Tailoring automotive E/E architectures with design space exploration

Laurent Jean-Baptiste
Forvia
Paris, France
paper.b1n7t@passinbox.com

Richefeu Julien
Forvia
Paris, France
julien.richefeu@forvia.com

Monsuez Bruno
ENSTA
Paris, France
bruno.monsuez@ensta-paris.fr

***Abstract*— Automotive E/E architectures truly became a creative ground for designers to innovate. Yet, we face a glass-ceiling trying to exploit such potential with existing tools, which led produced heterogeneous architectures to be altered over and over again. However, the subject remains neither simple nor in the hand of a single actor: carmakers, OEMs and standard consortiums do share responsibilities for those ambitious architectures. We developed a more meta oriented methodology, illustrated with us integrating an abstracted driving monitoring system (DMS) onto an external system. As a result, we found it possible to progressively move toward complex solutions while navigating the solution space. Moreover, this guided design space exploration (DSE) also allows us to backtrack to former decisions later on. For example, backtracking would occur if we receive new priorities from decision makers, which would involve a re-evaluation of our constraints. Compared to existing solutions, we encompass the entire design process as to view the target system as a single project, rather than an aggregation of components. This, at last, helps with global optimization and the avoidance of side effects otherwise brought at the final integration phase. Though promising, we are still in the process of seeing if our method will remain simple enough against the combinatory explosion of an exhaustive architectural description.**

***Index terms*—Toolchain, methodology, automotive E/E architecture, software defined vehicle, design, design space exploration, model based design**

I. Introduction

Automotive electrical and electronic (E/E) architectures reach an unmanageable level of complexity. Their current design is unable to support the growing amount of components which went from a couple, to more than a hundred in the past years [1]. It is further exacerbated by both: the former choice for a distributed topology, and the sourcing of components made by different original equipment manufacturers (OEMs). Allowing those black boxed components to cooperate, and communicate properly in a distributed manner, proves to be challenging [2].

Chronologically, automotive E/E architectures only had a couple of components attached to a backbone, justifying the efficient distributed design choice. It formed a cost-efficient "plug and play" kind of architecture [3]. A controller area network (CAN) bus topology took care of the communication without the need for a centralized host computer. Naturally, it also translates onto car makers being responsible for the architecture description, while delegating subcomponents to tier-X suppliers. The supply tier structure is a pyramid, with tier-1s handling the provision of features (or modules), tier-2s providing lower components, and tier-3s the materials and end parts.

For convenience, tiers actors supply complete, but often opaque solutions (with both the hardware and software), alleviating the need to know and deal with more than the necessary interface details.

Over time, numerous more technologies made their way in, bringing their share of new innovative and connected features. It's now rather uncommon to cross a modern car lacking a dashboard screen or a rear-view camera for parking assist. Moreover, those features are more capability hungry than their older counterparts:

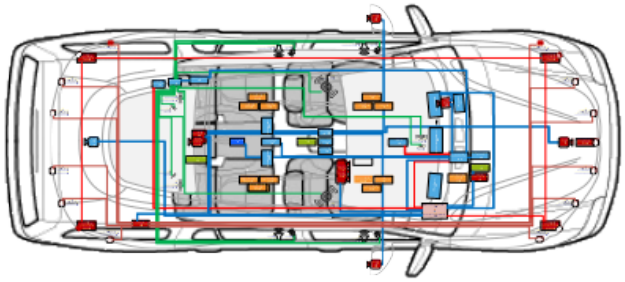


Figure 1: An automotive E/E architecture showing scattered and independent subsystems, with the audio in green, the radars in red, and the infotainment in blue.

For example, a parking assist used to only contain a beeping proximity sensor. Attaching a camera to this system would be a net enhancement, giving the driver a clear view and idea of what is going on behind the car. However, it also adds a strain on the backbone communication channel. The display will also have to be considered.

As a result of this digitization, cars can't maintain their former architectural design, as it's by nature unfit to support this many highly cooperating components.

The target of our work is to end up with flexible, and comprehensible enough architectures at the design level. In that regard, we worked on an iterative design methodology to guide the decision-making process. We also use design space exploration (DSE) in conjunction with a component-based model to leverage the approach. This contrast with the more fixed, concrete way where flexibility is granted by the architecture itself.

