Optimal Management and Configuration Methods for Automobile Cruise Control Systems

Arun Adiththan, Kaliappa Ravindran, and S. Ramesh

Abstract Autonomous systems incorporate varying degrees of adaptation behavior to sustain their operations with acceptable quality of service (QoS). The QoS capability of such highly complex dynamic adaptive systems depends on how well they respond to hostile external events. The paper formulates *model-based assessment* techniques to benchmark the QoS capability of a networked system of cars *S*. We elaborate on this approach with a MATLAB-SIMULINK-based case study of adaptive cruise control (ACC) system in automobiles: first, for in-vehicle CC and then for multi-vehicle coordinated ACC. We employ model-predictive intelligent control methods to dynamically adapt the ACC system configurations to attain optimal behavior.

Keywords QoS of adaptive systems · Model-based assessment · Automobile cruise control systems

1 Introduction

A quantitative assessment of the QoS (quality of service) capability of an embedded software system *S* enables a sustained QoS behavior and/or reduced cost of system operations, in the face of uncontrolled external environment conditions incident on *S* (Gjorven and et al. 2006). In this paper, we study the *autonomic management* of adaptive cruise control (ACC) processes in automobiles (Wiki 2016) based on a model-based system assessment logic.

The managed system is a native in-vehicle CC operating under hostile road conditions E^*: e.g., road slipperiness and elevation, wind forces, air density, etc. E^* depicts the events that are hard to measure but nevertheless significantly impact

A. Adiththan (✉) · S. Ramesh
General Motors R&D, Warren, MI, USA

K. Ravindran
The City University of New York, New York, NY, USA

A. M. Madni et al. (eds.), *Recent Trends and Advances in Model Based Systems Engineering*, https://doi.org/10.1007/978-3-030-82083-1_36

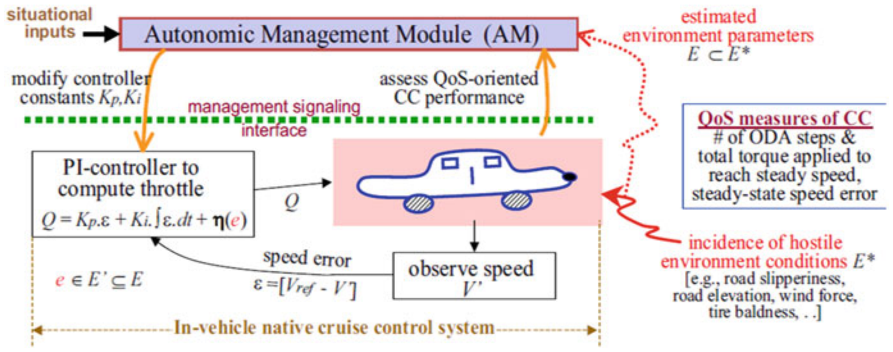
situational inputs
Autonomic Management Module (AM)
estimated environment parameters $E \subset E^*$
modify controller constants K_p, K_i
assess QoS-oriented CC performance
management signaling interface
QoS measures of CC
# of ODA steps & total torque applied to reach steady speed, steady-state speed error
PI-controller to compute throttle
$Q = K_p.\varepsilon + K_i.\int \varepsilon.dt + \eta(e)$
Q
incidence of hostile environment conditions E^*
[e.g., road slipperiness, road elevation, wind force, tire baldness, . .]
speed error
$\varepsilon = [V_{ref} - V']$
observe speed V'
$e \in E' \subseteq E$
In-vehicle native cruise control system

Fig. 1 Functional blocks composed in our ACC system (=CC + AM)

the CC operations. The autonomic manager (AM) is a situation-based control module that optimizes the in-vehicle CC behavior in a context of the sensed external environment conditions $E \subset E^*$. The AM is realized as a hierarchical control module interacting with the in-vehicle CC system via a signaling interface – as advocated in (Ravindran and Ramesh 2014). The paper extends our earlier preliminary study on the infusion of intelligent control behavior by external management modules (Adiththan et al. 2017).

