# Enabling the Future Auto with Model-Driven Services

Johannes Helander and Margus Veanes
Microsoft Research, One Microsoft Way, Redmond, WA 98052
jvh@microsoft.com, margus@microsoft.com

## ABSTRACT

*Building systems out of models may be a good approach for better understanding and specifying the behavior of software and its interaction with the physical world. This paper proposes taking the models beyond specification and validation, making models an integral part of any cyber-physical system. Continuous and discrete models can be used to predict and adapt to situations and to build reliable distributed computing systems. Models describing behavior and features of an automobile can be used to dynamically configure an automobile, making a subscription model of features possible.*

## THE AUTOMOBILE AS A COLLECTION OF SOFTWARE SERVICES

A modern automobile consists of a large number of components, each controlled by one or multiple microcontrollers. It is thus reasonable to consider an automobile as a distributed computing platform. The computers interact with the physical world through sensors and actuators. Since the physical world works in real-time also the software must operate in real-time, often with very tight temporal constraints. While actions taken by the individual components must be coordinated, the tight time constraints often necessitate local decision making—after all, communication latencies due to spatial distance is one thing technological advancement cannot do away with due to the underlying physics. To overcome the contradictory requirements, the individual components need to predict the situation around them and act proactively. The authors propose dealing with this software complexity by employing the abstraction of *software services* together with model-driven execution. The services provide a common abstraction for mapping hardware components and features to a homogenous software substrate. The services contain the functionality of the component—typically written as a normal object oriented program—as well as a model that uses stochastic, differential, discrete, and other approaches to model its surroundings—physical and cyber. The models are loosely synchronized with the models of the peer services so as to create a consistent distributed behavior.

## MODELS DESCRIBE FACETS OF BEHAVIOR

A model itself is a simplification of reality. As such each model describes some facet of reality, the component, the automobile, or feature in a way most appropriate for that particular facet. The models relate to functional and non-functional properties of the system, making all of them programmable. The non-functional properties include time, security, reliability, energy consumption, etc.

The composition of the models is **verified**, e.g. using symbolic analysis. Each model is evolved together with the software or hardware it relates to. The models are used to **specify** and **drive development**, to **test** and **validate** the implementation, and to **drive execution**. The *partiture* is a model program that describes the interrelation of functions and their temporal characteristics. The partiture essentially describes a distributed application, a set of functionality of a car—it is analogous to its musical counterpart, the short score that the conductor reads. The instruments are analogous to services and features. The partiture enables individual instruments (features) to participate at appropriate times. The times and relationships are themselves derived through **analysis** and specifications.

## Enabling features as a subscription

Many of the features that a car provides to a user can be viewed as a set of client-side services that interact remotely with online services through the internet. The client can for example request a tune-up for the car that can be performed remotely by an authorized mechanic; the user would only need to physically take the car to the shop if a more serious problem is encountered. The user could also buy additional features for the car, along with their behavioral specification in form of a partiture. Examples of additional features, available in generic hardware and mechanics, more efficiently produced in bulk, but not included in the original package at the time of purchase, include enabling a built-in GPS service and a navigation system, enabling the screen service and downloading movies to be viewed over a long trip, or acquiring 100 more horse powers for a trial period. If a feature is already enabled, there may be new updates available, e.g. an updated navigation map.
A user could sign up the car for an online service, where the health of the car is monitored constantly, or automatic installation of downloads is enabled. A user may want to choose between different providers of e.g. navigation software, in case there are multiple possible third party solutions available. Whether the automobile as a platform is open or closed is a business decision that can be enforced through cryptographic means. For instance, a certified dealer has a cryptographic certificate of the manufacturer that allows signing partitures. In an open environment the user thus has the option of choosing between quality and price. A user could also sign up for online games as part of the entertainment system in the car (for the passengers). Designing such services requires rigorous modeling and model-based analysis of the functionality and their interaction. Interaction between the services is governed by protocols that need to be modeled and tested. Such protocols also provide a way for third parties to provide alternative solutions that disentangle the currently hardcoded dependencies between a specific car model and make and the features available in that car.

## Challenges and milestones

- Modeling techniques: combining discrete and continuous models; combining functional and non-functional property related models and programs; combining structural and behavioral semantics; creating simplified online models for microcontroller use, perhaps updated by

more powerful computers as needed; making it easier to model and validate distributed and real-time systems.
- Programming paradigms: splitting problems into facets and bringing the facets back together in a consistent way; combining object oriented software components with model-driven execution; integration with domain-specific languages.
- Business challenges: monetizing features; increasing manufacturing efficiency; improving energy efficiency through more adaptive and smarter software; increasing safety by improved situation awareness.

### AUTHOR BIOGRAPHIES

**Margus Veanes** is a researcher in the Foundations of Software Engineering group at Microsoft Research. His research interests include model-based software testing, validation and development. He is a co-designer and co-developer of the model-based testing tool *Spec Explorer*. He is a co-author of the book *Model-Based Software Testing and Analysis with C#* and the primary developer of the *NModel* toolkit.
**Johannes Helander** is a researcher at Microsoft with an interest in distributed embedded systems, scalable real-time, consumer-centric security, and context history based prediction and modeling. In the past Johannes worked on a shadow paging database, the Mach operating system, the initial Microsoft Interactive TV, firmware for 3D graphics and the first TCP direct path, a distributed object system for Java, embedded C# with real-time extensions, smart watch software, a component based RTOS, the first Embedded XML Web Services, a remote shell for Windows Vista, the Embedded WS-Management Toolkit, touch-based and an evidence-based trust and security, an auto-adaptive distributed real-time scheduler, and the use of futures for expressing and scheduling parallelism in embedded software.

## Bibliography

1. **Jonathan Jacky, Margus Veanes, Colin Campbell, Wolfram Schulte**. *Model-Based Software Testing and Analysis with C#*, Cambridge University Press, 2008.
2. **Wolfgang Grieskamp**, **Dave MacDonald**, **Nico Kicillof**, **Alok Nandan**, **Keith Stobie**, **Fred Wurden**. *Model-Based Quality Assurance of Windows Protocol Documentation*, First International Conference on Software Testing, Verification and Validation, ICST, 2008.
3. **Johannes Helander, Risto Serg, Margus Veanes, Pritam Roy.** *Adapting Futures: Scalability for Real-World Computing,* Real-Time Systems Symposium, 2007
4. **Johannes Helander, Jürgo Preden.** *Adapting the Auto to a New Tune.* Rio de Janeiro : IEEE, 2006. RTSS 2006 - Workshop on Models and Analysis for Automotive Systems.
5. **Johannes Helander**. *Deeply Embedded XML Communication: Towards an Interoperable and Seamless World*, Proceedings of the 5th ACM international conference on Embedded software, Jersey City, NJ, September 2005.
6. **Johannes Helander**, **Stefan Sigurdsson**. *Self-Tuning Planned Actions: Time to Make Real-Time SOAP Real*, Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Seattle, May 2005.