In the next section, we highlight some important points impacting choices during the design. We then illustrate in Section 3 our methodology with a bottom-up approach. A fictive use-case in Section 4 covers the design of a driver monitoring systems (DMS). Section 5 discusses the results and limitations of our approach, while opening the subject for further applications. Future works will be brought in Section 6, which will conclude with our remarks.

II. Architectural considerations

A. Keeping cars updated

The lifespan of cars is steadily increasing [4], [5]. This means that embedded technologies will by default suffer deprecation for a longer usage period past the manufacturing date. To mitigate this, car makers had to follow upgrades strategies [6], such as carefully planning the frequency and timing of added value ECUs brought in.

Although, software-defined vehicles (SDV) compatible architectures recently allowed car makers to extend the lifespan of their fleets. Over the air (OTA) updates, application-level features, edge and cloud computing, followed by a strong app market [7] will then make up new longer-term strategies.

We should note that the mobile industry successfully went through this design shift [7, ch2]. While this looks encouraging, mobiles underwent massive architectural changes to move from old portable phones to the now very potent smartphones. Designing automotive E/E architectures remains very complex by nature, great care will have to be placed during the transition.

B. Mitigation strategies toward the complexity explosion

To address this, centralization iterations were made to structure the architecture onto hierarchical, and fewer entangled layers as seen in Figure 1. Instead of each feature being plugged into the main backbone, they are layered behind domain control units (DCUs). While it did help mitigate the distributed topology complexity explosion, two issues emerged: 1) those domains were still segregated in regard to the others, and 2) the lack of consideration for topographical details. The sound and radar subsystem, shown in green and red in Figure 1, had their components logically regrouped, yet, scattered across the entire car. Both used their wiring, which could here technically be mutualized. Using a common architectural backbone would also help to eliminate its wiring being the third-heaviest piece of equipment [8].

In essence, DCUs regroup logically similar features, but fail to address topographical considerations. Zonal gateways were brought in later down the road to instead structure the car into physical "zones". Zonals are under development [3], [9], [7] for most car makers. It's yet to be defined how much work and responsibilities the zonals will carry, or how to deal with security issues brought by mixing heterogeneous components.

It should be mentioned that both zonals and domains have their pros and cons [9]. Just like how distributed networks do work in their own conditions: a system overcommitting on using a single design pattern is likely to be short-sighted. Knowing how rich or legacy the design can be, making the best of both could be a necessity.

III. Methodology and toolchain details

A. The architectural model views

Our methodology in Figure 3 intends to cover the entire design cycle, and end with a meta and runnable prototype. Within a V-cycle project, this represents the descending steps excluding the implementation at the bottom. Our methodology is composed of a global iterative method paired with a toolchain. This toolchain is responsible for the evaluation and representation of the architecture.

Within that toolchain, we model the architecture using three executable and orthogonal views: 1) the functional view, or the features running on the system, 2) the hardware, or system layout, for the capabilities offered by the architecture, and 3) a mapping of the former to the latter.

The mapping is for the design space exploration. It aims at easily performing the combination work to gather metrics, test alternatives, and converge toward an optimum. This last view closely follows model based design (MBD) principles [10], [11] and heuristics [12, p 395]. The above views make up for the first out of 3 extendable components composing the toolchain in Figure 2.

Working with 3 independent views, and converging halfway, allows us to focus on each aspect without tainting the others. If we were excluding the system layout, we would for example start with a definition of the features we want, to think then of an architecture made for those. However, the architecture would end up being a consequence of the original design, instead of a platform made to provide resources. It could also overshadow alternatives since it is designed to respond to a fairly known set of functions.

B. *The evaluation toolchain*

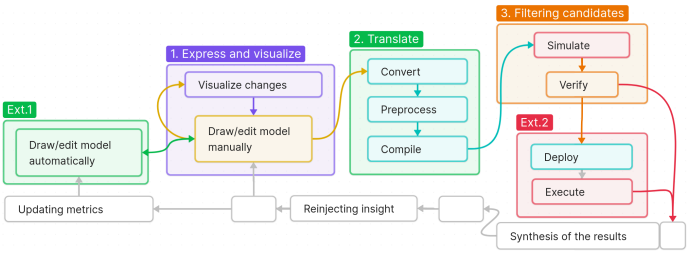


Figure 2: The design space exploration toolchain, used at the 5th step of the methodology

Beyond the raw architectural description, the first component also aims at providing a common discussion ground. We used a canvas to both visually represent and centralize information, in order to avoid redundancy and stay consistent along the workflow. To keep complexity in check and remain simple as much as possible, we delegate the burden to be exhaustive to the specifications.