The AM maps the user-level QoS-oriented preferences onto a CC logic. Here, QoS depicts multiple attributes of the speed ramp-up behavior of a vehicle when a reference speed is set, such as fuel consumption for speed ramp-up, latency in reaching a reference speed, and speed jitter in steady-state. The CC logic determines the torque generation from engine components to speed up/down the vehicle in pursuit towards a desired speed while meeting a desired QoS specs. It is often based on a proportional-integral (PI) controller with gain constants: (K_p,K_i) to map the observed speed error onto a torque. We use a CC simulation module developed at CalTech, written in MATLAB-SIMULINK, in our study of AM.

See Fig. 1. The AM consists of two functional modules: (i) *monitor* to assess the QoS performance of CC under the prevailing E^* and (ii) *configuration generator* to determine the modified in-vehicle CC settings (K_p,K_i) to effectively deal with E^*. (i) quantitatively analyzes how the QoS attributes are affected by E^*, whereas (ii) dynamically changes the control algorithm and/or its parameters to achieve a calibrated improvement under E^*. For instance, the torque generated by the in-vehicle CC is increased in the face of higher road elevations and/or wind forces (the latency and steady-state speed are also suitably controlled). The AM dynamically plugs in suitable in-vehicle CC parameters (K_p,K_i), as determined for the sensed environment $E \subset E^*$ under a certain policy. A policy may also factor in other situational inputs: say, road detours and barriers and multi-vehicle coordination needs.

We also consider a multi-vehicle cruise control (MVC), realizable by coordinating the per-vehicle CC modules. The multi-vehicle coordination involves generating vehicle configurations that enforce the inter-vehicle safety spacing constraints while maximizing the collective QoS experienced by the vehicle ensemble.

2 QoS-Oriented View of ACC System

The true model of a vehicle's raw physical process (RPP), i.e., how the engine, tires, and axles perform under various road conditions, is often not known in an exact form. So, the CC logic uses an approximate model of the RPP, as absorbed in a PI-control law (PI: proportional-integral) (CalTech 2014). The difference between the model-expected and observed speeds during a control step is also factored in a decision for the next step about the torque to be applied. An iterative sequence of such control steps leads the CC to a steady state.

The number of observe-decide-act (ODA) steps needed to reach the steady-state (K_{oda}) is a measure of the latency of CC for speed-up, with $\Sigma_{j=1}{}^{Koda}\ Q(j)$ being the total torque expended for the speed ramp-up – and hence the fuel consumed – and $[V_{ref} - V'(K_{oda})]$ being the steady-state error (SSE) (Kienke and Nielson 2005). The latency, torque, and SSE are the QoS attributes for the in-vehicle CC. A parameter setting of small values for the PI constants (K_p,K_i), for instance, yields a lower SSE, *albeit* at the expense of a larger latency and/or more fuel consumption.

The controller generates a trajectory of torque values $[Q_j]j = 1,2,\cdots,K_{oda}$ that finally leads to a sustainable speed V', where $V' \approx V$. An optimal QoS q'_{opt} depicts a scenario where the QoS utility of CC as perceived by the drivers is high, i.e., $U(q, q'_{opt}) \approx 1.0$. A control trajectory, as computed oblivious of the road conditions by a static PI-control law, often leads to a less optimal QoS (when compared with its intelligent counterpart). The suboptimality of QoS is more pronounced when the road conditions are light to moderate, because the static controller is constrained to a smaller region of the search space of [speed, torque] tuples for control trajectory generation.

In our model-based approach, the CC employs user-supplied utility functions to map a degradation in each of these QoS attributes onto user-level displeasure – and hence a contribution to the overall cost. Here, a throttle action X contemplated for speed ramp-up factors in the expected changes in [K_{oda}, Q, SSE] – and hence the cost incurred by X. Among various candidate actions $\{X\}$, the controller chooses an X that is expected to incur the lowest cost and then unfolds X to be exercised on the vehicle. Figure 2 illustrates a mapping of the system-level QoS of CC to an user-perceivable utility index: $U(q, q') \in [0,1]$.

