, and many development tools are available for developing Java applications. For example, the Eclipse integrated development environment (IDE) is based on OSGi technology and thus includes the tools necessary for build and test cycles on the developer's machine.

The prototyping platform's framework defines domain-specific principles and provides rudimentary functionality for component developers.[12] It thus eases the development process by defining coding principles to ensure the compatibility of different developers' components within the framework. The HIMEPP core framework defines several coding principles and offers supporting functions. HIMEPP's most important framework features are functions for event-based intercomponent communication (for loose coupling), dynamic starting and stopping of components (to save the embedded hardware's resources), enhanced logging functionality (for system testing and evaluation), and interfaces to third-party components. Figure 2 gives a system overview.

We included a wizard for setting up new projects to enhance the Eclipse IDE's functionality. Using the wizard, developers can select base components for their prototype and determine events the prototype should react to. The wizard automatically generates the necessary structure for an OSGi-compatible component. It sets up the component scaffolding, providing the developer with method stubs for the component's application logic.

HIMEPP's component-oriented design contributes to the fulfillment of the two major requirements we've described. Using the HIMEPP framework, the OSGi middleware, and the Eclipse IDE, software functions can be exchanged effortlessly. The eventing mechanism lets developers test the application outside of the car (for example, by injecting simulated events into the stream).

HIMEPP Base Components

HIMEPP extends the design elements derived from Gruhn and Thiel by offering preassembled components of frequently used functions (for example, access to user interfaces).[13] Providing such base components reduces development time and prevents implementation errors. Additionally, having base components that cover user interfaces in the car ensures a real-life user experience; novel applications have the look and feel of existing applications. Following the separation of concerns design principle, HIMEPP separates data input from an external source into the direct interaction with external hardware or software, optional semantic data interpretation, and the command execution in the application (see the components in the top right of Figure 2). Hence, minimal effort is required to add new hardware to the prototyping platform and to evaluate multiple interaction concepts by exchanging the semantic interpretation component.

HIMEPP base components are grouped into components that are part of the in-car prototype and components that developers can use when implementing and testing the application at their desks. Prototype components support application development by providing reusable functions. We identified the components most useful to developers by analyzing the user interface elements and data sources of existing applications on the stock Audi Infotainment Platform as well as applications in current research projects. Figure 3 shows some relevant elements found during that analysis.

Components used at the developer's desk represent and replace the in-car components, letting developers interact with the system as if it were installed in the car. The separation of concerns principle and the HIMEPP eventing mechanism (see Figure 2) make this exchange possible. The developer equivalent of a user interface component is a new HIMEPP service component that sends the same events and has the same methods as the user interface component it represents. The components interpreting user input (either the corresponding semantic or the application itself) are left unaltered; the simulated events appear to come from the regular component in the car. Thus, developers can quickly change the interaction with the pervasive applications by implementing different versions of the same semantic interpretation components. Hence, they can evaluate different interaction concepts of pervasive applications in the car without changing the application logic.

Case Study: Implementation and Evaluation of a Virtual Co-Driver

We present a case study in which another researcher in our department used HIMEPP to create a prototype for evaluating a novel interaction concept. The target *virtual co-driver* application consists of an avatar and artificial intelligence to help drivers cope with the increasing number of applications in cars.

Functionality

A virtual co-driver system (Avicos) gives the driver information about operating the car and lets the driver control a set of functions using natural speech.[14] Avicos offers two modes.