Since they serve as guidelines to express our models, it is perfectly fine for those to be incomplete this early within design process. We intend them to be refined as we progress with how we understanding the project being worked on. Being draft material leaves us with the possibility to cherry-pick only the expressible details with a direct impact on the result. For example, topological details or how processes interact with one another.

The second component, or translator, is purely technical and works behind the hood. It reads the model canvas and translates it onto different formats for external tools to process. Furthermore, it also allows the canvas to be later used as a graphical front-end to control the DSE (changing topographical settings, scale components by editing values, etc.).

The third component is the simulation engine itself. Once attached to the translator, it seamlessly produces key performance indicators (KPIs) related to the architecture we want to evaluate. Time being the most important reference for our simulation, we experimented first with KPIs such as latencies, waiting time when a resource is unavailable (annotated by a scheduler upon release), and various ratios and statistics. It's with those that we are then allowed to deduce the efficiency of more complex choices such as the communication layout, mapping choices (such as the placement of tasks, CPUs or redundancy of links). However, even though it is supported by the framework, no dynamic changes of behavior has been done to test how the system would behave. This would apply if a link has a sudden change in performance, such as a crash for safety analysis. Recorded results are then fed back in report files and displayed straight back to the canvas.

With these three components, the work now revolves around choosing, configuring and linking the architecture pieces within the three model views. The canvas front-end allows us to do this easily, compared to toying with the cryptic and unavoidable code-level details for if we had kept the simulation engine alone.

We did a proof of concept implementation using an Obsidian [13] canvas, a python script and the Metropolis simulation engine [14]. Obsidian is a note-taking app storing canvas in a convenient JSON format to parse.

Two extensions can be developed to enhance the exploration capabilities: a pre-canvas content generator to morph the architecture, and the ability to deploy on a real hardware setup.

The first one allows for the automatization of the process using metrics provided by either the simulation engine or a live demo. At this point, the design space exploration can technically be turned into an optimization problem.

The second extension gives us real KPIs by measuring it live. This can then be used to correct the arbitrary cost variables given to the simulation engine. Reality grounded metrics, such as latency, would otherwise only make sense relatively from one architectural candidate to another, since we use metamodels and abstract views. Full automation with the complete chain is possible for tuning use-cases, but would represents a considerable amount of work. Still, a manual intervention is likely unavoidable for any non-value changes, such as the topology in the demo setup or a change in equipment.

C. *The encompassing, iterative method*

The toolchain allows us to evaluate an architecture drew on a canvas. This gives us the luxury to focus on the architecture, design choices, and composition without worrying too much about tooling details. However, as it is, it lacks focus points and clear direction guidelines.

Car makers, OEMs and consortium all have different expectations for what makes a good architecture. OEMs value design flexibility over cost so they could sell already developed products to different car makers, and avoid redoing lengthy work by focusing on slight variations. Car