Our QoS-centric optimization approach for CC can be contrasted from the control-theoretic optimization approach advocated in (Bauer and Gauterin 2016). The latter too provides for model-predictive computation and horizon-based planning to deal with: (i) unexpected disturbances encountered on a road over short timescales and (ii) control inaccuracies arising from model uncertainty itself. In our approach, however, (i) and (ii) appear as a difference between the model-computed and observed costs, thereby triggering a throttle action to lower the cost

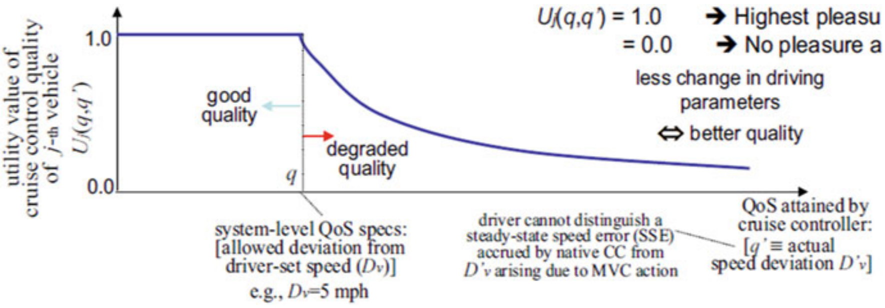

Fig. 2 Usefulness measure of a cruise control system

in a next control step. Because of its reliance on predefined cost relations and utility functions, our approach is more amenable for incorporation as generalizable software-level solutions (say, offloading the expensive model-predictive computations to a cloud).

3 Autonomic Control of In-Vehicle CC

We adjust the K_p and K_i parameters dynamically based on the sensed and/or estimated environment conditions E. For example, if the wind speed increases, K_p is jacked up to generate higher amounts of torque in the throttle actions that would counter the wind forces more effectively (than what a statically set K_p would otherwise do). Likewise, the K_i parameter that captures the accumulated modeling error since the start of a CC action is also jacked up to accelerate the learning. Our dynamic setting of (K_p, K_i) based on the estimating the environment parameters such as road elevation (θ), air density (ρ), and road friction coefficient ($C_r \in [0,1]$) depicts an infusion of controller intelligence with the process level modeling of core components of in-vehicle CC system.

3.1 Optimal Setting of In-Vehicle CC Parameters

Our study on how the QoS changes vis-a-vis some representative (K_p, K_i) parameter values of the PI controller reveals two points. First, a different controller configuration can yield a better QoS under the current ρ-C_r conditions. Second, an optimal setting of the controller parameters can change with ρ-C_r. We thus reason that [K_p, K_i] should be adjustable according to the current ρ-C_r conditions.

We comprehensively analyze the impact of [K_p, K_i] on the overall optimal behavior of the CC sub-system of a vehicle. It involves determining how the optimal

