Softw Syst Model (2009) 8:551–566
DOI 10.1007/s10270-009-0116-5

EXPERT'S VOICE

Model-based design: a report from the trenches of the DARPA Urban Challenge

Jonathan Sprinkle · J. Mikael Eklund · Humberto Gonzalez · Esten Ingar Grøtli · Ben Upcroft · Alex Makarenko · Will Uther · Michael Moser · Robert Fitch · Hugh Durrant-Whyte · S. Shankar Sastry

Received: 2 October 2008 / Revised: 8 February 2009 / Accepted: 11 February 2009 / Published online: 5 March 2009
© The Author(s) 2009. This article is published with open access at Springerlink.com

Abstract The impact of model-based design on the software engineering community is impressive, and recent research in model transformations, and elegant behavioral specifications of systems has the potential to revolutionize the way in which systems are designed. Such techniques aim to raise the level of abstraction at which systems are specified, to remove the burden of producing application-specific programs with general-purpose programming. For complex real-time systems, however, the impact of model-driven approaches is not nearly so widespread. In this paper, we present a perspective of model-based design researchers who joined with software experts in robotics to enter the DARPA Urban Challenge, and to what extent model-based design techniques were used. Further, we speculate on why, according to our experience and the testimonies of many teams, the full promises of model-based design were not widely realized for the competition. Finally, we present some thoughts for the future of model-based design in complex systems such as these, and what advancements in modeling are needed to motivate small-scale projects to use model-based design in these domains.

Communicated by Bernhard Rumpe.

J. Sprinkle (✉)
University of Arizona, Tucson, USA
e-mail: sprinkle@ECE.Arizona.Edu; sprinkle@acm.org

J. M. Eklund
University of Ontario Institute of Technology,
Toronto, Canada
e-mail: Mikael.Eklund@uoit.ca

H. Gonzalez · S. S. Sastry
University of California, Berkeley, USA
e-mail: hgonzale@EECS.Berkeley.Edu

S. S. Sastry
e-mail: sastry@EECS.Berkeley.Edu

E. I. Grøtli
Norwegian University of Science and Technology,
Trondheim, Norway
e-mail: grotli@itk.ntnu.no

B. Upcroft
University of Queensland, Brisbane, Australia
e-mail: ben.upcroft@uq.edu.au

A. Makarenko · M. Moser · R. Fitch · H. Durrant-Whyte
University of Sydney, Sydney, Australia
e-mail: a.makarenko@cas.edu.au

M. Moser
e-mail: m.moser@cas.edu.au

R. Fitch
e-mail: r.fitch@cas.edu.au

H. Durrant-Whyte
e-mail: h.durrant-whyte@cas.edu.au

W. Uther
National ICT Australia, Sydney, Australia
e-mail: William.Uther@nicta.com.au

1 Introduction

One of the defining challenges in robotics in the last decade is the series of DARPA Grand Challenges, focusing on the continued development of theory, techniques, and personnel in the domain of autonomous unmanned ground vehicles (UGVs). Of equal importance to the efforts in control theory, computer vision, and other fields is the importance of software techniques to deliver a dependable executable system. The literature of experimental robotics shows that there are many examples of software-controlled robotics systems, even autonomous systems. However, many advanced

techniques in software modeling and systems modeling have continued to mature since the first DARPA Grand Challenge, and we discuss in this work how our team utilized some of these techniques in our effort with the Sydney-Berkeley Driving Team.

1.1 Scope

Given the complexity of the DARPA Urban Challenge competition, we must be careful to describe the scope of this paper in order that its relevance to the software and systems modeling community is clear. In particular, this paper leaves to other disciplines and publications many topics.

- This paper does not present the design of an autonomous vehicle.
- This paper does not present a description of best-practice algorithms for autonomous vehicles.
- The purpose of this paper is not to describe sensors or actuators that streamline system development.
- This paper does not advocate the use of a particular software methodology, middleware, or architecture.
- This paper does not give technical insight into why any particular team completed, or failed to complete, the DARPA Urban Challenge.

With these notable exceptions to the scientific contributions of this paper, the complexities of the competition, as well as the complexities and features of the system, do provide an intriguing intersection of requirements and technology upon which many members in the software and systems modeling community would enjoy testing various techniques, including theory, software tools, best practices, etc. The purpose of this paper is to describe the techniques, tools, issues, and risks of such techniques, as well as providing some commentary in the analysis section regarding the competition's focus and definition, and its role in the application of these techniques.

- This paper describes how techniques such as model equivalence were used in our entry for the DARPA Urban Challenge.
- This paper describes (in brief) the real-time issues that arise when system components with ill-defined models of computation (i.e., third-party software) are not well-understood, but must be integrated.
- This paper discusses how short-term projects, with deadlines based on ad hoc systems integration, may reduce the potential impact of modeling technology.
- This paper describes how disruptive modeling technologies, with high-risk high-reward use, may be difficult to apply to an already high-risk high-reward project with loosely-coupled group cooperation.

Although we describe portions of our system in great detail, we leave the design justifications to the other papers [2,7,19], and in this paper we instead provide some reflection on lessons learnt from the overall integration, and whether variances in the competition's rules and regulations would have provided the opportunity for impact of more disruptive technologies.

1.2 The DARPA Urban Challenge

The DARPA Urban Challenge was the third race in the series of Grand Challenge events sponsored by the Defense Advanced Research Projects Agency (DARPA) to encourage a groundswell of robotics researchers to build autonomous ground vehicles. In brief, the Urban Challenge aimed to bring autonomous ground vehicles to the city streets, while following traffic laws as defined in the California Driver's Handbook. A full description of the race is provided at DARPA's website, http://www.grandchallenge.mil/grandchallenge, though for archival purposes the reader may refer to [14].

2 Background and previous work

2.1 Providing the system

Solutions to the system require (roughly) three tiers of effort: the problems, the technical issues, and the measures of confidence. We will spend much of this paper describing how modeling aided with the latter two.

The problems to be addressed include localization, sensing, control, obstacle avoidance, path optimization, and others. Each of these problems is its own discipline of study and research, and as such the details of our design is outside the scope of this paper. Nonetheless, each of these problems will be solved by a particular functional description, which is characterized by input/output relationships and an execution cycle. Thus, an overall design lends itself to componentization, where various functional components exchange data and produce updated information for other components.

In Fig. 1, a functional view of a component is provided. In our context, the function of a component is such that a component fires based on an input data set, and produces some output data set. Generically, we state that a component's functional behavior can be expressed as $y = f(x)$, where $y \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$. Thus, the rank of y and x are

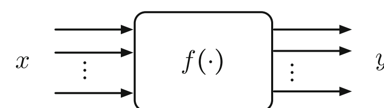


Fig. 1 A functional view of a component

not necessarily equal. Further, the firing of a component (to compute a new y) is generally data driven in our application. Details of component firing parameters, concretized as a model of computation, is a well-studied problem in embedded systems design, and readers are referred to the work in [3,10] for relevant discussions.

Critical-path technical issues to provide answers to the problems of localization, sensing, etc., include stability analysis and robustness of controllers and sensors, functional correctness of algorithms, fault-tolerance, and more. Again, many of these issues have a dedicated community of researchers, but in order to successfully solve these technical issues, we must accompany the technical implementation of our solution with measures of confidence.

Measures of confidence are both best-practices, and new advancements in systems theory, which provide objective information regarding the system's behavior. Examples of these measures are functional equivalence between prototypes and final implementations, regression tests, system integration models, and hardware- and software in-the-loop tests. Critical to our application is the similarity of simulation engines to the vehicle, as we discuss later.

2.2 Model-based systems integration

Previous work in the integration of complex systems shows that a model-based approach can be successful even at large scales. For example, Long and Misra in [11] address the specific productivity increase using the Saturn Site Production Flow (SSPF) modeling language. Since the late 1990s, Boeing developed several iterations of the Bold Stroke control system middleware [15], which was a component-based approach to large-scale systems composition. Using various modeling languages (see for example, ECSL in [13]), systems using Bold Stroke as the control system could be configured, and their runtime files generated in a matter of hours.