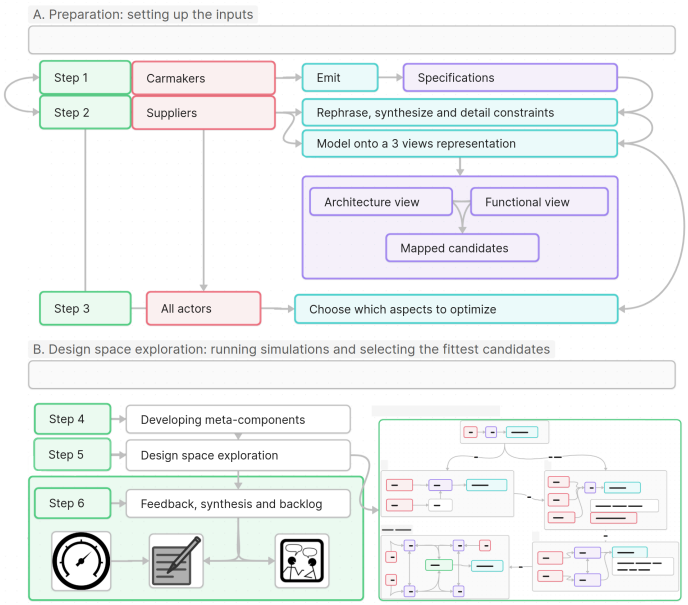


Figure 3: The encompassing design methodology

makers value the cost and compatibility with the existing, which often leads to a custom and tailored solution. Consortiums value the design extensibility and potential ways to monetize by license to benefit from the "contagion": using a component developed with a patented API will force car makers or OEMs to purchase said license. By developing an encompassing method on the toolchain, we allow actors to set up a clear context and then explore compatible alternative designs.

To build up those candidates for evaluation, we encompass the toolchain with the following six methodological steps:

1) *Specifications:* We first gather and rephrase the expectations into functions, capabilities and constraints. This makes up for the bulk of the specifications. It is to be noted that functional business specifications are likely expressed in a way that cannot be directly translated. Infotainment, for example, is a high level system-wide feature involving many things needing to be broke down first.

2) *Models:* Once there, we translate those into tangible, technical, and abstracted views. For a global system optimization pass to be performed, the design granularity has to be gradually increased by starting to decompose each feature as a set of at least software and hardware components. Constraints, API details and middleware requirements will come during refinement stages. We start here to define the views we will need later.

3) *Goals:* We choose the variables, constraints, and optimizations directions we want to drive the exploration toward. Our design work revolves around this very decision-making process. Each iteration of the methodology starts and stops at a point where candidates get ready to be refined. The choices are rendered visible by the exploration, along with the decisions made by the stakeholders based on the extracted KPIs. Such decisions could be to converge toward a cheaper design, or allow the designer to test more disruptive solutions for greater insight.

This third step, deciding which direction to explore, is also when the content of the third toolchain component is defined. We require at least a (meta)-model simulation engine to be present for runtime feedback. Coupling the engine with more analysis capabilities would enhance the relevance of the pass further if needs be, at the cost of an increase in its complexity and configuration time.

For example, if the only goal is to test for if a proposition can reach a given safety level, then using a formal verification framework such as SPIN [15] would be in order. If we are after a certification or a validation of the design, then a less constrained static model analysis could be performed to extract data such as the fairness in CPU usage, expected latencies or bandwidth estimated usages.

4) *Meta-components:* We implement the metamodel components as libraries for the simulation engine. This work only needs to be done once under the form of configurable packages. Moreover, most automotive design can be summarized onto a selective set of common, grounded features with often only a few of disruptive features. If the foundation features are already developed, not much remains to be done at the first few abstraction levels.

5) *Evaluation:* At this point we have a drew, backed-up first model we can use the toolchain on. The toolchain produces a set of results with the provided views of the system. This is the exploration part of the methodology. If properly done, non-technical members of the project could handle this part. To some extent, it is possible for them to perform minor edits on the canvas and communicate about the impact their decisions would have (drag and dropping boxes onto the canvas, and clicking on "run"). We can be creative here and alter the global topology, the strength and kind of components or their amount.

6) *Retrospective:* This is the synthesis, and feedback step. Actors analyze the results and draw conclusions from the fittest architectural candidates they have out of this iteration. A decision will then be made to keep or switch the refinement direction (like if a certain component has finally reached the expected results), re-weight constraints, or call for the start of a demo-ready implementation.

D. Notes on the methodology

Since automotive E/E architecture is an ill-structured problem, it is likely we'd have to rework and complete the original set of specifications.

Hence why keeping a record of our exploration with each decision made also accounts for an important part of the methodology. The exploration tree will be refined as we get a better understanding of the project. As long as the decision to start the development is kept on hold, or until an agreement is made, we repeat the cooperative process between the actors and repeat the process back to step 1.

Depending on the constraints and optimization direction, it might not be efficient to consider the architecture as a multi objective optimization problem. In that case, we need to first rank them and either define the order in which they will be dealt with, or use dedicated algorithms to select the most fitting one [16]. Once done, it becomes possible to work toward a particular direction until the objective is reached (such as the validation that certain areas of the model are compliant with cybersecurity requirements).

