Paper Number: 2007-01-1273

An Initial Study on Monetary Cost Evaluation for the Design of Automotive Electrical Architectures

Arkadeb Ghosal, Alberto Sangiovanni-Vincentelli
University of California, Berkeley

Sri Kanajan, Randall Urbance
General Motors Research

ABSTRACT

One of the many challenges facing electronic[1] system architects is how to provide a cost estimate related to design decisions over the entire life-cycle and product line of the architecture. Various cost modeling techniques may be used to perform this estimation. However, the estimation is often done in an ad-hoc manner, based on specific design scenarios or business assumptions. This situation may yield an unfair comparison of architectural alternatives due to the limited scope of the evaluation. A preferred estimation method would involve rigorous cost modeling based on architectural design cost drivers similar to those used in the manufacturing (e.g. process-based technical cost modeling) or in the enterprise software domain (e.g. COCOMO).

This paper describes an initial study of a cost model associated with automotive electronic system architecture. The model's intended use is to evaluate system cost drivers in response to various architectural decisions (e.g. choosing a communication bus topology or mapping a function to hardware). The primary cost driver categories explored are design and development, part fabrication, assembly and in-service costs. The preliminary version of this cost model focuses on describing the key influences on cost, but not the entire mathematical model. The paper presents the cost model with the help of influence diagrams and illustrates the use of the cost modeling methodology through an automotive case study – a steer-by-wire system. As future work, we propose to build a cost model and supporting methodology that accounts for architecture evolution to address the issue of evolving architecture requirements as well as when and where to employ new technology in the architecture.

INTRODUCTION

The growing electronic content in automotive products has made *Electrical, Controls and Software* (ECS) architecture design a key strategic area in which OEMs must be competent. Due to new content and rapid technology evolution, automotive ECS architecture design requires engineers with extensive experience and technical savvy. As such, ECS architecting becomes as much an art as it is a science. The ECS architect must be able to understand the impact of design decisions on various system metrics, such as dependability, scalability, reusability and, of course, system monetary cost[2]. To meet the challenges of ECS architecture design exploration, a systematic approach that comprehends both qualitative and quantitative metrics should be developed to account for architecture trade-offs. This paper focuses on one of the architecture evaluation metrics – i.e. the system monetary cost - and offers a model to illustrate how primary cost drivers can cascade from architecture design decisions.

Various business case and costing methodologies (e.g., activity-based costing and balanced scorecard) exist for supporting product development and are championed by several groups in industry and academia. Our work on cost models is based on previous work and interviews with OEM experts from different parts of their organization. The typical business case analysis assumes as an input an estimation of capital investment, part cost and the return on the deployment of the architecture. This work is complementary to the business case analysis in the sense that the goal here is to estimate the investment and part costs of the electrical architecture. In particular, we base our analysis on two previous cost modeling and ECS architecture evaluation research studies discussed in the Related Work section.

[1] Electronic systems refer to the electrical subsystem within a vehicle which includes wiring, bus-node and sensor-actuator architecture.

[2] In a way monetary cost is the unifying metric across all other metrics.

The paper is organized as follows: We first present the modeling work upon which our approach has been built. Then we introduce the details of our cost model, its organization and examples of the calculations. Afterwards, we present a steer-by-wire architecture exploration case study that illustrates cost analysis with our modeling method. Finally we present our conclusions.

RELATED WORK

Technical Cost Modeling (TCM). Developed over the past two decades by Joel Clark and his staff at MIT, the TCM method [1] attempts at forecasting the primary cost drivers associated with the manufacture of a product by modeling the capabilities and constraints associated with the processes used. The goal of TCM is to create predictive models that illustrate how design and processing decisions, and changes to these decisions, affect expected manufacturing operational characteristics and their resultant costs. TCMs consist of derived statistical and industrial relationships that link a product design (e.g. a part's wall thickness) or process decision (e.g. stamping press tonnage) to processing intermediates (e.g. material consumption, scrap rates or cycle times) which drive the final modeled cost. The final cost drivers in the TCM method fall into two main categories: fixed and variable costs. Fixed costs of production, i.e. investments in tools, equipment and facilities necessary for production, do not scale in direct proportion with production volume. As production volumes increase, fixed costs are amortized over more units, and consequently, the unit costs show economies of scale. Variable cost categories, like direct/indirect materials, labor and energy, are typically consumed at a relatively constant rate based on part mass, cycle times and processing efficiencies, and tend to exhibit little cost scaling with production volume.