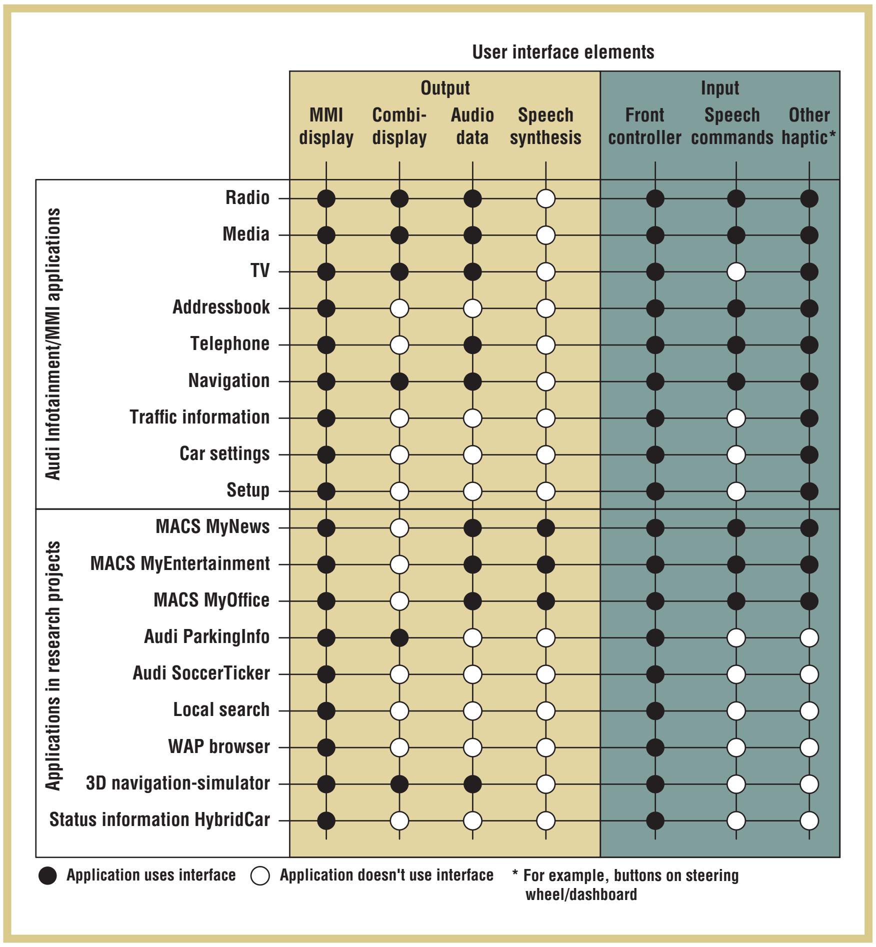
In *handbook mode*, Avicos answers questions about the car's functions and lets the driver change entertainment and comfort settings using speech input. Avicos responds using text-to-speech synthesis to output handbook texts. To improve human-car interaction, a female avatar on the dashboard display indicates the device usage.

In *touch-and-tell mode*, Avicos lets drivers learn about unfamiliar devices. After activating this mode, the driver uses the device (that is, the button, knob, or lever) and receives the corresponding information about the device as in handbook mode.

Implementation

The Avicos prototype has three subsystems: the input system, the reaction determination system, and the output system. The input and output systems use HIMEPP base components. The avatar's rendering engine was added as a component. The reaction determination system holds the application logic and thus was implemented from scratch.

The first step in development using HIMEPP is adding a new project using the HIMEPP wizard in Eclipse. On the first screen, the wizard collects all data necessary for setting up component scaffolding—for example, its name and starting procedure, and whether it provides a service to other components or triggers events. Developers can select services provided by other components (that is, base and self-developed components) on their machine to import and subscribe to events from those components. For Avicos, the base components imported were audio output, GPS locations (for evaluation purposes), speech input and output, and the new avatar component. Additionally, the Avicos prototype component subscribes to events that inform it about finished audio output and speech synthesis, new user input via a haptic device or speech recognition, and new data from the CAN bus or GPS device (left window in Figure 4). On the wizard's second screen, developers can add Java and operating-system-specific libraries referenced in their code, as well as resources to be included in the component. Avicos imports the Chatterbean library, a Java version of the Alice conversational agent, as well