Each iteration aims to mutate the architecture to support a new constraint. To avoid the explosion of complexity, those have to be abstracted first and refined with the rest of the model over time.

Once done, the process can be reiterated until a simulated solution fits the original, refined requirements or if the exploration managed to reach a solution ready to be deployed. At this point, it is possible to start the development process and the certification process in parallel based on the already tested model used during the designing work.

While we are leaning toward supporting complete ecosystems, the methodology can also be used partially to support small models. It then allows the designer to incorporate more of the steps as the architecture gets more complex.

We see gradual scaling as essential when it comes to assembling a heterogeneous system, especially if we aim at keeping things simple from the start. Moreover, this allows the designer to dig through the abstraction layers one step at a time and evaluate different views to choose from early in the design. To illustrate, we can start with how to plug an independent DMS onto a pre-existing ecosystem. If we need to merge the two more deeply, we can see if it is possible to wire the cameras directly onto the ecosystem processing resources. If the DMS is to be upgraded further to support an occupant monitoring systems (OMS), then the same set of originally used components can be reworked.

IV. Establishing a draft, DMS design use-case

We are going to display a small example on how constraints and expectations can naturally mutate. It will help us highlight the relevance to keep a decision tree and a methodology to iterate with if a former decision has to be remade.

A. Step 1: Feature specification and constraints

New cars in Europe will legally have to embed a DMS [17, art. 6] by July 2024. This is a new regulatory constrained feature which forced OEMs to adjust their designs. Among other things, a DMS detects when the driver is losing focus. A safety constraint is also present due to the infrared light pointing at the driving to work at night. Blinding the driver is, by all means, not recommended. This detection is based on a camera pointed at the driver, looking for signs of gaze and distraction. In that case, a bare-bones DMS implementation emits a signal fetched by the rest of the architecture, but remains otherwise passive. If we are creative, we could design it to be proactive and communicate with a driver-assistance subsystem (ADAS), which could slow the car down or force a stop.

B. First iteration: initial candidates and insight

The DMS can be modeled as a camera and a light, paired with a CPU running an algorithm fed with the video. With its single function, we can use a straightforward 1 to 1 mapping and assume the algorithm is by default running on the CPU. It would be a completely independent and separated system whose outputs would be then transmitted for action by the main system.

Designing a DMS solution as an independent sub-subsystem gives us a couple of notable advantages. Being ISO 26262 [18] certified is going to be straightforward: This small system can be verified only once, is self-contained, and won't need to be verified again due to third-party changes. Entangled systems tend to suffer from this, as it makes noticeably harder to validate large chunks. Choosing your own, custom components is also very comfortable and avoids the burden to fit somewhere else. If the camera has to be tuned to a certain frequency or color setup, it is also possible without impeding something else. Finally, we can measure end to end performance and answer car maker's request for quotation (RFQ) with solid, reproducible metrics.

Unfortunately, that DMS in itself doesn't provide any added value: A design like the above will only increase the cost of the car. From an OEM perspective, the DMS feature is a commodity and a monetary impact to be avoided. With that in mind, we can add a new economic constraint by putting a cost marker on the solution's composition.

This solution presents itself as a dead-end. Starting small allowed us to foresee such kind of issues before any (meta)-development work was required. However, we do not need to discard our solution and start yet again from scratch: any regular DMS implementation will have the components we already defined. Pivoting from here using our model views is going to be straightforward.

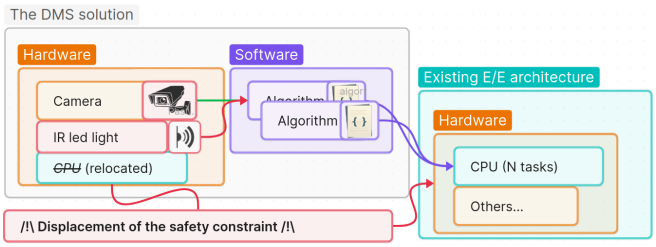


Figure 4: A DMS using existing processing resources, with the potential side effect of propagating a safety constraint related to the IR light onto the host system.

Since a static analysis was enough without exploring alternatives, we could jump to the reflection and skip steps 4 to 5 (i.e.: developing new meta components, toying with the exploration toolchain, and synthesizing results).

Step 6 will be about defining the acceptable costs and set the direction toward its reduction. We are now ready to repeat the process and look for new alternatives, but this time with a model enriched with a justified constraint.