As the TCM method only focuses on primary cost drivers, not a complete set of cost factors (some items, such as inventory, logistics, sequencing, taxation, may be ignored), the models should not be used as an estimated part pricing. The strength of technical cost modeling is its use for comparing two or more competing designs, architectures or processes for manufacturing a product and determining an expected cost difference.

Axelson's Cost Model. This model [2] takes into account hardware design, software design and part cost of the components of an electronic architecture. A UML-based modeling technique is used to show the relation between an architecture and its components. The model does not appear to be sufficient for complex analyses like the ones needed for automotive architecture trade-off studies. While understanding that the perspective of cost (internal vs. external) is essential for this domain, there are additional important considerations like integration and in-service related costs. Axelsson argues that the model can be applied to more general architecture studies; however, the omission of part maintenance cost and flash related costs is an important drawback for automotive ECS architecture analysis.

PROPOSED COST MODEL

Defining ECS Architecture. In this paper, ECS architecture is defined as a set of electronic control modules, M , and a set of interconnections, I . For each module there is a set of software modules S, a set of hardware components O , packaging (PCB and housing) P and connectors C (wiring and pins to the outside). Each component can be of five different types: processors, analog sensors, analog actuators, secondary storage and peripherals. The interconnections consist of two sets: harnessing components H (e.g. cut-leads, connectors or splices) and bussed electrical center (BEC) components B (e.g. relays fuses and circuit breakers). This model is not comprehensive, but based on interviews and analysis of a number of architecture studies, it captures the key architectural parameters. For now, it is assumed that these parameters are adequate to estimate important cost intermediates, such as software development effort and flash programming time, and their impact on the relative cost of the ECS architecture alternatives.

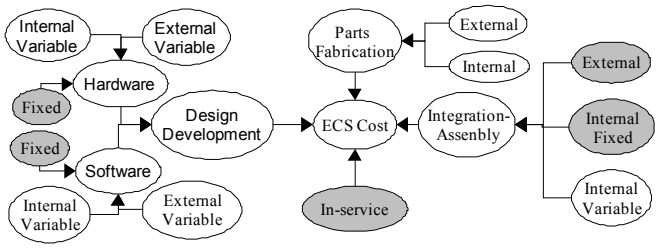

Figure 1: Overall ECS Cost Model[3]

Primary Cost Categories. The proposed cost model reflects the cost elements occurring during the phases of the ECS architecture life-cycle:

1. *Design and development cost* – the cost of specifying, engineering, and validating the design,
2. *Part fabrication cost* – the manufacturing cost for the parts and sub-systems of the architecture,
3. *Assembly cost* – the cost of integrating the parts into the vehicle during production, and
4. *In-service cost* – the cost of ownership resulting from repair and/or maintenance.

Given the increasing OEM interest in the role of software development, design and development cost was further divided into two sub-categories: software and hardware development cost. A further refinement of the cost model delineates internal (cost carried, incurred and driven by the OEM) and external (cost incurred by suppliers of ECS components) cost factors in addition to fixed and variable cost factors to reflect dependency on volume.

[3] The shaded ellipses denote that an explicit cost model has not been provided for these elements.

Figure 1 shows the initial categorization of primary cost drivers integrated into the hub of an influence diagram and illustrates the underlying structure of our ECS Architecture Cost Model. Figure 2 shows the categorization of the secondary cost drivers.

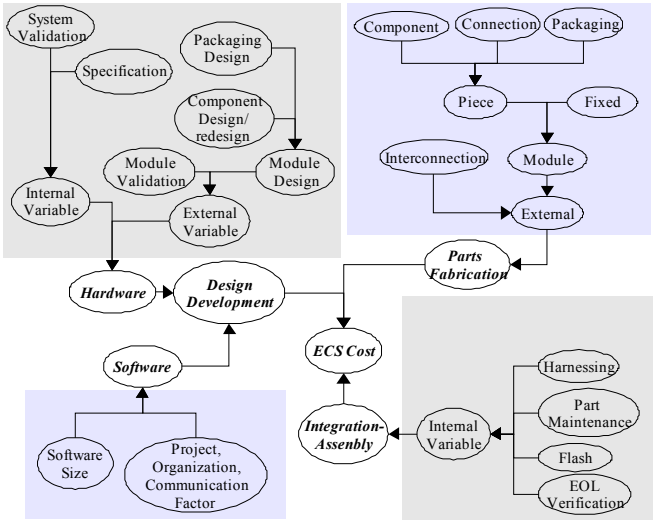

Figure 2: A More Detailed Overview

Product Line Cost. A *product line* is defined as the set of products that share a common, managed set of features that satisfy specific needs of a selected market. We call each of these products a *specific configuration*. The electrical architecture that implements the entire product line is called the *product line architecture*[4]. A product line architecture consists of a set of *architecture instances* where each architecture instance is an instantiation of the product line architecture with respect to a given configuration.