Large-scale production systems also benefit greatly from a model-based approach. The Future Combat Systems (FCS) program, with its complexity and rapidly changing requirements, benefits greatly from the use of an integrative strategy [16]. Some of these benefits include the ability to reconfigure the system easily, and regenerating the "glue" that holds much of the system together, while others may include the ability to perform verification or validation of some kind of the system [4]. Each of these examples details a *fielded* system that utilizes model-based technologies.

3 Understanding the domain

The domain of this application lies jointly in the physical devices and algorithms necessary to control and sense, and the software necessary for the computational and communication aspects of the system.

3.1 Object-oriented modeling

There were various object-oriented modeling tasks to be performed throughout the project. These data structures are typically static, and were defined by domain objects such as the number of data points available from a sensor, its operational frequency, etc. In order to facilitate a distributed computational framework, we utilized a middleware solution. Unfortunately, traditional modeling tools did not provide a structural modeling and code generation solution, where here we mean that the generated code would either be interface definitions, or full software specifications, including getter/setter functions and data marshalling.

While it would have been possible to build a domain-specific solution for the purpose of the project, the amount of code required to define these data structures was on the order of 200 lines, and (as we mentioned before), many structures were statically linked to the hardware in use. We utilized the generative programming features of the middleware (after building interface definitions by hand) to generate getter/setter functions and data marshaling behaviors from those human-created interfaces.

3.2 Systems/integrative modeling

Several aspects of the system model, traditionally seen as "low-hanging fruit" for modeling technology, are not significant enough to justify the effort in a project with this short timeline, as the expected changes are not frequent. For example, only a few models of behavior could be easily specified as a state machine. The most common of these was the behavior of the emergency-stop (E-stop), which was required for each team's entry to immediately bring the vehicle to a stop. However, the behavior of the E-stop was prescribed by the rules, and thus, once correctly specified would not be updated. Again, the motivation to formalize this behavior using a state machine modeling tool and then generate code for our specific middleware is low, as that would require significantly more effort than specification of the state machine in traditional software.

Large unknowns in system components, such as laser data, inertial measurement unit (IMU) data, global positioning system (GPS) data, all require custom code to integrate into the system-wide data structures and some component-based delivery system. A model-based approach is appropriate for the integration strategy, but the implementation of the various drivers is best-suited for low-level programming, as memory management was a key factor in both getting the data, and maintaining acceptable performance for the real-time behavior.

Many of these same components that require advanced low-level programming also utilize 3rd party software (or hardware) subject to proprietary restrictions, or are perhaps even restricted by International Trafficking in Arms Regulations (ITAR). Each of these factors means that black-box components will be a norm in the system.

3.3 Model-based prototyping

Finally, although algorithms for high-level behavior are often difficult to express in models, this is not always the case. For example, Simulink is an excellent design and analysis tool, but requires specific hardware to use code generation techniques. Given that the domain of autonomous navigation and control is itself a research domain, the tools, techniques, and analysis necessary are not always known in advance, and thus the development of domain-specific models and tools requires the knowledge of the structure of the generated code, or model, a priori. As such, a great portion of the high-level algorithms were prototyped in software, and while they may now be ready for model-based approaches based on an abstraction of the problem and domain, during the time of the competition they were not. We discuss this tradeoff later in Sect. 6 .

4 Model-based techniques: what was used

Our system utilized several different kinds of modeling techniques, primarily during the design phase. These included the ability to test model equivalence, give measures of software equivalence, to recover models of behavior from data, and to decouple system interaction from software specification. Further, using object oriented models of our vehicle we were able to rapidly simulate the vehicle's behavior, though the accuracy of those simulations in comparison to the physical model was nontrivial to determine. In this section, we discuss each of these issues.

4.1 Model equivalence

In initial phases of the project, many algorithms were prototyped, and models simulated, using well known modeling tools. For example, the initial models of the vehicle kinematics were prototyped in Simulink, and using those models the initial steering and acceleration controllers were developed in Simulink as well. Additional behaviors, such as obstacle avoidance, and waypoint following, were layered upon these models.

Since our final implementation was not going to use these Simulink models at runtime, we created software-based ports of the behaviors of these models, and used the ability of Simulink to substitute software modules for Simulink blocks. These software modules were substituted one at a time, and a set of regression tests run to compare results. This brings to the forefront a structure for testing these modules and comparing their results. To discuss this structure, we talk about a specific component that utilizes several mathematical models for its behavior: the local navigation function. This component used model-predictive control to generate control inputs for velocity and steering angle. The relevance of this component to our approach is that its development lasted the entire length of the competition, thus requiring it to go through the stages of prototype (for proof of concept), analysis and refinement (to approximate the vehicle's behavior), and implementation and rewriting (to move behavior from the prototype simulator to the actual vehicle). This meant that the same mathematical models and software might run in two to three different semantic domains, and definitely run at more than one level of fidelity. We next discuss how we tested the gradual migration of this important component using the technique of *model equivalence*.

Figure 2 shows a component model of a model predictive controller (f_{mpc}) and an associated vehicle model (f_v). Nonlinear model-predictive control (NMPC) problems, in general, consist of the following steps:

1. solve for the optimal control law starting from the state x_k at time k;
2. implement the optimal input $u_k, \ldots, u_{k+\tau-1}$ for $1 \leq \tau \leq N$; and
3. repeat these two steps at time $k + \tau$.

The solution for the optimal control law can be found by formulating a cost function and considering it when

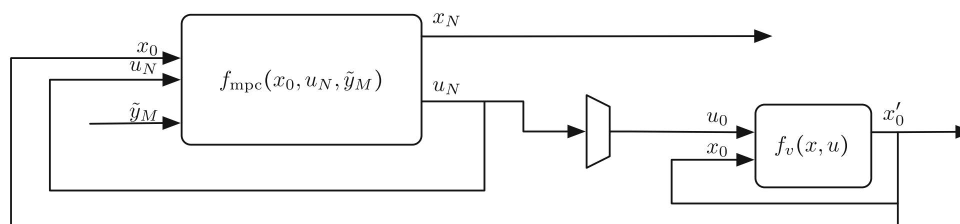


Fig. 2 Component model of a model predictive controller and an associated vehicle model. Note that the model for the vehicle's new position, $f_v(x, u)$, is not necessarily the same model used by the predictive controller for the vehicle's position

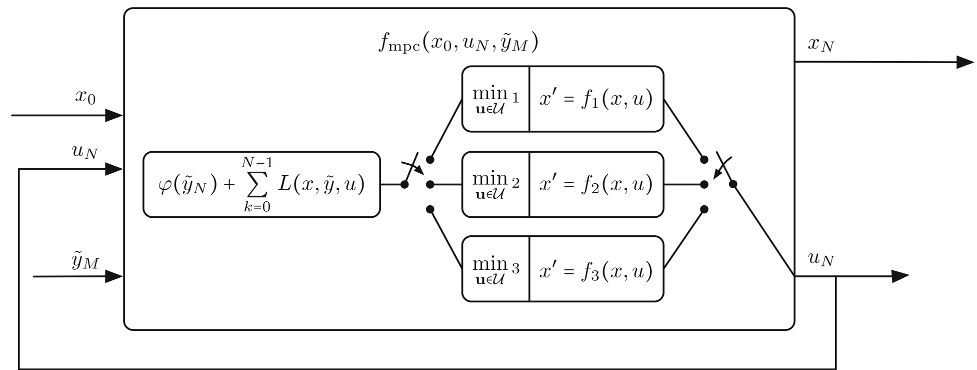


Fig. 3 Run-time choices allowed easy switching between various optimization strategies. An abstraction of the model-predictive controller implementation shows that one of several optimizers might be chosen to perform the selection of control inputs

performing the optimization. As described in [1] it is possible to compose this cost function by using the specific details of the application, and the designers best knowledge of optimal performance of the object or trajectory being tracked. We discuss a software equivalence model for various optimizers in Sect. 4.2.

4.2 Software equivalence testing

When testing the functional behavior of control algorithms, Mathworks's Simulink provides a domain-specific, synchronous execution model in which new algorithms and system models can be developed and analyzed. Included in the MATLAB Toolbox suite is a library of optimization routines, which allow for batch-processes optimization (suitable for synchronous execution of a model).