C. *Second iteration: centralization ?*

To reduce the cost of our standalone solution, we can check for things to remove. Out of the 3 views, we cannot suppress the algorithm, which leaves us with the hardware layout. We can choose to borrow processing resources directly from the host architectures.

This solution has several pros and cons: firstly, this prevents duplicated resources, a clear cost cut. However, the software development will likely cost more for it to run on multiple target architectures. Secondly, car makers can optimize their design with better hardware. Those units could also be used by other unrelated features and evaluated for performance. A dedicated refinement stage would address that later down the road, once the bigger decisions branches are clear. Lastly, it is fewer responsibilities for OEMs, which only have to provide their subsystem needs. Yet, car makers will now be burdened with the safety brought by the software handling the LED light.

We are starting to scale the methodology as the model is growing more complex. Compared to the first iteration, we now have room to test different alternatives and drew them on canvas.

Some back and forth will be needed between the actors to correctly weigh the options. For example, is the CPU expected to be a safety equipment already ? How many platforms shall be supported ? Do we plan on building a synergy between the DMS and other equipment, such as Zonals in case we want to add telematics or diagnostics ?

The runtime part of the chain can be left for when we will be done with the static content (overall cost and safety).

D. *Third iteration: refinement*

In the present use-case, we assume the above CPU centralized solution has been accepted by the actors and is ready. We explored a couple of alternatives and decided to now detail the external CPU into our model and include third-party software. The car maker also decided to disclose its original intention to facilitate the design This design would use the DMS video stream, and display it on the front screen side by side with another video (i.e., to allow video calls to happen while at the same time displaying the rear-view camera (RVC) during a slow parking maneuver).

For the exercise, we have also detailed our function view by adding two more components: 1) A converter unit, which convert the analogical video stream into a processable format. Later on, this function could also correct fisheye effect or perform demodulation if the camera requires it. 2) A merging unit, to fuse both videos and transmit them to the front screen. For now, we assume it will be processed by a CPU, but the function could be refined to have a much better cost to performance ratio if used on a GPU.

We arranged 3 mapped candidates C1, C2 and C3 in Figure 5. C1, the original distributed design, will serve as a reference point as it is our first and most understood solution. Each component runs on its own hardware and as a result, both cameras stream to their respective converter, which in turn forward the clean stream to a merging computing unit before rendering. Unlike C1, C2 is a fully centralized where even the cameras are included within a single monolithic solution. Even though their functional flow is identical, only a single network link is now needed: Between the main unit and the screen. Finally, C3 is a middle ground solution. It limits itself merging converters and the merging unit.

Running the toolchain on those models brings results regarding their performance in this setup. Yet, the important portion doesn't come from this evaluation alone, but from the ability to test any variations solely by drawing a new model. Even though C1, C2 and C3 are fictive, we covered a centralized and distributed approach. The results will be used as a discussion ground to drive the next solution.

Due to automotive constraints, such as real-time or safety, centralizing systems components elsewhere is not always worth the effort. Centralizing forces components to be brought to the grade level of the most restricted component running on it: real time implies a response within a defined time-lapse (airbag), and the safety enforces a certain level of reliability. High levels of safety require redundancy by design to make sure the subsystem will remain functional in the event of a localized failure.

To summarize, automotive E/E architectures are rich but heterogeneous systems with mixed criticality compo-

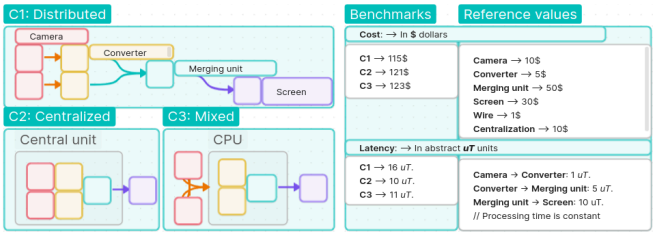


Figure 5: A benchmark with arbitrary weights on different emulated candidates by the toolchain

nents. Hence, why producing an entire design without clearly defined methodologies or outdated tools is such a difficult manual task.