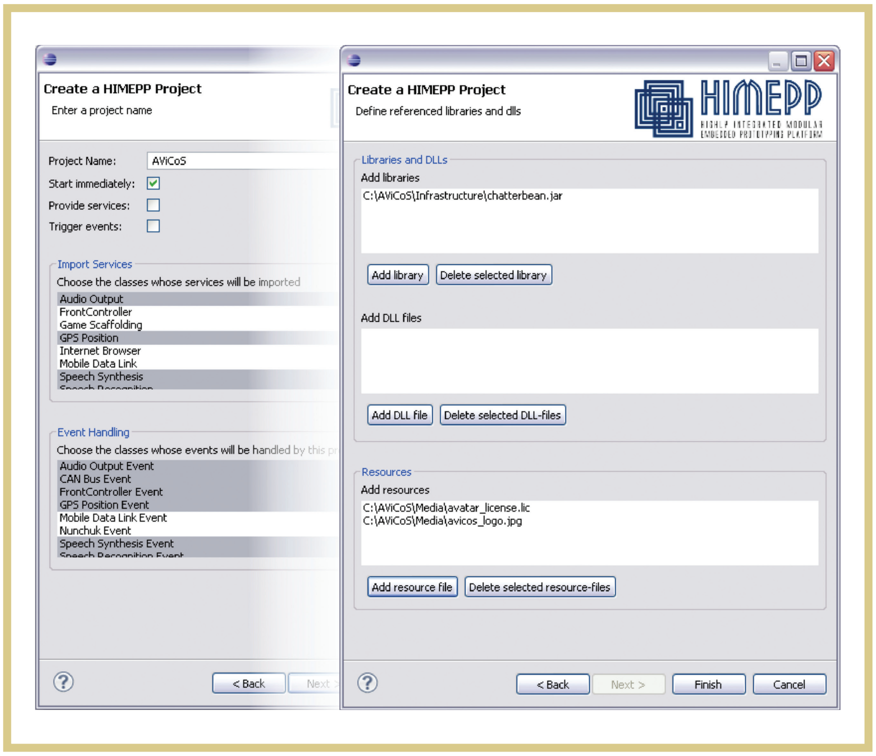


User interface elements

		Output				Input		
		MMI display	Combi-display	Audio data	Speech synthesis	Front controller	Speech commands	Other haptic*
Audi Infotainment/MMI applications	Radio	●	●	●	○	●	●	●
	Media	●	●	●	○	●	●	●
	TV	●	●	●	○	●	○	●
	Addressbook	●	○	○	○	●	●	●
	Telephone	●	○	●	○	●	●	●
	Navigation	●	●	●	○	●	●	●
	Traffic information	●	○	○	○	●	○	●
	Car settings	●	○	○	○	●	○	●
	Setup	●	○	○	○	●	○	●
Applications in research projects	MACS MyNews	●	○	●	●	●	●	●
	MACS MyEntertainment	●	○	●	●	●	●	●
	MACS MyOffice	●	○	●	●	●	●	●
	Audi ParkingInfo	●	●	○	○	●	○	○
	Audi SoccerTicker	●	○	○	○	●	○	○
	Local search	●	○	○	○	●	○	○
	WAP browser	●	○	○	○	●	○	○
	3D navigation-simulator	●	●	●	○	●	○	○
	Status information HybridCar	●	○	○	○	●	○	○

● Application uses interface ○ Application doesn't use interface * For example, buttons on steering wheel/dashboard

Figure 3. By analyzing which user interface elements are used in existing end-user and research applications, the base components needed in HIMEPP to support development of such applications are derived.


Create a HIMEPP Project
Enter a project name

Project Name: AViCoS
Start immediately: ✓
Provide services: ☐
Trigger events: ☐

Import Services
Choose the classes whose services will be imported
Audio Output
FrontController
Game Scaffolding
GPS Position
Internet Browser
Mobile Data Link
Speech Synthesis

Event Handling
Choose the classes whose events will be handled by this
Audio Output Event
CAN Bus Event
FrontController Event
GPS Position Event
Mobile Data Link Event
Nunchuk Event
Speech Synthesis Event

Create a HIMEPP Project
Define referenced libraries and dlls

Libraries and DLLs
Add libraries
C:\AViCoS\Infrastructure\chatterbean.jar
Add DLL files