Such routines[1] generally optimize the cost function

$$J = \varphi(\tilde{y}_N) + \sum_{k=0}^{N-1} L(x_k, \tilde{y}_k, u_k), \tag{1}$$

where φ accounts for the cost associated with the goal at the end of the time horizon, while L accounts for a the cost at each time step in the horizon interval. Usually these functions take quadratic forms as, $\varphi(\tilde{y}_N) \triangleq \frac{1}{2}\left(\tilde{y}_N^{\mathrm{T}} P_0 \tilde{y}_N\right)$, and,

$$L(x, \tilde{y}, u) \triangleq \frac{1}{2}\tilde{y}^{\mathrm{T}} Q \tilde{y} + \frac{1}{2}x^{\mathrm{T}} S x + \frac{1}{2}u^{\mathrm{T}} R u \tag{2}$$

even though L can also include non-quadratic terms as representations of more complex objectives.

The Q, S, and R square matrices (along with additional terms describing constraints) each serve as weighting factors in the cost function (see [18] for details). Regardless of the implementation, there is an outer loop that performs this optimization:

$$\min_{u \in \mathcal{U}} \left(\varphi(\tilde{y}_N) + \sum_{k=0}^{N-1} L(x_k, \tilde{y}_k, u_k) \right). \tag{3}$$

[1] The most widely used is `fmincon`, which is a local minimizer of a general nonlinear function subject to equality and inequality constraints.

There is, then, an interest to determine whether two optimization schemes produce the same u for the same initial conditions u_0, x_0, $\tilde{y}$. Such a software equivalence test can enable confidence for heuristic searches, or can yield concerns that such heuristics may need additional effort in development. It is worth noting that the functional equivalence is tested here, as the behavior in time is sought to be optimized (i.e., to run the vehicle in real-time). Nonetheless, most nonlinear optimization schemes are not suitable for real-time execution, as an upper bound on execution time is difficult to guarantee.

An abstraction of our implementation of the f_{mpc} function is shown in Fig. 3. In our particular example, we had two algorithms from which to choose at runtime to cover the waypoints requested of the vehicle (represented by the desired trajectory, $\tilde{y}$), each of which used variations of gradient descent to optimize the control inputs. One algorithm was more aggressive, as it enabled a wider exploration of the search space, while the other was more conservative with its consideration of the geometric properties of the controller and vehicle behaviors. It is important to note that each of these predictive algorithms *does* require a model of $x_{k+1} = f_v(x_k, u_k)$ with which to forward-predict the x_N that represents the future state of the system (in our case, a vehicle, v) to optimize; each predictive algorithm may use a slightly different model, depending on the fidelity required for rapid and accurate optimization.

At design time, though, we needed to know whether the controllers were solving certain problems correctly. In order to come to this conclusion, we developed a model to check the equivalence of software-based optimizers on certain regression problems. Figure 4 shows this methodology, where rather than selecting *one* optimizer at runtime, we run *all* optimizers and compare their functional output on a static set of initial conditions. Again, note that each of these optimizers requires a model of f such that $x_{k+1} = f(x_k, u_k)$, but that each optimizer in this figure uses the same predictive model as the baseline optimization. This tests the behavior of the *optimizer only* by keeping constant the predictive vehicle model. However, many different vehicle models were used in various portions of the design, and we discuss this in Sect. 4.5.

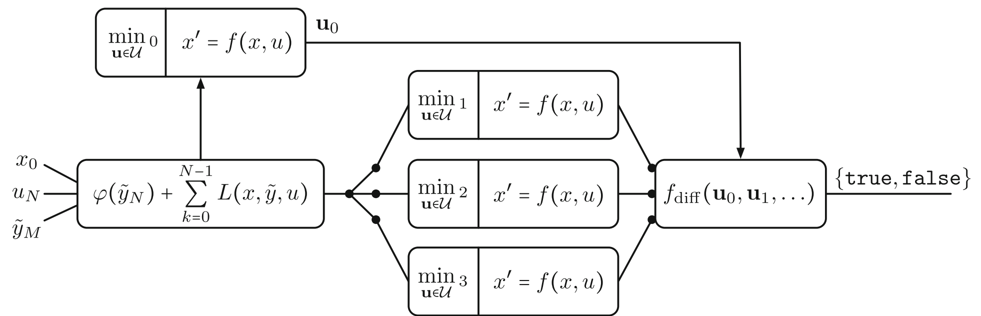


Fig. 4 Design-time methodology allowed equivalence testing between the optimizers. This was important to gain confidence in heuristic approaches, or in improvements to the runtime of the optimizer

Our equivalence tests required full equivalence on our regression tests. That is, $f_{\text{diff}} = \texttt{true}$ (see Fig. 4) meant that the largest scalar element difference between two control vectors, $\max(\mathbf{u}_i - \mathbf{u}_j) = \epsilon$, where $\epsilon \approx 0$. With such stringent tests performed offline, we could select an optimizer based on its runtime performance.

4.3 Recovering models from data

Modeling of the various physical subsystems of the vehicle was necessary for both simulation and control. In general, the equations governing these models were assumed and parameters were determined to fit the models to the available data. In some cases, black-box models were also used, but we will illustrate the process of data collection from the vehicle as used to fit one particular subsystem model: the steering angle.

The Ackermann vehicle model [12] provides an approximation of vehicle motion for vehicles with four tires, and steering control for the front two tires. Although the model allows different angles for the tires, it does not account for mechanical vehicle stability and wear items, such as the transmission slip, and limited slip differential, when accounting for motion.

The model is therefore is a good first-order approximation to vehicle motion: however, for control inputs, and predictive control, it is sufficient to consider an even simpler model—a bicycle model—where the left and right tires are combined into a "virtual" bicycle, with one front and one rear tire at the center of the vehicle's longitudinal axis. Figure 5 shows the Ackermann Model, and bicycle model. The motion equations of a bicycle model are presented in (4) where (x, y) are the cartesian coordinates of the front wheel, v is the speed of the body of the bicycle, ψ is the angle of the body of the bicycle with respect to a fixed frame, δ is the angle of the front wheel with respect to the body of the bicycle and b is the length between front and rear wheels.

$$
\begin{aligned}
\dot{x}(t) &= v(t)\cos(\psi(t) + \delta(t)) \\
\dot{y}(t) &= v(t)\sin(\psi(t) + \delta(t)) \\
\dot{\psi}(t) &= \frac{1}{b} v(t)\sin(\delta(t))
\end{aligned}
\tag{4}
$$

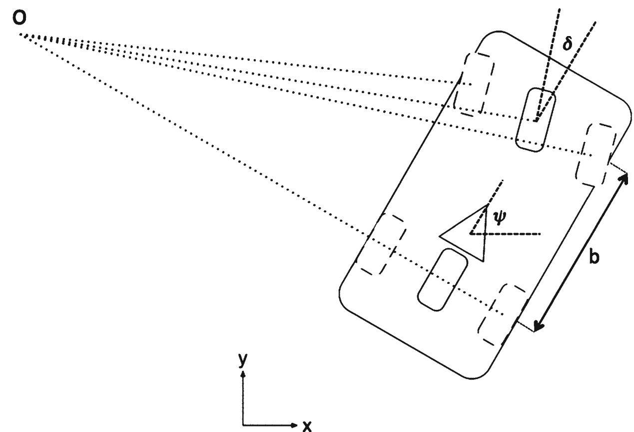


Fig. 5 The Ackermann Model, with a simplified bicycle model, used for control, and predictive modeling of the vehicle's trajectory of motion

The data used to fit the models were obtained from 3 sources: the vehicle's Controller Area Network (CAN) bus, a *NovAtel SPAN* system and the custom actuation system developed by the Sydney-Berkeley Driving Team. The vehicle's CAN bus provided measurements of the internal state of the car such as both pedal positions (accelerator and brake), the independent speed of each wheel, the revolutions per minute of the engine and the gear used at each time step. The *NovAtel SPAN* system provided accurate measurements for position, velocities (linear and angular) and accelerations (linear and angular). The actuation system was used in a tele-operation mode, eliminating human errors during the data acquisition process and allowing a fast and continuous sampling of the steering wheel angle and the signal applied to both pedals. The time synchronization of all the sources was performed by a computer running the *QNX* real-time operating system.