Consider the example in Figure 3. There is a product line with n unique configurations: $C_1, C_2 ... C_n$ with production volumes $V_1, V_2 ... V_n$ respectively. The architect decides on m product line architecture alternatives $P_1, P_2 ... P_m$ with n architecture instances for each alternative. The major trade-off that the architect has to contend with during the definition of the product line architecture is the degree of reuse of each primitive architectural element within the product line architecture. In one extreme the architect can go towards a product line architecture where every configuration has the exact same architecture instance vs. the other extreme of architecture instances that are unique for every configuration. In the former, there is a larger degree of reuse since the same components are used across all the configuration volumes. However, there is a penalty of over design for configurations that do not require all the architecture resources. This wasted resource is generally known as "give away cost". In the latter, each architecture instance is essentially optimized specifically for the configuration that it houses. Hence, the degree of component reuse is compromised but at the benefit of zero give-away cost for any configuration.

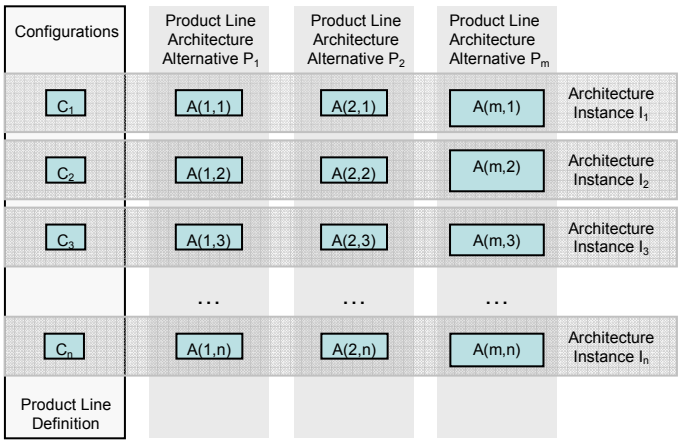

Figure 3: Relation between Configurations of a Product Line and Architectures Instances of a Product Line Architecture Alternative

The focus of this paper and the related case study is the cost model for an architecture instance of a product line architecture. Relating this to Figure 3, the cost model essentially allows the architect to evaluate the cost of an architecture instance that is represented by an element in a specific row and column such as $A(1,1)$ or $A(2,1)$ or $A(m,1)$. The architect can then evaluate the total product line cost by measuring the degree of reuse of all the primitive architectural elements of the product line architecture and then applying the cost model on all the architecture instances while accounting for the level of reuse of each architectural element. The total product line cost can then be used to compare between different product line architecture alternatives. An overall cost model and case study based on product lines is targeted as future work.

SOFTWARE DESIGN AND DEVELOPMENT COST

This category encompasses the cost of developing and validating the software that is executed in embedded modules within the ECS architecture. The ***fixed*** cost items in software development include IT infrastructure, software licenses and office space (both for OEM and suppliers) - all items that are relatively independent of the architecture chosen. For this reason, *software fixed costs are not explicitly considered in this paper, even though they should be included in a full assessment.*

Software ***variable*** cost (both OEM and supplier), e.g., the cost of the engineering effort for coding, depends greatly on architecture design decisions. The proposed model computes the variable cost (Figure 4) as a

[4] Product line architecture includes the description of structural properties for building a group of related systems (i.e., product line) through the description of components and their interrelationships. The inherent guidelines about use of the components capture the means for handling required variability among the systems. [7]

product of software development effort (man-months) and labor costs. The model calculations used to compute the software development effort are based on the widely accepted COCOMO model [3]. In COCOMO, the software effort is based on function points (the number of inputs, outputs, inquiries, file structures etc), the number of lines per function point (related to the software language) and engineering organizational factors. The organizational factors take into account a number of influences, like communication overhead, product complexity, the experience of the development team, and the availability of resources.