Resources
Add resources
C:\AViCoS\Media\avatar_license.lic
C:\AViCoS\Media\avicos_logo.jpg

HIMEPP
HIGHLY INTEGRATED MODULAR EMBEDDED PROTOTYPING PLATFORM
Add library
Delete selected library
Add DLL file
Delete selected DLL-files
Add resource file
Delete selected resource-files
< Back
Next >
Finish
Cancel

Figure 4. Setting up the Avicos prototype with the HIMEPP wizard. In the first window (left side), the developer enters the project name and chooses services to import and events to subscribe to. In the next window (right side), the developer selects libraries and other resources.

as the license file needed by the third-party application rendering the avatar and a logo to display at startup (see the right window in Figure 4).

After the developer finishes the wizard, HIMEPP presents the project setup in the Eclipse environment. Method stubs for the component's application logic are available, the component definitions needed for OSGi are set up automatically, and services selected in the wizard are set up for use. Additionally, a run configuration is created for the OSGi runtime, specifying the components to use and their launch sequence for the overall application.

Avicos works by gathering user input from CAN bus, haptic interface, and speech recognition. It then passes the input to the (external) conversational agent that interprets it and determines a response, which is presented to the user using the avatar component as well as speech output.

During the implementation and testing phase, developers can launch the prototype on their own machines by starting a new OSGi runtime, using the run configuration initially created by the HIMEPP wizard. The last development step is deploying the prototype in the car. Selecting the newly created run configuration in the deployment wizard wirelessly copies the entire prototype to the hardware platform in the car. The Avicos implementation used the developer base components for testing the functionality of the virtual co-driver at the developer's desk— for example, by simulating certain speech input or bus signals. When transferred to the car, the Avicos application reacted as it did on the development machine.

Evaluation

Sixty-seven subjects evaluated Avicos, which was installed in a modified stock car. The developer videotaped the evaluation, recording the driving situation as well as the subjects' behavior, as illustrated in Figure 5. During evaluation, subjects used the system to complete simple tasks, either while parking or driving in real traffic. A developer was present during all the evaluations to oversee the process, answer questions, and react in case of a user or system error.

From the video analysis and before and after questionnaires, the prototype's benefits were identified by comparing the virtual co-driver to the user handbook, including Avicos's perceived ease of use and perceived usefulness in real-world scenarios. The prototyping platform proved to be sophisticated enough to run the system without any developer interaction.

The possibilities introduced by HIMEPP must be tempered with three shortcomings. First, HIMEPP doesn't prevent developers from evaluating applications in the car that might be dangerous to use. Intended usage focuses on evaluating applications that were previously found safe in simulations or a driving simulator. However, it seems necessary to find a way to disable developers from creating unsafe application logic while upholding HIMEPP's potential. Second, HIMEPP's potential to create prototypes using hardware is limited. Although the hardware platform lets us integrate new user interfaces (we evaluated the use of a Nintendo Wii Nunchuk controller in the car) and external hardware (for example, a second GPS device), changes to existing interfaces, such as the steering wheel unit, are still expensive to realize. Third, we haven't been able to implement a Y-switch to send CAN messages either to the car's ECUs or HIMEPP as we did for switching the display output. Although HIMEPP ignores messages not intended for the prototype, the ECUs receive and process all messages. Currently, a workaround disables functions in the ECUs to prevent side effects when a HIMEPP prototype reacts to a CAN message. However, this leaves the functions also unavailable during the car's regular operation.

Further research is required in many areas. Foremost, our approach to develop prototypes in the automobile centers on enabling the developer to present a working application in the car. This is a crucial first step, but extensive data gathering during evaluations and subsequent analyses of this data is required for developers to understand users' needs and reactions to prototypes. Although HIMEPP supports data gathering from component events, the means for combining and analyzing that data with developer-recorded data (such as evaluation videos)[10] are missing and need exploration. Moreover, HIMEPP could let users design prototypes using predefined application blocks representing user interface or data elements. The platform should be expanded to allow this visual design of applications by "dragging and dropping" prebuilt blocks, enabling users to create custom prototypes.