V. Discussion

There is a significant difficulty gap between the design of a standalone feature and the optimization of an entire ecosystem, as seen with our previous DMS use-case. A single designer responsible to prototype such a solution would have to spend considerable time and effort doing the evaluations and combination work.

Without the means to explore, evaluate, and structure the process around all the skilled people involved, tasking someone with this will likely only result in a disconnected design that will later suffer from erosion.

Due to this, safety and cybersecurity, for example, are often delayed even though it is capable of shattering an originally well-made design: if we had ignored the safety propagation constraints in our DMS use-case and carried on with the design until development, we could have ended backtracking nearly back to square one.

Moreover, we also refined the semantic of our goal from developing a DMS, to adding DMS capabilities to an existing automotive E/E architecture, to finally model outside components to fit best. We see this as a shift in how the original design complexity issue shall be seen and later addressed.

As a result, a toolchain and an encompassing decision-making methodology were applied, making that very design a more straightforward and accessible task. As is the case with new projects, we have reserves for if our methodology will be capable of truthfully carry a full design.

More development will be needed to reach the potential we see in it. We think that if we had a clear how-to guide combined with the right tools to guide and track the DSE, we wouldn't have had such complexity issues in the first place. The time to market delay would be shortened, and car features adapted over time, preventing timed obsolescence and depreciation.

We intended our methodology to act more as a guide rather than a fully detailed how-to, for a couple of reasons: First, it is the role of the toolchain to evaluate and explore technical architecture compositions and type of network topology. Second, there is more to a design than the sum of its technical components. Decision-making is the key factor into achieving a pertinent design, and to enforce coherency, the work of an architect will always be needed. Consideration of political natures, legacy components, and experiences will come into play later.

Our methodology also aims at structuring the interaction between the actors, gathering and phrasing all the specialized constraints onto a comprehensible whole. Once done, progress can be done and evaluated iteratively. To some extent, it extracts the most creative and decision rich part of the work and leaves behind the technicality of it behind the simulation and modeling engines. At this point, we are able to see the system as an optimization puzzle composed of pre-developed pieces to arrange.

The work presented here is adaptable to other areas if needs be. On one hand, it could be applied on telecom distributed networks as they are also composed of a heterogeneous mix of vastly different components [19]. However, they do have their share of restrictions: retro compatibility of the connection points (4G, 5G, ...) and a much larger number of actors. On the other hand, modeling a distributed program for cluster computing would be vastly easier, as it is a single actor environment with likely only performance constraints.

As long as it is possible to model the constraints, our methodology and toolchain pair would help explore the space of solutions to optimize and draw a concept of an ecosystem. Issues would have to be addressed, but for the first few abstract iterations it would work out of the box. It is once the last refinement stages will be reached that more dedicated field implementation expertise will be required to prune exploration branches.

A. Future work

Realistically, two things can be completed within the medium term: 1) A better version of the toolchain to deal with more complex architectures and 2), testing the methodology on real use-cases in complement with the original approach to enrich the proposal feedback, and challenge our work.

If the work ends up gaining traction, various more areas would benefit from further research. The first two will relate to design modeling theory, as to express as simply as possible a rich and untangled ecosystem. We use the MBD, but other designing ways could prove to be more convenient when it comes to converging toward a solution, layer by layer, and one decision at a time. Other research would then need to be conducted to effectively translate

the methodology from the prototyping stage, to an industrial environment.

We raise questions such as if it will be possible to fit in every constraint within a single, visual representation, or the number of tooling layers that may be required to attain the desired outcome is also yet to be answered.

A few more questions to open future work on the subject: how well could the model be translated toward an implementation, how do you manage people around it, how to use continuous integration or agile development alongside it ? Or how to scale the methodology to be efficiently used by teams to optimize a design composed of products from each of them. That is yet to be more deeply defined and tested.

VI. Conclusion

What was originally a front-end to a simulation engine ended up into a more potent methodology. Our methodology is now a structure to help early in the design process of automotive E/E architectures. Applied to a fictive DMS use-case, it managed to regroup architecture specifications and unrelated constraints, while allowing us to: 1) gradually use the methodology content as the model grows in complexity, 2) forward the resulting candidates to a simulation engine, 3) perform (semi)-automated DSE and 4) allow non-technical decision makers to drive said exploration early in the design process to enhance communication between parties.