We use δ_t as the angle of the tire (relative to the vehicle's heading angle, ψ), and $\delta_{sw} \in [\delta_{sw_{\min}}, \delta_{sw_{\max}}]$ as the angle of the steering wheel. For most four-wheel vehicles, $\delta_{sw_{\max}} - \delta_{sw_{\min}} \approx 5\pi$.

In order to gather useful and sufficiently rich data, we put the vehicle through several motion plans, including several regular-use scenarios (i.e., road driving by a human). To illustrate the process, we note that the actual steering angle

was not available on the vehicle, but a simple model of the steering was used in which we determine the tire angle from calculating the difference in tire speed between the front tires, left and right. This information is captured from the CAN, as described above.

The estimate is based on the following model:

$$v_l = r_l \omega \quad (5)$$

$$v_r = r_r \omega \quad (6)$$

$$r_r = r_l + C_{lr} \quad (7)$$

$$\omega = \frac{v_r - v_l}{C_{lr}} \quad (8)$$

$$\delta_t = \frac{\omega}{v} \quad (9)$$

where v_l and v_r the respective velocities of the left and right tires, r_l and r_r are the radii of the left and right tires of the angle of the vehicle's turn, C_{lr} is the centerline distance between the two front tires, ω is the rotational velocity of the logical middle tire, and δ_t is the steering angle of the logical middle tire.

In using these models, we can derive parameters for the following approximate model for the tire angle:

$$\delta_{t_{\text{estimate}}} = \Delta_{\text{offset}} + \Delta_{\text{scale}} \delta_{sw} \quad (10)$$

where $\delta_{t_{\text{estimate}}}$ is an estimate of the logical center tire's angle, Δ_{offset} and Δ_{scale} are constants representing the linear offset and scaling factor (respectively), and δ_{sw} is the measured steering wheel angle. This methodology is described in greater detail in [5] and is based on foundational work found in [9]. Figure 6 shows the accuracy of our models graphically.

To derive the parameters, we use offline optimization routines to compare the actual value of the vehicle's position, with our estimate of the vehicle's position using this simplified steering model for steering wheel input control. Measurement errors are minimized by selecting as optimization inputs the Δ_{offset} and Δ_{scale} parameters, to minimize the difference of the actual measured tire angle and $\delta_{t_{\text{estimate}}}$ over the entire dataset.

In similar ways, other and more complex subsystem models were determined and used for simulation and control of the vehicle. These areas are described in Sects. 4.4 and 4.5.

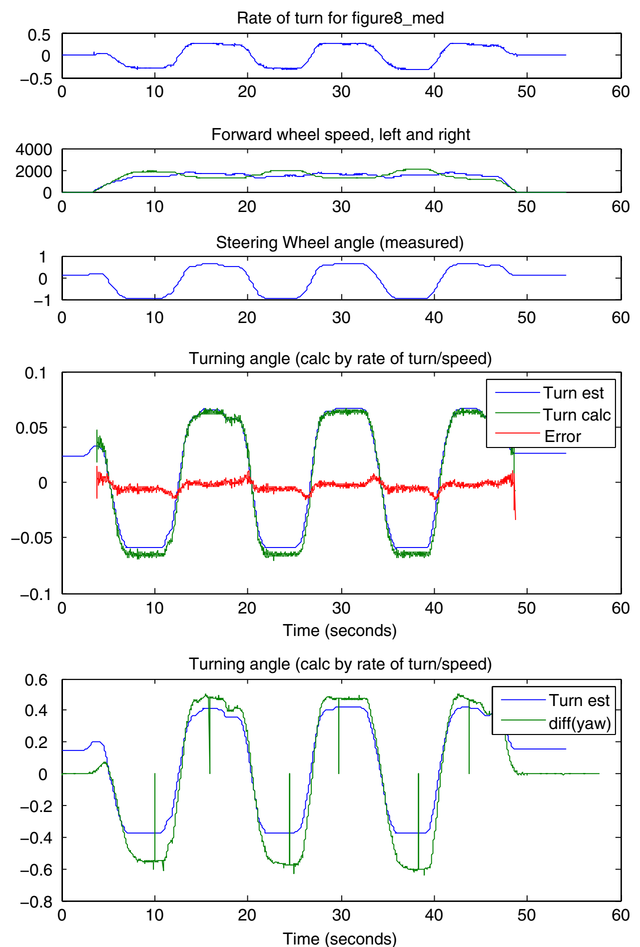


Fig. 6 A model of the car turn angle, based on steering wheel angle measurements. The top graph shows the rate of turn (calculated in (8)). The next graph displays the front wheel speeds. Note here that the nonlinearities in the rack and pinion are evident at the most significant steering wheel angle, even though the wheel is not fully locked. The next graph shows the measured steering wheel angle, based on CAN data. The final two graphs compare the estimated value with two calculated turn angles: that of the values in the top graph divided by the speed, and the bottom graph reflecting yaw rates gathered from the differential GPS unit

4.4 Vehicle model and environment simulation

This work relied on open-source software for much of the physical world simulation, including realistic rigid-body dynamics of various vehicles in concert with sensors for those vehicles, and simulated terrain information. We utilized the Gazebo simulation platform, which is part of the Player/Stage software suite [6]. Gazebo allows three-dimensional physical interaction and control of compositionally created (i.e., object-oriented) objects in space. Through the Open Dynamics Engine (ODE), Gazebo simulates rigid-body dynamics of the compositionally created models.

What makes Gazebo a particularly interesting—and applicable—software platform for physical model simulation is its ability to provide alternate perspectives of the simple models using a suite of included sensors, as well as provisions for customized sensors. These sensors use ray tracing from the sensor models to retrieve distance information of other objects, thus providing realistic data that reflected changes to the position, orientation, and velocity of the simulated vehicle in its simulated environment.

Gazebo provides a mechanism to create vehicles, sensors, environmental features, and visualizations through the object-oriented composition of various system provided

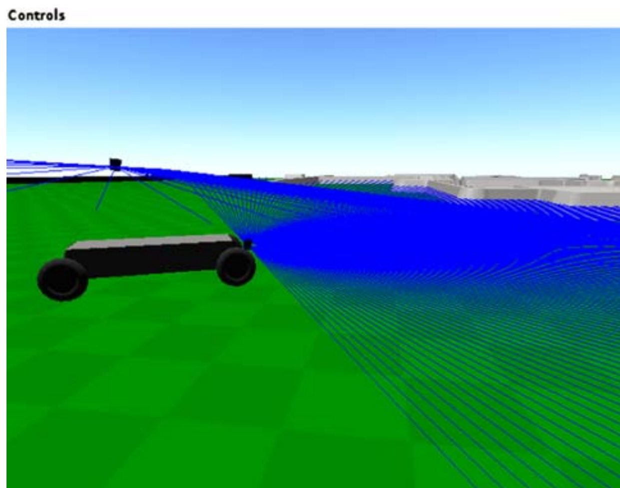


Fig. 7 A rendering of the vehicle with both lasers showing their ray trajectories

objects. Because the software emerged from members of the robotics community, the software implementations of various sensors (such as the Sick LMS) were directly compatible with the hardware purchased for the vehicle. An example of how a vehicle is created, and rendered, in Gazebo is presented in Fig. 7.

The trajectories shown in Fig. 7 represent the configured laser distance returns for the hardware, so the presence of obstacles and even other vehicles in the simulation environment are detected through the rendering of the model. With specification of exact position/orientation of sensors on the simulated vehicle, we could test issues of obstacle detection, vehicle avoidance, etc., without using the physical vehicle. However, an important question to answer is: *is simulation using Gazebo sufficiently equivalent to driving the physical hardware?* This is appropriate even for kinematic or approximate models we use elsewhere in the design, and such questions can only be answered with hardware-in-the-loop. We discuss how to approach such an equivalence question next.

4.5 Physical model: sufficient equivalence

Several models for the motion of the vehicle for given control inputs ($x_{k+1} = f_v(x_k, u_k)$) were used at varying fidelities throughout the project. As already discussed, a kinematic model was used for the model predictive controller, rigid-body dynamics were used in the Gazebo simulation, and of course the actual vehicle's model, which is highly nonlinear, dependent on many environmental parameters and nonlinear components (e.g., the torque response of the engine and transmission at different speeds), and thus unsuitable for characterization through analysis of its physical construction.