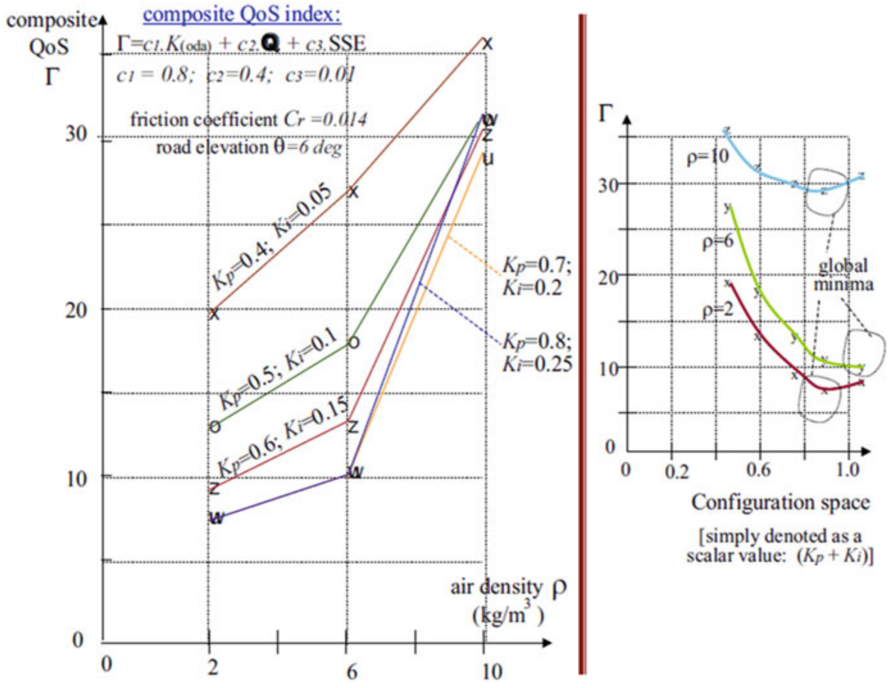

Fig. 3 Simulation results on optimal per-vehicle CC design

controller setting, denoted as [K_{po}, K_{io}], changes under different ρ-C_r conditions. We formulate a composite QoS index γ for the CC sub-system in terms of individual QoS attributes [K_{oda},Q, SSE] attained with a controller setting[K_p, K_i], as:

$$\gamma = c1.K_{oda} + c2.Q + c3.SSE, \tag{1}$$

where *c1*, *c2*, and *c3* are positive constants (we use the notations q and γ interchangeably in the paper). The γ depicts a mapping of the QoS attributes onto a scalar space, attaining an optimal value at: [$K_p = K_{po}$, $K_i = K_{io}$]. The domain-specific interpretation of [K_{oda},Q, SSE] attributes, namely, lower values mean a better QoS, casts the optimization goal as finding the lowest γ value.

We conducted a gradient search of the [K_p, K_i]-space to determine the optimal setting. The composite QoS index γ was computed from the [K_{oda},Q, SSE] data output by the SIMULINK module for various [K_p, K_i] settings under a given ρ-C_r. Figure 3 shows the γ values for different [K_p, K_i] and ρ-C_r values. It is found that the setting [$K_{p0} = 0.7$, $K_{i0} = 0.2$] gives the optimal point, namely, the lowest γ value. The system has a global minimum point, which is detected by the gradient search algorithm in a short number of epochs.

An optimal parameter setting [$K_{p0},\ K_{i0}$] is tied to the external environment conditions: [ρ,C_r]. So, a reasonably accurate (and low cost) estimation of the ρ-C_r values is needed.

3.2 Declarative Specs for Autonomic Control

An infusion of the intelligent control capabilities however requires capturing the domain-knowledge pertinent to CC operations by the AM, as described below.

The AM module may employ policy functions to adjust the (K_p, K_i) parameters – c.f. Eq. 1. We assume that the information about [ρ,C_r, θ, ...] is available to the AM in some form (say, with estimators running in the CC system's control plane). The question then is: how does the AM orchestrate the changes to (K_p, K_i)? This is because triggering these changes requires AM to capture the domain-knowledge: such as an increase in θ or ρ requires generating a higher Q. Over a certain operating region, the domain-knowledge can be codified as axioms and rules maintained by the AM to relate a change in [ρ,C_r, θ, ...] to a desired change in (K_p, K_i).

A declarative specs of the AM-level basic rule relates the current value of ρ to the newly sensed value of ρ as:

$$\begin{aligned} &> [\rho_{sen}, \rho_{cur}] \rightarrow \\ incr\_k_p &\left[diff\left(\rho_{sen}, \rho_{cur}\right), \alpha_{(p,1)}\right] \wedge \\ incr\_k_i &\left[diff\left(\rho_{sen}, \rho_{cur}, \delta_\rho\right), \alpha_{(i,1)}\right]; \\ &incr\_k_p[x, y] \rightarrow \end{aligned}$$

The applicative functions *incr_K*$_p$[.] and assign K_p(.) are exported by the CC system for use by the AM: first, to realize an increase of K_p subject to the condition $\rho_{sen} > (\rho_{cur} + \delta_p)$ and second, to signal a plug-in of the modified K_p to the PI controller. Similarly, the update rules for K_p based on the changes in C_r and θ can be specified. Likewise, the update rules for K_i can be specified.

A declarative specs of the PI-controller update rules can make the AM oblivious of the domain-knowledge, enabling its reuse (at a meta-level). The AM is basically a symbolic processor that evaluates the truth or otherwise of various relations and then plugs in a new (Kp,Ki) as needed.