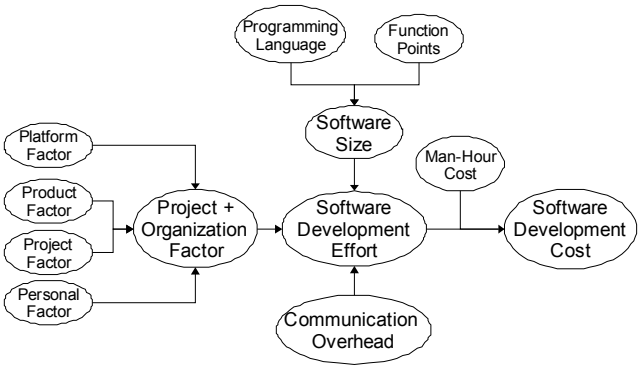

Figure 4 Software Cost Model

HARDWARE DESIGN AND DEVELOPMENT COST

The model includes the cost of specifying and designing the hardware and validating the operation of the integrated system. The ***fixed*** costs (internal and external) include items like IT infrastructure and personnel, tools, test benches, prototype set ups and management. As suggested above in software development, these fixed cost factors are assumed to be relatively independent of the architecture decisions and are not explicitly investigated in this work.

The ***variable*** costs of hardware design consist of OEM engineering activities, like total system specification, documentation and validation, summed with supplier engineering activities, such as design, development, and validation of the ECS components and sub-systems.

The internal hardware design variable cost (Figure 5) is measured by the specification effort (in man-months) and the Electrical Engineering OEM engineering labor wage. The specification effort depends directly on complexity of functionalities, number of modules, software-hardware interfaces and new features, network topology etc. However, creating engineering effort models for the specification process requires intimate details about the OEM's engineering. OEMs should analyze historical data to generate trends and drivers for engineering effort within their own organizations. The internal ECS system validation cost, as in the case of specification, is obtained from respective effort and labor wages. The system validation effort is driven by a number of factors: protocol type, number of interfaces, number of interrupt routines, software design methodology, concurrency, type of testing, number of IO's, number of sub components, network controller etc. Both the specification and validation effort models can be refined into more explicit mathematical forms driven by architectural design decisions, but are only discussed qualitatively above.

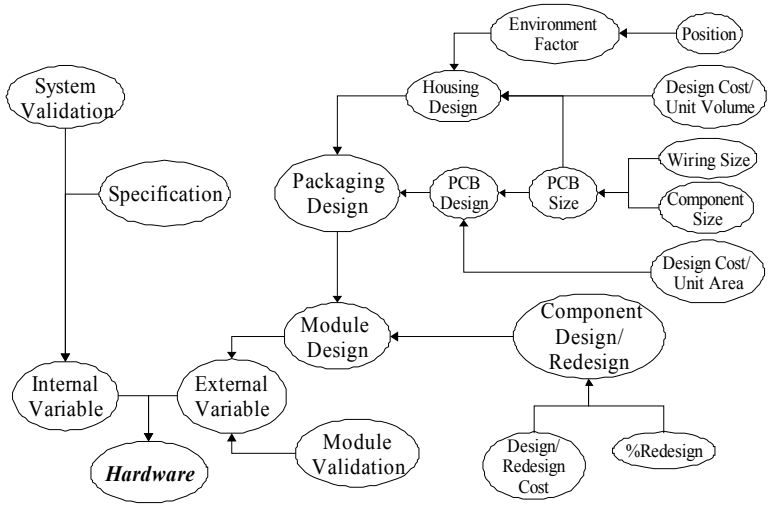

Figure 5 Hardware Cost Model

The external variable cost (Figure 5) for hardware design is modeled as the sum of the control module design cost C_M^d, the validation and verification cost C_V^d and the wiring design costs (assumed small in this work as compared to the cost of designing modules and validation, and perhaps considered an integrated activity). The control module's design cost, C_M^d is the sum of the design cost of each individual module $m \in M$. The design cost of a module is the sum of the cost of designing / redesigning the sub-components C_O^d and the cost of designing the overall packaging C_p^d (the design of the interconnections has been assumed to be small or integral to the components' design). The packaging design cost for a module is $C_p^d = C_{pcb}^d + C_h^d$ where C_{pcb}^d is the PCB design cost and C_h^d is the housing design cost. The PCB design cost is the product of the component size (the sum of the size of each component) s_O, the wiring size (the space required to wire components of a certain size and provided as a percentage) w and PCB design cost per unit area c_{pcb}^d. The housing design cost C_h^d is the product of PCB size ($s_O \cdot w$), the environment factor (e_f) determined by the location of the module and the housing design cost per unit volume, c_h^d. The design / redesign cost C_O^d of a module m is the sum of the design/redesign cost of the set of components $O(m)$. The design/redesign cost for a component is decided by the change or rework required; the amount of rework required may be 100%, implying that a component has to be designed from scratch. For a standard off-the-shelf processor, the design cost is assumed to be negligible. External component/sub-system validation variable costs are expected to be driven in a manner similar to the OEM as detailed in the paragraph above.