However, many of the functional components rely on the tuning of parameters, or automatic use-based optimization in order to operate correctly on the measure of confidence platform—the actual vehicle. Since it is infeasible to perform all of this tuning on the vehicle due to the number of components and vehicle availability, it is important to have some measure of confidence that the various models for simulation are *sufficiently* equivalent to the physical platform. This is described in Fig. 8.

One technique to achieve a measure of confidence is to use the same techniques used on the optimizer in Sect. 4.2 by providing a known set of inputs to several vehicle simulation models, as well as the actual vehicle, to assess equivalence. The system's implicit nonlinearity precludes a transfer function based on impulse-response, though linear approximations of the model (as discussed in Sect. 4.3) are certainly an option.

Specifically, the measure of confidence is the array of vectors, $\Delta\mathbf{x}_1, \ldots$, where each element in the vector is the difference to the ground truth (f_v). The measure of confidence is inversely proportional to the magnitude of each vector element. Significant errors for particular models will point to a particular physical model abstraction that is not valid. This technique is shown in Fig. 9. The ground truth model, which is the vehicle, is represented by f_v. Various models used throughout the design included kinematic models in Matlab ($f_{v_{\text{mat}}}$), rigid body simulation in Gazebo ($f_{v_{\text{gaz}}}$), predictive models used in the controller ($f_{v_{\text{mpc}}}$), as well as other models not shown here. In order to make sense of the comparison of these functions, it is not enough to do an equivalence check. Interesting difference functions, which are out of the scope of this paper, could provide indications that certain models performed well under velocity inputs, but not steering inputs, while others performed well vice versa. Thus, the expected output should be a vector of differences to the ground truth,

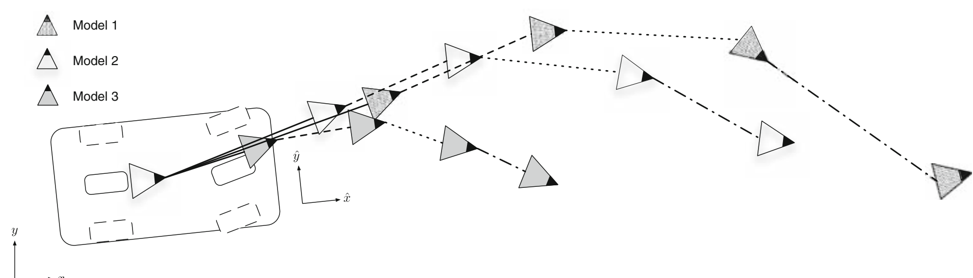


Fig. 8 Position of the center of gravity after time-based injection of control inputs may be drastically different, depending on which model is used to simulate vehicle trajectory

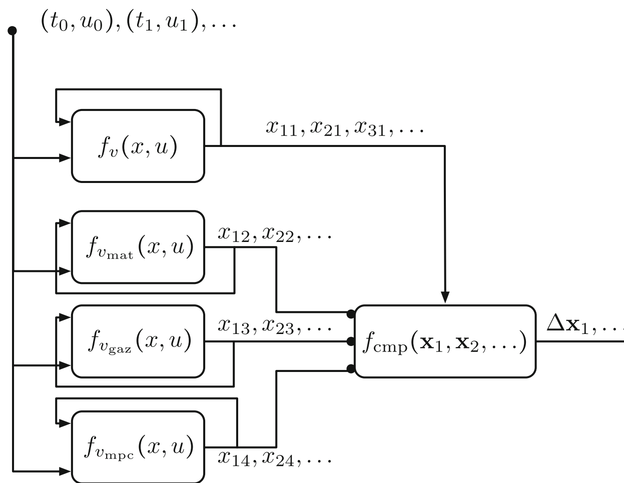


Fig. 9 Comparing several vehicle models with the ground truth (actual vehicle) to assess sufficient equivalence

subject to further analysis. The smaller these delta functions, the more confidence in the various vehicle models. We leave this complex analysis and representation of the actual data captured in our runs to other publications.

4.6 Software models

We employed a component-based design methodology to produce our integrated system. This component-based philosophy required all executing software to be housed (or wrapped) in a software component that required and/or provided certain interfaces. These interfaces were language-independent datatypes, and encoded into the Internet Communication Engine (ICE) format.

So, regardless of the language of implementation, or the philosophy of implementation, the input/output relationships of any component could be quickly modeled as providing/requiring relationships. The semantics of the execution of each component was left up to the component design, and could not generally be controlled using runtime configuration.

This gave great flexibility to each component designer. Some components operated using a dataflow model of computation, where any input was readily transformed into an output, whenever input tokens came in on the required interface. In this case, the criterion for firing is any new token on the input. Some components had no interface inputs at all, as they were passing along data from hardware, or from a simulation. These components were also dataflow based, in that they produced new data as soon as they got it.

For components with more than one input, however, the criteria for firing becomes more interesting. Should the required interfaces be held to an ordered, blocking read (block on `i1`, then block on `i2`, then fire), or should a new token on either input allow firing (perhaps with the previous value from the other input)? Should there be a timeout for this blocking read, and/or should the timeout be the same for all required interfaces?

It was these components with more than one input (and sometimes more than one output) whose specification introduced complexity to understanding the composite model of computation. After connecting the various data paths, it is unclear, at best, how the system operates. Is each component executing when any data value arrives, only at certain data values, through a complex semaphore, always at a certain time?

Clearly the composite model of computation emerges from the model of computation for individual components. This heterogeneous approach led to significant confusion when developing new components, as a common model was not utilized. In defense, however, developing this common model would detract from efforts in passing competition milestones.

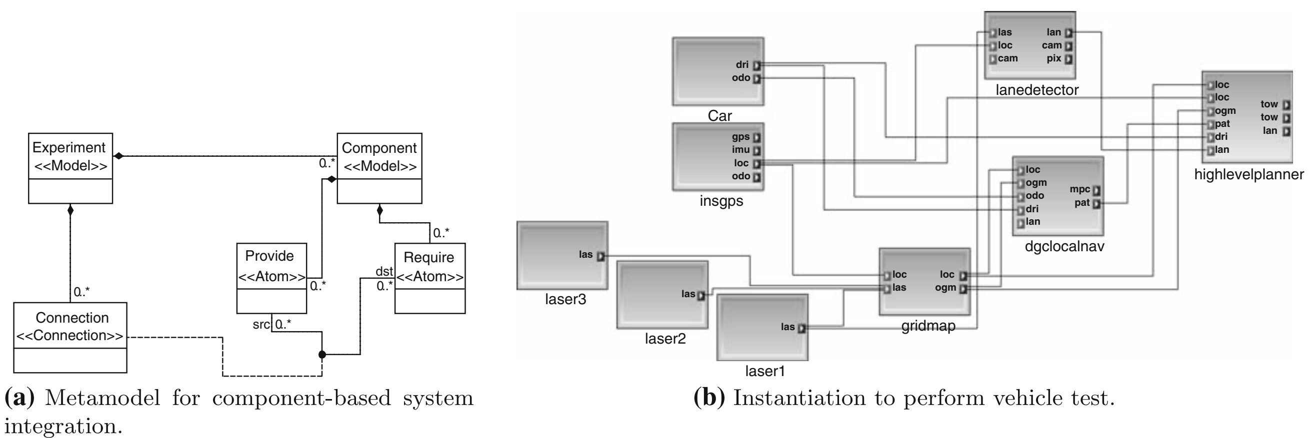
Ongoing research by team members involves the explicit timing of these components and more orderly operation based on strict operation of components according to a well-defined model of computation. For example, periodic timing of components, rather than a pure data-driven execution of components. Such work promises to reduce the variability of the entire system when executed on heterogenous distributed hardware.

Regardless of the implementation of each component, the interdependency of components on each other plays a key role in their assembly at runtime, and at design time. In order to see this interdependency, a metamodel can be used to describe the various types and associations permitted when creating a simulation, or a run-time experiment. Notably, our middleware was robust in that a change in configuration was all that was required to run with actual data from a sensor, or data from regression tests, or data from another component.