4 Multi-vehicle Cruise Control (MVC)

We outline the key tenets of a multi-vehicle cruise control system (MVC) for automobiles. Figure 4 shows the functional modules. The QoS depicts how safely a set of vehicles maintain their set speeds within acceptable deviations while avoiding

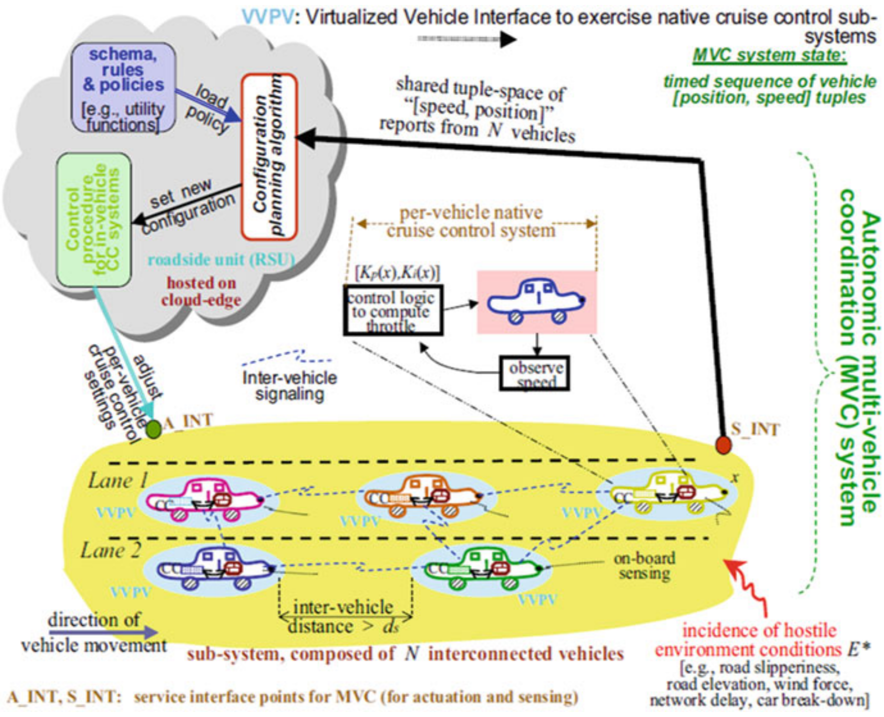
VVPV: Virtualized Vehicle Interface to exercise native cruise control sub-systems
MVC system state:
timed sequence of vehicle [position, speed] tuples
schema, rules & policies [e.g., utility functions]
load policy
Configuration planning algorithm
Control procedure for in-vehicle CC systems
set new configuration
roadside unit (RSU) hosted on cloud-edge
shared tuple-space of "[speed, position]" reports from N vehicles
per-vehicle native cruise control system
[Kp(x),Ki(x)]
control logic to compute throttle
observe speed
adjust per-vehicle cruise control settings
Inter-vehicle signaling
A_INT
S_INT
Autonomic multi-vehicle coordination (MVC) system
Lane 1
Lane 2
VVPV
CC
x
on-board sensing
inter-vehicle distance > ds
direction of vehicle movement
sub-system, composed of N interconnected vehicles
incidence of hostile environment conditions E*
[e.g., road slipperiness, road elevation, wind force, network delay, car break-down]
A_INT, S_INT: service interface points for MVC (for actuation and sensing)

Fig. 4 Functional modules of a MVC system

drastic changes. We provide an empirical view of how an optimal QoS behavior can be determined for a multi-vehicle ensemble.

4.1 Modeling Aspects of MVC

Suppose a vehicle X takes an action to speed up or down, as part of its local CC action. X cannot hit any of the vehicles $\{X'\}$ moving in its vicinity. To satisfy this critical safety requirement, a coordination of the vehicles is needed wherein an intended move of X is evaluated in conjunction with its effects on one or more of the nearby vehicles $\{X'\}$. For instance, a slowdown of X can force a vehicle X' behind also to lower its speed (but may not affect the vehicles ahead). The extent of slowdown of X' is determined by many parameters, such as its distance to X, environment conditions (e.g., road wetness, vehicle traction), occupant characteristics (e.g., comfort level, children and old-aged passengers), etc. We envisage that the multi-vehicle coordination controller C gets hosted on a roadside assist compute node (possibly located at the edge of a compute cloud), with adequate physical and logical sensors and computational resources to enforce the QoS needs of MVC. *C* interacts with the various vehicles under its control realm over wireless signaling channels. Given that each vehicle itself implements an onboard CC module, the coordination of multiple vehicles by *C* is deemed as a hierarchical control.