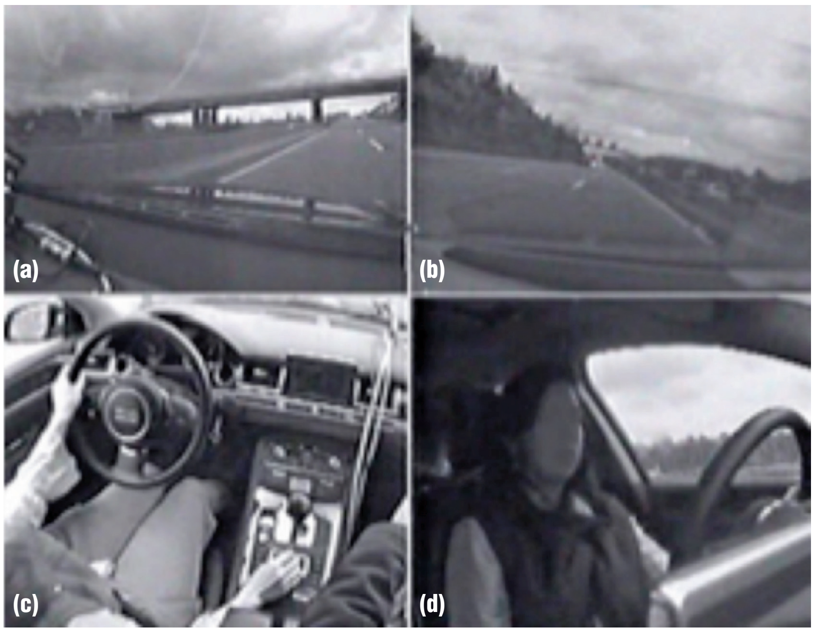


Figure 5. Multiplexed video stills from a user evaluation on the road, documenting the traffic situation on the road ahead (a) and behind (b) as well as user interaction by filming the position of the driver's hands and the content on the screen (c) and the driver's gaze (d). (Face blurred to protect the subject's privacy.)

the AUTHORS

Holger Hoffmann is a postdoctoral researcher at the Research Group for Information Systems at Kassel University. He works with research groups on automotive software and services, ubiquitous and mobile computing, and open innovation and runs publicly funded research projects. His teaching and research areas include automotive applications, ubiquitous and mobile computing, collaboration engineering, and project management. Contact him at holger.hoffmann@uni-kassel.de.

Jan Marco Leimeister is a full professor of information systems and is the Chair for Information Systems at Kassel University. He runs research groups on virtual communities, e-health, and ubiquitous and mobile computing and manages several publicly funded research projects. His teaching and research areas include IT innovation management, service science, ubiquitous and mobile computing, collaboration engineering, e-health, online communities, and IT management. Contact him at leimeister@uni-kassel.de.

REFERENCES

1. D.D. Salvucci, "Rapid Prototyping and Evaluation of In-Vehicle Interfaces," *ACM Trans. Computer-Human Interaction*, vol. 16, no. 2, 2009, pp. 1–33.
2. P. Mambrey and V. Pipek, "Enhancing Participatory Design by Multiple Communication Channels," *Human-Computer Interaction: Communication, Cooperation, and Application Design*, vol. 2, H.-J. Bullinger and J. Ziegler, eds., Lawrence Earlbaum Associates, 1999, pp. 387–391.
3. Y.-K. Lim et al., "Comparative Analysis of High- and Low-Fidelity Prototypes for More Valid Usability Evaluations of Mobile Devices," *Proc. 4th Nordic Conf. Human–Computer Interaction*, ACM Press, 2006, pp. 291–300.
4. H. Sharp, Y. Rogers, and J. Preece, *Interaction Design*, 2nd ed., John Wiley & Sons, 2007.
5. A. Riener, "Simulating On-the-Road Behavior Using Driving Simulators," *Proc. 3rd Int'l Conf. Advances in Computer–Human Interactions*, R. Jarvis and C. Dini, eds., IEEE CS Press, 2010, pp. 25–31.
6. A. Pretschner et al., "Software Engineering for Automotive Systems—A Roadmap," *Proc. 2007 Future of Software Eng.* (FOSE 07), IEEE CS Press, 2007, pp. 55–71.
7. M. Broy et al., "Engineering Automotive Software," *Proc. IEEE*, vol. 95, 2007, pp. 356–373.
8. Y. Li, J.I. Hong, and J.A. Landay, "Topiary: A Tool for Prototyping Location-Enhanced Applications," *Proc.*