PARTS FABRICATION COST

Parts cost include the manufacturing of parts required for the production version of the ECS architecture. As the current supply chain for ECS components relies almost exclusively on suppliers for fabrication; internal part fabrication is not explicitly modeled here. Because fabrication costs are dominated by external supplier manufacturing activities, the ***fixed*** costs of production (investments) are included as a portion of the ***variable*** cost per piece. The model for external part fabrication (Figure 6) is divided into two sub-categories: cost for control modules, C_M^p and cost for interconnection components, C_I^p.

Part cost for the control module

Each module's part cost is the sum of the tooling cost C_{tm}^p and the piece cost C_{pm}^p. The tooling cost is accounted as the share per piece computed from the tooling investment and production volume. A good assessment of tooling investment can be done by curve-fitting techniques on historical data on the basis of design properties (like complexity level, newness, number of module functionalities etc). The piece (variable) cost of the module is $C_{pm}^p = C_p^p + C_i^p + C_o^p$ where C_p^p is the cost of packaging, C_i^p is the cost of inter-connection and C_o^p is the cost of the sub-components.

Module Packaging Cost

A module's packaging cost $C_p^p = C_{pcb}^p + C_h^p$ where C_{pcb}^p is the PCB cost and C_h^p is the housing cost. The PCB cost can be calculated as the product of the sub-component size s_O, the wiring size (accounting for the space required to connect components together) w and the PCB cost per unit area c_{pcb}^p. The housing cost can be computed as the product of the PCB size ($s_O \cdot w$), the environment factor, e_f, driven by the location of the module and the cost per unit volume of housing, c_h^p. Refer to [2] for a detailed review of methods for calculating packaging costs.

Module's Sub-Component Cost

This cost is the sum of the part cost of individual sub-components of a module. The cost model for a sub-component can vary widely, so a proper categorization is very important. The cost of a processing unit can be calculated as the sum of the processor, the memory and the flash cost. The processing costs are driven by the effort, invocation rate, speed and type, while the memory cost is determined by the data size and the program size (see [2] for a mathematical framework relating cost to these design factors). The processing unit's flash cost is related to the effort to upload the boot and operating software and the calibration sets. The supplier's flash cost often depends on the technique being used, e.g., the cost-per-byte for in-line flashing techniques is negligible, whereas the cost is proportional to the size of the software if the part is flashed at the end of the manufacturing process.