Figure 10a shows a metamodel for a graphical language allowing composition of various components. This metamodel is ultimately simple, but provides the ability to create interconnection diagrams such as those in Fig. 10b to abstract the overall system integration. It is important to have a high-level perspective of these models for multiple reasons:

- to explain to new team members the overall structure;
- to see points of “cutting” where new components can be inserted;
- to verify that all “required” interfaces are provided by some other component; and
- to understand the startup order of the system, based on its dependencies.

In fact, the final two points are not as important, because (as we discuss in Sect. 3) the components are fairly static,



(a) Metamodel for component-based system integration.

(b) Instantiation to perform vehicle test.

Fig. 10 **a** A metamodel for a graphical language allowing composition of various components. **b** An example model, where various components connect the provided and required interfaces

and holistic simulations can be scripted to run without much user intervention. Despite these mitigating factors, there is significant potential impact for modeling (especially generative techniques) in this particular area, as we discuss in Sect. 7.

Although these models were developed early in the project lifecycle, they were not used until after the competition (in related research). The reason for this, in addition to the reasons previously mentioned regarding the relatively static nature of components for the composition, is that effort required to integrate the models into the methodology of the team was substantial, and detracted from the necessary technical work. The tradeoff, then, was that the software was developed more rapidly (by individuals), but less accessible to new team members because both its development and integration were done at a low level. The alternative, to spend extra time on high-level techniques that would improve new team members' ability to contribute in later stages of the competition, would jeopardize meeting the demanding competition deadlines. To make this point clear, we now move the discussion to how these deadlines impacted our use of models in the system.

5 The impact of deadlines

The deadlines in the DGC3 were actually the deadlines for components of the DGC competition, not for the standalone medium-scale systems each team could produce. Thus, DARPA's primary motivation was not necessarily to produce the best individual components (i.e., vehicles), but rather to produce the best collection of components that could operate within the final competition site. This subtle difference actually impacts the technical deliverable and confidence measures of the participants in the competition.

5.1 Team and system requirements

Table 1 details the schedule for a team competing in the Urban Challenge, and Fig. 11a shows elements of this schedule graphically. The major deadlines for consideration by the team are the video submission, site visit, and final event. In each of these deadlines, the measure of confidence is *the performance of the autonomous vehicle*. In essence, the full development of an autonomous platform must be completed less than 11 months after the project kickoff,[2] and rudimentary software performing a small set of tasks must exist by this time.

DARPA referred to this set of requirements as *basic navigation*, and the small set of tasks included:

- navigation by waypoints;
- avoidance and passing of a stalled car; and
- demonstration of emergency stop capability.

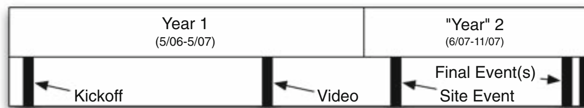
While these tasks are certainly not trivial, they *can* be developed independently from more advanced behavior which must be shown during the site visit, which was generally held around 2–3 months after the video was submitted. Some examples of this advanced behavior, which was called *basic traffic* includes:

- items from the video;
- vehicle stays in lane;
- vehicle departs from and returns to lane; and
- vehicle can perform u-turn.

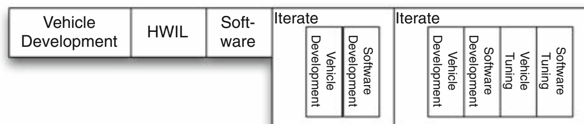
[2] For teams who hoped to obtain funding from DARPA for purchasing a vehicle, this time actually fell to around 8 months to await notification of funding or not.

Table 1 Team schedule (from http://www.darpa.mil/grandchallenge/)

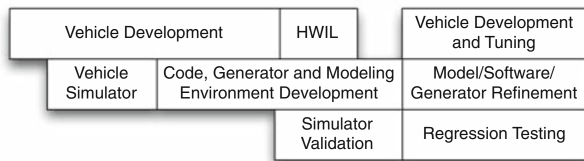
Date	Activity/event	Location
May 1, 2006	Urban Challenge Program Announcement	
May 20, 2006	Participants Conference	Reston, VA
April 13, 2006	Submission of Video	Online
May 10, 2007	Site Visit Selection Announcement	
June 11–July 20, 2007	Site Visits	Performer Site
August 9, 2007	Semifinalist Announcement	
	Location Announcement	DARPATech, Anaheim, CA
October 24–25, 2007	Teams Arrive	Victorville, CA
October 26–31, 2007	National Qualification Event	Victorville, CA
November 3, 2007	Urban Challenge Final Event	Victorville, CA



(a) The DARPA Urban Challenge milestones.



(b) A schedule in line with system-integrator contracts.



(c) A schedule favoring model-based design, but violating milestones.

Fig. 11 Various schedules to complete the DARPA Urban Challenge. **a** The timelines for completing the DGC, as required by DARPA (these are provided in more detail in Table 1). **b** If work was contracted to a system integrator, regular checkpoints for progress are used to guarantee that development is making regular progress. However, in order to show progress in timelines, suboptimal decisions must be made, and then undone in later stages of delivery. **c** An approach that is decoupled from the physical platform provides parallel development of software, models (or modeling environments), but at the cost of delaying vehicle delivery, thus delaying progress milestones

Clearly, performance based on these criteria would also satisfy that of the video. The proximity of the deadlines, however, and the fact that failure to pass the video portion results in removal from the competition, may incline teams to focus on those details without regard to those of the site event.

5.2 Solved and unsolved problems: erosion of motivation

The tasks for the video (basic navigation) are essentially *solved* problems, where little if any research is necessary. The problems for the site event, though, represent an opportunity to develop new and innovative approaches to these problems, which (by their definition) subsume the video requirements.

What motivation exists, then, to develop innovative techniques either in the algorithms for system behavior, or modeling and analysis tools to build such systems? The risk is great, in that the deadline for basic navigation is based on known deadlines and capabilities, so there is a tendency to continue to approach the problem by looking at the new requirements at each phase, and introducing new software. This iterative approach is generally scalable, except that the measure of confidence is a single vehicle, and thus testing is limited by physical constraints.

5.3 Deadline inversion

Again, this is quite a subtle issue, but it can mean that the immediacy of the next deadline to show measures of confidence will trump a systematic approach to providing measures of confidence for the most important issue at hand: correct functional behavior of the scientific advancements of the *problems*. In other words, immediate deadlines focus work on the functional behavior *at all times* and *with the vehicle as the primary testbed* since the vehicle's behavior is the measure of confidence. This comes at the cost of a principled system integration strategy, since the development of such strategies—although they have many advantages in rapidly reconfiguring the system, allowing untrained users to experiment with the system, and formalizing the interactions of system components—will replace efforts that could have been spent on the functional behavior.

5.4 Responsibility by the organizers

We acknowledge that in order to put together a *competition*, DARPA must put together a running system (of many vehicles) where individual components behave properly.

Although many of our above points are critical to the organization of the competition, the alternative is to either

a. allow vehicles with no (realistic) hope of winning to continue in the competition out of courtesy, but eliminating them just before they appear in the final event, thus wasting the time and efforts of those teams, or
b. allow any vehicle (including perhaps vehicles that are unsafe) to participate in the final event, perhaps endangering the efforts of others.

It is clear, then, that the organizers have amortized the selection process out of a genuine desire to have a safe competition, where objective measures are used to determine competition. However, we believe that in order to provide that safe and fair venue, one cost was a decreased motivation for the development of innovative techniques for system integration. These techniques tend to mature later in the design cycle, though with a proven record for rapid configuration (see [8, 18, 20] and [17] for examples where high-level autonomous software operated correctly on a testbed with which the developers had no physical contact, but only a reliable simulator).

6 Reflections

Upon completion of this project, and discussions with other researchers in model-based technologies, it is clear that some of the foundational research currently underway in the community deserves some mention.

6.1 Philosophies of modeling and our effort

There are several design philosophies that are radical new approaches to developing systems. We address each of these with respect to this application.