The QoS in MVC scenario is determined by how far each car is forced to deviate below from the driver expected speed. We define the speed deviation, D_v, as the difference between a driver-set speed and a speed adjustment computed by the MVC controller *C*. One scenario where C may choose to adjust the speed is when a vehicle *V* interacts with other vehicles $\{X'\}$ that have lower locally set speed limits. The D_v is thus different from the in-vehicle *SSE* parameters, but the driver often cannot distinguish between D_v and *SSE*. The adjusted speed computation done at *C* may consider the safe following distance ds set by individual drivers while turning ON the cruise control mode. Different drivers may independently choose d_s values according to the acceptable comfort level while the vehicle is on cruise mode.

The MVC works by potentially overriding the speed limits of individual car drivers, to achieve a collective safe driving of the *N*-vehicle ensemble while striving to keep the D_v of each vehicle low for a given d_s setting (i.e., maximize the per-vehicle QoS of CC). The QoS of MVC is thus gauged by how good the MVC algorithm strives to keep all the *N* vehicles safe with minimal displeasure. An assessment of this MVC capability requires a model of the multi-vehicle ensemble.

4.2 Optimal Configuration of MVC System

A globally optimal configuration for N vehicles is one that minimizes the cost:

$$\Gamma_{\mathrm{m}} = \sum_{k=1}^{N} 1\text{-}U_k \left[D_v(k) - D_v{}'(k) \right]; \qquad (2)$$

subject to a constraint that the inter-vehicle distances are higher than the safe-limit d_s. Here, V(k) is the MVC-enforced speed limit of k^{th} vehicle and $D_0(k) = [V_{ref}(k) - V(k)]$ and $U_k(.)$ is the per-vehicle utility function. Here, $D'_v(k) = [V_{ref}(k) - V(k)]$, where $V'(k)$ is the actual speed sustained by the local CC of k^{th} vehicle, with $\varepsilon(k) = [V(k) - V'(k)]$ being the steady-state error (SSE) induced by the native PI-control law. $D'_v(k)$ is not known to the k^{th} driver, unless the MVC signals about the speed override. Here, the computation of Γ_{m} for a single configuration would amount to running *N* instances of SIMULINK module (each instance with potentially a different parameter set) and summing the per-driver displeasure accrued in that configuration.

Determining the globally minimal Γ_{m} is however an NP-complete problem, as the MVC should conduct an exhaustive search of all feasible configurations. If *L* is the number of distinct speed settings possible, the computational complexity of an exhaustive search would then be $O(L^N)$. Therefore, one should employ a greedy (and/or genetic) search algorithm to quickly come up with a reasonable suboptimal configuration. Such an MVC-level QoS definition would require knowing what the global minimum would be – in order to determine how suboptimal a greedy-computed solution is.

In the absence of accurate knowledge about the best configuration G^*_m, we have the MVC specify a cost threshold to compute an acceptably suboptimal configuration G^a. The use of situational knowledge by a management entity would allow specifying $G^a{}_m$ (say, through a GUI). We then use G^a as a *reference benchmark* for analysis purposes. Now, the MVC would run its own heuristics-based search of some representative configurations to quickly come up with its own view of what the (sub-)optimal configuration would be – which we denote as $G^a{}_g$. If $\Gamma(G^a{}_g) > \Gamma(G^a)$, the search gets preemptively stopped; otherwise, the search would end by a designer-set termination condition: say, the cost difference $[\Gamma(G^a) - \Gamma(G^a{}_g)]$ falls below a threshold. Thereupon, the MVC system is deemed to have reached a steady state.

4.3 Configuration Search Algorithms

As outlined in the earlier section, we employed greedy search and genetic search methods to decide a next state of the MVC system that would generate a close-to-optimal trajectory. To determine how suboptimal the trajectory is, we also carry out an exhaustive search of the state space to find the best solution Γ^*_{m} (which is basically a brute-force computation).