17th Ann. ACM Symp. User Interface Software and Technology, ACM Press, 2004, pp. 217–226.

9. S.R. Klemmer et al., "Suede: A Wizard of Oz Prototyping Tool for Speech User Interfaces," *Proc. 13th Ann. ACM Symp. User Interface Software and Technology*, ACM Press, 2000, pp. 1–10.

10. B. Hartmann et al., "Reflective Physical Prototyping Through Integrated Design, Test, and Analysis," *Proc. 19th Ann. ACM Symp. User Interface Software and Technology*, ACM Press, 2006, pp. 299–308.

11. Int'l Organization for Standardization, *ISO 13407 Human-Centred Design Processes for Interactive Systems*, ISO, 2000.

12. C. Szyperski, *Component Software—Beyond Object-Oriented Programming*, 2nd ed., ACM Press, 2002.

13. V. Gruhn and A. Thiel, *Komponentenmodelle*, Addison-Wesley, 2000.

14. V.A. Nicolescu, *Gestaltung Avatar-basierter Natürlichsprachlicher Hilfesysteme für den Einsatz in Fahrzeugen* [Avatar-Based Design of Natural Language Help Systems for Use in Vehicles], Cuvillier, 2009.

IEEE computer society

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field.
MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.
COMPUTER SOCIETY WEBSITE: www.computer.org
OMBUDSMAN: To check membership status or report a change of address, call the IEEE Member Services toll-free number, +1 800 678 4333 (US) or +1 732 981 0060 (international). Direct all other Computer Society-related questions—magazine delivery or unresolved complaints—to help@computer.org.
CHAPTERS: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.
AVAILABLE INFORMATION: To obtain more information on any of the following, contact Customer Service at +1 714 821 8380 or +1 800 272 6657:

- Membership applications
- Publications catalog
- Draft standards and order forms
- Technical committee list
- Technical committee application
- Chapter start-up procedures
- Student scholarship information
- Volunteer leaders/staff directory
- IEEE senior member grade application (requires 10 years
- practice and significant performance in five of those 10)

PUBLICATIONS AND ACTIVITIES
Computer: The flagship publication of the IEEE Computer Society, *Computer*, publishes peer-reviewed technical content that covers all aspects of computer science, computer engineering, technology, and applications.
Periodicals: The society publishes 13 magazines, 18 transactions, and one letters. Refer to membership application or request information as noted above.
Conference Proceedings & Books: Conference Publishing Services publishes more than 175 titles every year. CS Press publishes books in partnership with John Wiley & Sons.
Standards Working Groups: More than 150 groups produce IEEE standards used throughout the world.
Technical Committees: TCs provide professional interaction in more than 45 technical areas and directly influence computer engineering conferences and publications.
Conferences/Education: The society holds about 200 conferences each year and sponsors many educational activities, including computing science accreditation.
Certifications: The society offers two software developer credentials. For more information, visit www.computer.org/certification.

COMPUTER SOCIETY OFFICES
Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928
Phone: +1 202 371 0101 • **Fax:** +1 202 728 9614
Email: hq.ofc@computer.org
Los Alamitos: 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314
Phone: +1 714 821 8380
Email: help@computer.org

MEMBERSHIP & PUBLICATION ORDERS
Phone: +1 800 272 6657 • **Fax:** +1 714 821 4641 • Email: help@computer.org
Asia/Pacific: Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan
Phone: +81 3 3408 3118 • **Fax:** +81 3 3408 3553
Email: tokyo.ofc@computer.org

IEEE

revised 5 May 2011