Everything is a model. This is a natural outgrowth of the philosophy "everything is an object" from object-oriented techniques, and to some degree this materialized for our effort, in the same way that it materialized in OO. To the greatest extent, we developed several component models for the behavior of our system, and those models interacted with one another through message and data passing. However, we did not continue a model-based refinement past the specification of the component interface, and instead depended on general-purpose languages to provide the functional behavior of those components. We address this next.

Model everything. This philosophy does not quite restate that of "everything is a model," as in this case we (the developers) are called to create a model of everything, not treat everything as if it has some model. As we pointed out previously, our implementation was not carried out through widespread modeling, and there are a few reasons for this:

– the diverse team of experts in robotics, computer vision, software, and control were not all familiar with even software modeling techniques;
– the operating environment of real-time behaviors required many components to run on a real-time operating system with limited tool support;
– the behavior of many components is best-specified using general-purpose techniques, especially the advanced control algorithms used.

In addition to these reasons, there was the issue of third party hardware and software, which ran as a black-box for many devices. This means that we had many behaviors which we understood functionally, but had no control over its behavior.

If I did not model it, it is not in the system. For small-scale systems, such as a simulation of a system, or even medium-scale systems where specialized hardware is available to interface with modeling tools, this approach is quite appropriate. In our case, our budget did not permit an integrated modeling and hardware solution for various sensor devices and computational devices. This implied that various hardware used to acquire data (such as serial cards for the laser rangefinders) would need to be interfaced via low-level drivers. The complexity of using a software modeling technique to make vendor-specific calls to a device on the bus of a computer running a real-time operating system is tremendous.

While it is not impossible to produce this as a model-based solution, we argue that in our case such investment in labor would not be well spent. Thus, we had to accept that major portions of our system would require low-level software interaction. We were able to abstract this interaction on the highest level as functional components, but there was no escaping the effort to write the drivers.

All code used is generated code. Many integrated systems follow this philosophy, and it is common for middleware implementations to follow it as well. To some degree, we used this philosophy to produce our data structures and relevant getter/setter methods. However, as we mentioned previously, we eventually had to provide some functionality not easily produced using state charts or sequence diagrams.

Again, this is not to say that with a solution in hand, we could not provide a model-based tool that synthesized code to perform these duties: the issue was that we did not know what the code needed to look like in the first place. Having written it once, spending time to make a generic interpreter to synthesize code for similar scenarios would have been an interesting solution, but would have endangered continued

participation in the competition. However, it does provide a useful perspective for future work, as we discuss later.

6.2 The promise of model-based methodologies

Since the inception of model-based methodologies and domain-specific languages, several advantages have been put forth, including:

- rapid redeployment after a design change;
- verification and analysis using models;
- domain users building models, not programming experts; and
- raising the abstraction of specification.

Even with solid effort by modeling experts, however, most of these promises failed to materialize. The question then, is: *does this failure to materialize speak to limitations of the model-based methodologies, limitations of the domain, or perhaps some other factor?*

Our perspective is that each of these promises faced almost certain failure in this *competition*, though not necessarily in this domain. For example, most teams in the competition had a fairly static design due to the rigorous platform-based proofs required by the organizers. This meant that the likelihood of major design changes was fairly low compared to, say, a website design. For the design changes that *did* occur in our project, we absorbed each of these by using a component model for system functionality, allowing rapid reconfiguration by connecting various component interfaces at runtime.

We next address a major promise of domain-specific modeling: that domain users, rather than software experts, create system implementations through modeling. For some portions of our implementation, this did materialize (e.g., using Simulink, as discussed in Sect. 4.1). In the end, our implementation platform did not support such executable models (and we did not use compatible hardware with MATLAB's Real-Time Workshop), and to create a model-integrated program synthesis solution might have jeopardized further participation in the competition.

Finally, we ask whether we were able to raise the abstraction of our specification. In fact, this was our most obvious success, in that the high-level behavior of the system could be succinctly expressed as functional components operating with data dependencies. As such, the system design resembled a large block diagram (as seen in Fig. 10b). However, we spent very little of our time specifying the system on this high level, and almost all of it in the low-level functional behaviors of each component. Nonetheless, without the high-level specification of the system, we would have lost time in design, testing, and runtime, if we did not have the flexibility to reorganize the system rapidly.

7 Outlook to the future

As we look to the future of model-based design for heterogenous systems, we believe that this domain is ripe for impact by the modeling community. Many system specifications are never made, and their behavior emerges from ad hoc integration strategies. Many functional behaviors are re-expressed time and again by new users who are unable to take advantage of code reuse techniques. Many control and robotics experts are comfortable with low-level software, but cannot perform any rigorous analysis or verification of their integrated system as there is no high level specification or model. Many experts in this domain lose tremendous time struggling with the integration of heterogenous tools, where the semantics are unclear or data transformations are required.

Each of these areas of impact, as well as others not mentioned, present a clear benefit that can be addressed by those in the modeling community by better understanding the domain of these kinds of problems. However, there is the substantial barrier of demonstrating that model-based approaches will in the end provide some benefit. Some domain power users will be reticent to abandon their techniques for new ones that have not yet proved themselves. Such is a reasonable goal for model-based design researchers: to demonstrate that the benefit of using model-based design is not always enhanced productivity, but may include allowing rapid reuse by others, or reducing investment by a new user of learning to use their software or system, or providing analysis and simulation capabilities that did not yet exist.

Acknowledgments This work was due to the tremendous effort of the Sydney-Berkeley Driving Team, which entered the DARPA Urban Challenge in 2006. Team members who performed work tangential to that described in this paper include Alen Alempijevic, Ashod Donikian, Todd Templeton, Eric Chang, Pannag R. Sanketi, David Johnson, Jan Biermeyer, Vason P. Srini, Christopher Brooks, Mark Godwin, and many others who donated their time and efforts. The Sydney-Berkeley Driving Team was supported in part by Rio Tinto, Komatsu, Chess at UC Berkeley, Toyota, ZeroC, and Advantech. Additional in-kind support was provided in the form of discounts on equipment from the following manufacturers: SICK, NovAtel, and Honeywell. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

Open Access This article is distributed under the terms of the Creative Commons Attribution Noncommercial License which permits any noncommercial use, distribution, and reproduction in any medium, provided the original author(s) and source are credited.

References

1. Allgöwer, F., Zheng, A.: Nonlinear Model Predictive Control. Progress in Systems and Control Theory, vol. 26. Birkhäuser Verlag, Basel (2000)