Tools do exist to cover the different aspects of the methodology, but we found none to perform a guided DSE from the original functional specification, nor consider the set of different constraints together.

While our work is at the prototyping stage, we believe it to be a step in the right direction toward producing a reliable architecture faster, and ultimately end up capable enough for them to be adapted. This is in contrast to generic, often suboptimal architectures attempting to fit a large spectrum of use-cases in order to mitigate redundant development cost. As for an industrial use-case, a supplier would be able to evaluate different alternatives very early in the design, and provide insight regarding possible improvements. Optimizations could be done "as-is," or with off-the-shelf components to respond well to RFQs.

At last, we allowed our methodology to be translated to the design of other engineering systems. Some adaptation would be needed, such as identifying and modeling their constraints, but that would be about it before the instrumentation could carry on. As our time is limited, usage outside the automotive industry shall be left for future research. In the short term, we plan on conducting studies on the development and design of the metamodel itself to be in better associated with the simulation. Finally, this would allow the methodology to be challenged on more sophisticated architectures, and ultimately be used in production.

References

[1] P. Pelliccione *et al.*, "Automotive Architecture Framework: The experience of Volvo Cars," Jun. 2017.

[2] J. Happian-Smith, Ed., *An introduction to modern vehicle design*, Nachdr. Oxford: Butterworth-Heinemann, 2006.

[3] A. H, H. Morteza, and K. Alois, "E/E Architecture Synthesis: Challenges and Technologies." [Online]. Available: https://doi.org/10.3390/electronics11040518

[4] B. W. Hamilton and M. K. Macauley, "Heredity or Environment: Why Is Automobile Longevity Increasing?," 1999, [Online]. Available: http://www.jstor.org/stable/117513

[5] V. Nguyen-Tien, R. J.R. Elliott, E. Strobl, and C. Zhang, "Estimating the longevity of electric vehicles: What do 300 million MOT test results tell us?," Jan. 2024.

[6] T. Şahin, L. Köster, T. Huth, and T. Vietor, "How to Upgrade Vehicles? Release Planning in the Automotive Industry," May 2021. doi: 10.1007/978-3-658-33521-2_12.

[7] T. Irawan, A. Nonnenmacher, and S. Dirk, *The Software-Defined Vehicle A Digital-First Approach to Creating Next-Generation Experiences*. O'Reilly, 2023.

[8] Ixia, "Automotive Ethernet: An Overview." [Online]. Available: https://support.ixiacom.com/sites/default/files/resources/whitepaper/ixia-automotive-ethernet-primer-whitepaper_1.pdf

[9] V. Bandur, G. Selim, V. Pantelic, and M. Lawford, "Making the Case for Centralized Automotive E/E Architectures," vol. 0, no. 2, p. , Feb. 2021, doi: 10.1109/TVT.2021.3054934.

[10] J. Jeff C, "Elements of Model-Based Design." Feb. 2010.

[11] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli, "System-level design: orthogonalization of concerns and platform-based design," pp. 1523–1543, Dec. 2000, doi: 10.1109/43.898830.

[12] M. Maier, M. W. Maier, and E. Rechtin, *The art of systems architecting*, 3rd. ed. Boca Raton, Fla.: CRC Press, 2009.

[13] X. E. Li Shida, "A personal note-taking software." [Online]. Available: https://obsidian.md/

[14] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli, "Metropolis: an integrated electronic system design environment," 2003, doi: 10.1109/MC.2003.1193228.

[15] G. J. Holzmann, "Spin - Formal Verification." [Online]. Available: http://spinroot.com/spin/whatispin.html

[16] P. Payal and A. Naveen, "Approach to Evaluation of Alternative System Architectures," in *AoSec 2023*, Dec. 2023.

[17] "Regulation (EU) 2019/2144." Accessed: Aug. 01, 2023. [Online]. Available: http://data.europa.eu/eli/reg/2019/2144/oj/eng

[18] 14:00-17:00, "ISO 26262-1:2011." Accessed: Sep. 15, 2023. [Online]. Available: https://www.iso.org/standard/43464.html

[19] "5G-PPP-5G-Architecture-White-Paper." [Online]. Available: https://5g-ppp.eu/wp-content/uploads/2017/07/5G-PPP-5G-Architecture-White-Paper-2-Summer-2017_For-Public-Consultation.pdf