For greedy search, the MVC controller treats a subset of the vehicles $\{X\}$ as ignoring its commands, when computing a new configuration (even though $\{X\}$ may be compliant) – where $1 < |X| \ll N$. The non-consideration of vehicles $\{X\}$ in a decision-making by the MVC manifests as allowing them to stay course, as determined by their native CC sub-systems – while the remaining vehicles $(N - |X|)$ are treated as controllable. This leads to a much small-dimensional search space: $O(L^N - |X|)$. A deterministic search therein yields a candidate that incurs the lowest cost among the small set of configurations considered.

A genetic search, on the other hand, occurs over the full-dimensional space of N vehicles but with a random selection of candidate vehicle configurations. A candidate configuration is produced after multiple crossover checks on the model-computed vehicle states. The random mutation of multiple solutions (two or more) occurs on those known to be good for at least a selected subset of the vehicles. The rationale is that the sub-elements of parent configurations known to be good for some vehicles will likely yield a better offspring when combined (i.e., a new configuration good for additional vehicles as well). The evolutionary process leads to a better offspring, albeit probabilistically, as a result of the crossovers of sub-elements from good parents.

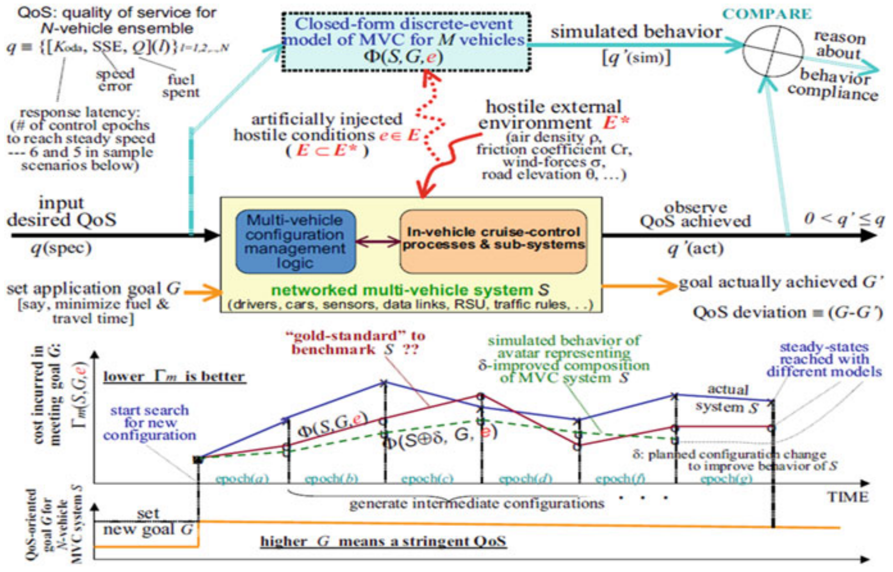
QoS: quality of service for N-vehicle ensemble
q ≡ {[Kods, SSE, Q](l)} l=1,2,..N
speed error
fuel spent
response latency: (# of control epochs to reach steady speed --- 6 and 5 in sample scenarios below)
Closed-form discrete-event model of MVC for M vehicles Φ(S,G,e)
simulated behavior [q'(sim)]
COMPARE
reason about behavior compliance
artificially injected hostile conditions e∈E (E⊂E*)
hostile external environment E* (air density ρ, friction coefficient Cr, wind-forces σ, road elevation θ, ...)
input desired QoS q(spec)
Multi-vehicle configuration management logic
In-vehicle cruise-control processes & sub-systems
observe QoS achieved q'(act)
0 < q' ≤ q
set application goal G [say, minimize fuel & travel time]
networked multi-vehicle system S (drivers, cars, sensors, data links, RSU, traffic rules, . .)
goal actually achieved G'
QoS deviation ≡ (G-G')
"gold-standard" to benchmark S ??
simulated behavior of avatar representing δ-improved composition of MVC system S
steady-states reached with different models
cost incurred in meeting goal G: Γm(S,G,e)
lower Γm is better
start search for new configuration
Φ(S,G,e)
Φ(S⊕δ, G, e)
actual system S
δ: planned configuration change to improve behavior of S
epoch(a) epoch(b) epoch(c) epoch(d) epoch(f) epoch(g)
generate intermediate configurations . . .
TIME
QoS-oriented goal G for N-vehicle MVC system S
set new goal G
higher G means a stringent QoS