2. Basarke, C., Berger, C., Rumpe, B.: Software and systems engineering process and tools for the development of autonomous driving intelligence. J. Aerosp. Comput. Inf. Commun. **4**(12), 1158–1174 (2007)
3. Brooks, C., Lee, E.A., Liu, X., Neuendorffer, S., Zhao, Y., Zheng, H.: Heterogeneous concurrent modeling and design in java (Volume 1: Introduction to Ptolemy II). Technical Report UCB/EECS-2008-28, EECS Department, University of California, Berkeley (2008)
4. Dubey, A., Nordstrom, S., Keskinpala, T., Neema, S., Bapty, T., Karsai, G.: Towards a verifiable real-time, autonomic, fault mitigation framework for large scale real-time systems. Innov. Syst. Softw. Eng. **3**(1), 33–52 (2007)
5. Eklund, J.M., Korenberg, M., McLellan, P.: Nonlinear system identification and control of chemical processes using fast orthogonal search. J. Process Control **17**(9), 742–754 (2007)
6. Gerkey, B., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics, pp 317–323. ICAR 2003
7. Herpin, J., Fekih, A., Golconda, S., Lakhotia, A.: Steering control of the autonomous vehicle: Cajunbot. J. Aerosp. Comput. Inf. Commun. **4**(12), 1134–1142 (2007)
8. Keviczky, T., Borrelli, F., Balas, G.J.: Decentralized receding horizon control for large scale dynamically decoupled systems. Automatica **42**(12), 2105–2115 (2006)
9. Korenberg, M.J.: A robust orthogonal algorithm for system identification and time-series analysis. Biol. Cybern. **60**, 267–276 (1989)
10. Lee, E.A., Zheng, H.: Operational semantics of hybrid systems. In: Proceedings of Hybrid Systems: Computation and Control (HSCC), LNCS, vol. 3414, pp. 25–53. Springer, Berlin, 2005 (Invited Paper)
11. Long, E., Misra, A., Sztipanovits, J.: Increasing productivity at Saturn. Computer **31**(8), 35–43 (1998)
12. Nebot, E.: Navigation system design. Lecture Notes, May 2005. Center of Excellence for Autonomous Systems, University of Sydney, Australia
13. Neema, S., Karsai, G.: Embedded control systems language for distributed processing. Technical Report ISIS-04-505, Vanderbilt University, Institute for Software Integrated Systems, 2004
14. Seetharaman, G., Lakhotia, A., Blasch, E.: Unmanned vehicles come of age: the DARPA Grand Challenge. Computer **39**(12), 26–29 (2006)
15. Sharp, D.: Avionics product line software architecture flow policies. In: Proceedings of the 18th Digital Avionics Systems Conference, vol. 2, pp. 9.C.4-1–9.C.4-8, 1999
16. Sharp, D.: Hybrid and embedded software technologies for production large-scale systems. In HSCC ’02: Proceedings of the 5th International Workshop on Hybrid Systems: Computation and Control, pp. 1–2. Springer, London (2002)
17. Sprinkle, J., Ames, A.D., Eklund, J.M., Mitchell, I., Sastry, S.S.: Online safety calculations for glideslope recapture. Innov. Syst. Softw. Eng. **1**(2), 157–175 (2005)
18. Sprinkle, J., Eklund, J.M., Kim, H.J., Sastry, S.: Encoding aerial pursuit/evasion games with fixed wing aircraft into a nonlinear model predictive tracking controller. In: Conference on Decision and Control, 2004
19. Upcroft, B., Makarenko, A., Moser, M., Alempijevic, A., Donikian, A., Uther, W., Fitch, R.: Empirical evaluation of an autonomous vehicle in an urban environment. J. Aerosp. Comput. Inf. Commun. **4**(12), 1086–1107 (2007)
20. Waydo, S., Hauser, J., Bailey, R., Klavins, E., Murray, R: UAV as a reliable wingman: a flight demonstration. IEEE Trans. Control Syst. Technol. **15**(4), 680–688 (2007)

Author Biographies

Jonathan Sprinkle is an Assistant Professor of Electrical and Computer Engineering at the University of Arizona. Until June 2007, he was the Executive Director of the Center for Hybrid and Embedded Software Systems at the University of California, Berkeley. In 2006–2007, he was the co-Team Leader of the Sydney-Berkeley Driving Team, a collaborative entry into the DARPA Urban Challenge with partners Sydney University, University of Technology, Sydney, and National ICT Australia (NICTA). His research interests and experience are in systems control and engineering, through modeling and metamodeling, and he teaches in controls and systems modeling. Dr. Sprinkle is a graduate of Vanderbilt University (PhD, MS) and Tennessee Technological University (BSEE). e-mail: sprinkle@ECE.Arizona.Edu

J. Mikael Eklund is the Director of Electrical and Software Engineering Programs and an Assistant Professor on the Faculty of Engineering and Applied Science at the University of Ontario Institute of Technology. He received his Ph.D. from Queen’s University in 2003 in Electrical and Computer Engineering. His research areas include Autonomous Systems (Robotic vehicles, smart sensors for assisted living), nonlinear system identification and control, and medical image processing. From 2003–2006 he was a Visiting Postdoctoral Scholar at the University of California, Berkeley. e-mail: Mikael.Eklund@uoit.ca

Humberto Gonzalez was born in Chile in 1981. He received the B.S. and M.S. degrees in electrical engineering from the University of Chile, in 2005. Currently he is a graduate student in the department of Electrical Engineering and Computer Sciences at University of California, Berkeley. His research interests are in applications to robotics of nonlinear and optimal control. e-mail: hgonzale@EECS.Berkeley.Edu

Esten Ingar Grøtli received his MSc degree from Department of Engineering Cybernetics at Norwegian University of Science and Technology in 2005, and is currently pursuing a PhD degree at the same department. The master program included one year at the Institute for Systems Theory and Automatic Control, University of Stuttgart (2003/2004). In 2006/2007 he visited Professor Shankar Sastry's group at UC Berkeley. His research interests include control of mechanical systems in theory and applications. e-mail: grotli@itk.ntnu.no

Ben Upcroft is a Senior Lecturer in Mechatronics at the University of Queensland. He completed his undergraduate degree in Science with Honours and continued a PhD in ultracold atomic physics in 2003. Throughout his PhD, Ben had a keen interest in robotics, which led him to a Postdoctoral position (2003) at the ARC Centre of Excellence for Autonomous Systems, School of Aerospace, Mechatronics, and Mechanical Engineering, University of Sydney. He has run major industrial projects involving autonomous aircraft, offroad and urban vehicles, and network communications. Ben currently focuses on computer vision aided localization and navigation for autonomous ground vehicles. e-mail: ben.upcroft@uq.edu.au

Alex Makarenko is currently the ARC Postdoctoral Fellow at the Australian Centre for Field Robotics. He received his Ph.D. in Mechanical, Aeronautical and Mechatronic Engineering from the University of Sydney, the M.Sc. in Aeronautics and Astronautics from MIT, and B.S. in Mechanical Engineering from Rensselaer Polytechnic Institute. His current interests lie in the areas of distributed inference in human and sensor networks, robust vehicle operation, and reusable robotic software. e-mail: a.makarenko@cas.edu.au

Will Uther received his B.Sc. in 1995 from the University of Sydney, and his Ph.D. in 2002 from Carnegie Mellon University. He joined National ICT Australia in 2003 and is currently a senior researcher there with a conjoint appointment at the University of New South Wales. Before working with autonomous urban vehicles, Dr. Uther was heavily involved in the Four-Legged League of the RoboCup international robotic soccer competition. e-mail: William.Uther@nicta.com.au

Michael Moser is currently pursuing a M.Sc. in Mechatronics at University of Sydney. He received a degree as Dipl. Ing. (FH) (comparable to a B.Sc. with Honours) in Applied Physics from University of Applied Sciences Weingarten in 2001. During 2006-2007 he was employed as a Technical Officer at the Australian Centre for Field Robotics. He was the lead technical engineer of the Sydney-Berkeley Driving Team. His current research interests are in automated multisensor registration and data fusion. e-mail: m.moser@cas.edu.au

Robert Fitch is a research fellow with the Australian Centre for Field Robotics at the University of Sydney, Australia. Previously, he held a research position at National ICT Australia (NICTA) in Sydney. He received the Ph.D. in Computer Science from Dartmouth College in 2004 and the B.A. from Oberlin College in 1996. His research interests include motion planning, distributed planning and control, hierarchical reinforcement learning, and modular robots. e-mail: r.fitch@cas.edu.au

Hugh Durrant-Whyte received the B.Sc. in Nuclear Engineering from the University of London, U.K., in 1983, and the M.S.E. and Ph.D. degrees, both in Systems Engineering, from the University of Pennsylvania, U.S.A., in 1985 and 1986, respectively. From 1987 to 1995, he was a University Lecturer in Engineering Science, the University of Oxford, U.K. and a Fellow of Oriel College Oxford. Since 1995 he has been Professor of Mechatronic Engineering at University of Sydney. His research work focuses on robotics and sensor networks. He has published over 350 research papers and has won numerous awards and prizes for his work. e-mail: hugh@acfr.usyd.edu.au

S. Shankar Sastry is currently the Dean of Engineering at University of California, Berkeley. He received his Ph.D. degree in 1981 from the University of California, Berkeley. He was on the faculty of MIT as Asst. Professor from 1980–1982 and Harvard University as a chaired Gordon McKay professor in 1994. His areas of personal research are embedded and autonomous software for unmanned systems (especially aerial vehicles), computer vision, computation in novel substrates such as quantum computing, nonlinear and adaptive control, robotic telesurgery, control of hybrid and embedded systems, network embedded systems and software. He has coauthored over 400 technical papers and 9 books, and is a member of the National Academy of Engineering and the American Academy of Arts and Sciences (AAAS). e-mail: sastry@EECS.Berkeley.Edu