Fig. 5 Avatar-based approach for MVC system benchmarking

4.4 *Avatar-Based Benchmarking of MVC System*

We employ *avatars* to generate multiple benchmarks for an MVC system S in a simulated world. An avatar is basically a mathematical model of S executed in a virtualized world, which is subjected to simulated environment conditions that are close approximations to the actual conditions incident on S. A model $\Phi(S,E,G)$ representing the current system S is used to benchmark the actual behavior of S. This objectively verifies the goodness of S in meeting its QoS goal G in the face of environment conditions $e \in E$. See Fig. 5.

Different models of S that represent alternate compositions of S from the core functional components, namely, the per-vehicle CC processes and the configuration parameters, may yield different QoS behaviors. For instance, an increase in speed and/or lane change of one or more cars can be parameterized in the sub-system models, thereby computing its effects on the MVC-wide QoS – and the associated configuration cost (S,G,E). Such purported changes are first analyzed by a model-driven simulation of the modified MVC processes $(S + \delta, e,G)$ in a virtualized world, before committing a configuration change δ in the actual MVC system. The analysis involves two things: (1) comparing the estimated behavior of model $\Phi(S + \delta,E,G)$ with the currently observed run-time behavior of actual system S and (2) infusion of system-level changes δ, if needed, in the operational processes of S using compositional methods (e.g., changing the K_p/K_i parameters of in-vehicle CC). In the above light, the behavior computed by the model (S, e,G) serves as a "gold-standard": first, to determine how good is the actual behavior of S relative to the expectations and second, to provide the basis for a calibrated change in S. In general, the different benchmarks obtained from alternate compositions of S can be analyzed to meet the desired optimality conditions. The model-driven simulations of $\Phi(S,E,G)$ and $\Phi(S + \delta,E,G)$ can be seen as different avatars of S for benchmarking and reconfigurations of S.

5 Conclusions

Our work adds a *software cybernetics* dimension to the autonomic management of an in-vehicle CC system. The dynamically settable CC logic allows a vehicle to be more responsive to the stringent demands in a multi-vehicle coordination scenario, relative to a statically set CC. Our compositional design of ACC system promotes incremental evolution and reconfiguration of ACC functionality by software and/or model reuse. The paper also described a multi-vehicle coordination functionality composed from the per-vehicle CC models (static or dynamic) – realizable on a roadside unit (say, hosted on a cloud edge). Our adaptive control of vehicles meets a critical functionality requirement for Intelligent Transport Systems, as enunciated in (Koopman and Wagner 2014).

References

Adiththan, A., K. Ravindran, and S. Ramesh. July 2017. Management of QoS-oriented Adaptation in Automobile Cruise Control Systems. In *Proc. Intl. Conf. on Autonomic Computing, (ICAC)*, 79–80. Columbus.

Bauer, K. and F. Gauterin. 2016. *A Two-Layer Approach for Predictive Optimal Cruise Control*. SAE Technical Paper: 2016-01-0634. https://doi.org/10.4271/2016-01-0634.

Caltech Research Group. Cruise Control. 2014. http://www.cds.caltech.edu/murray/amwiki/index.php/Cruise control.

Gjorven, E., F. Eliassen, and J.O. Aagedel. 2006. Quality of Adaptation. In *Proc. SELF: Self Adaptability and Self-Management of Context-Aware Systems, Silicon Valley*.

Kienke, U., and L. Nielsen. 2005. Road and Driver Models. Chap. 11. In *Automotive Control Systems: Engine, Driveline, and Vehicle*. Springer.

Koopman, P., and M. Wagner. Jan 2014. Transportation CPS Safety Challenges. In *NSF Workshop on Transportation Cyber Physical System*. Arlington.

Ravindran, K., and S. Ramesh. 2014. *Model-Based Design of Cyber-Physical Software Systems for Smart Worlds: A Software Engineering Perspective. In workshop on Modern Software Eng*. Hyderabad: Methods for Industrial Automation (MoSEMInA), ACM-ICSE.

Wikipedia. 2016. *Autonomous cruise control system*. https://en.wikipedia.org/wiki/Autonomous cruise control system.