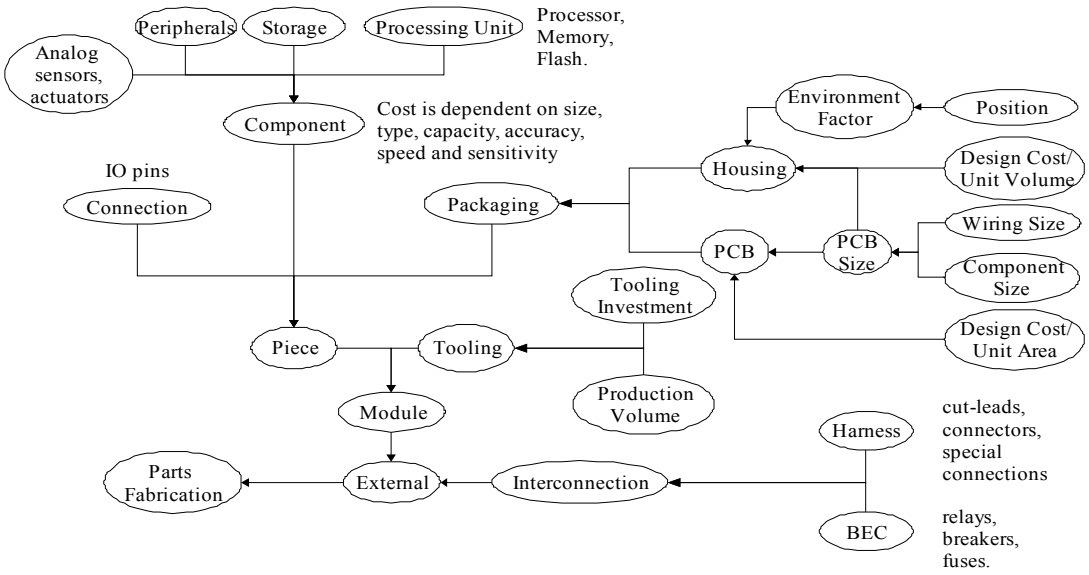

Figure 6 Part Fabrication Cost Model

Other sub-components may have simpler cost models. Analog sensors and actuators are standard components; costs are often determined by the type (the physical element being measured or controlled) and required sensitivity (accuracy of the reading and speed of access), and can be read from tables. Secondary storage modules may also be purchased as standard components; their cost is driven directly by the memory type (EEPROM, DRAM) and the capacity. Modeling the cost of peripheral components may require calculations for special items, from network controllers to timers, power supply circuits to DRAM to I/O circuits. A detailed model for each peripheral is beyond the scope of this work, but may be undertaken as a separate effort to identify architecture design cost drivers.

Cost model for Interconnecting elements

Interconnection costs are calculated as $C_I^p = C_H^p + C_B^p$,where C_H^p is the part cost for harnessing and C_B^p is the part cost for BEC components. The harness cost is computed as C_H^p = $C_{cl}^p + C_{co}^p + C_{sp}^p$, where C_{cl}^p is the cost of the cut-leads, C_{co}^p is the cost of the connectors and C_{sp}^p is the cost of specialized components (e.g. splices, gold-plated contacts, etc). Calculating the number of cut-leads is standard from the position of modules, number of pins and connectivity information. While the cost per cut-lead can be computed, the process is OEM-specific and requires rigorous modeling; hence we refrain from modeling the cost and assume that the cost per cut-lead is provided. The number and cost of connectors and specialized components is calculated in a similar manner. C_B^p is the cost of the relays, circuit breakers and fuses used for an architectural design; an estimate of the number of these parts can be projected based on past designs.

ASSEMBLY COST

The ECS architecture assembly cost includes the cost associated with connecting the parts and placing them into the vehicle body. The connection cost for supplier parts and sub-systems has been accounted for in part fabrication; only the internal assembly cost will be modeled here. We assume here that many internal ***fixed*** cost factors (e.g. plant space, tools, and equipment) would be nearly identical for ECS architectures with the same functionality. These costs are not explicitly calculated, but can be explored in future models. The proposed cost model calculates the internal ***variable*** assembly cost (Figure 7) in five major categories: placement, harnessing, part maintenance, flash and end-of-line (EOL) verification.

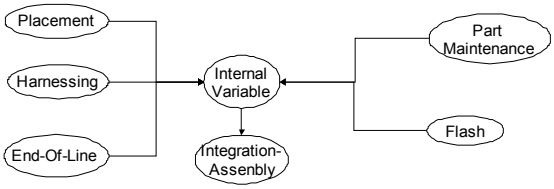

Figure 7 Integration Cost Model

The placement cost includes the cost of inserting the system into the vehicle, including fastening, attachment and labor to perform the operations. This cost is computed as the product of the number of modules and an average cost per module. The harnessing cost is calculated as a product of the number of harnesses and the average cost of harness placement. EOL verification is driven by the effort to check the functionality after assembly. Discussions with OEM experts indicate that the EOL verification cost is relatively independent of the architecture design decisions if the feature set remains constant.

Part Maintenance Cost

This cost is driven by the engineering and production effort to keep track of new part numbers. For a hardware component, the part maintenance cost is accounted for when a new part is introduced into the plant. The maintenance cost of new software is driven by the introduction of new calibration sets to each hardware component, which can act like a multiplicative factor on new part numbers. Since the burden of managing part numbers in production is OEM-dependent, explicit calculations will not be shown in this paper. Our model assumes that information on part maintenance cost and on whether a part is new is provided.

Flash Cost

A plant's internal flash cost is computed from the flash time required by all the control modules and the flashing cost per unit time. The flash time is dependent on a number of factors like: parameters for control modules (e.g. the number and size of calibration sets), parameters for buses (run-time latency) and parameters for the network. The flash cost-per-unit-time is computed from the number of workstations, the cost per workstation, production capability, the upper limit of flash time and the number of buses, and is independent of design decisions. The unit cost may not be a constant, and is usually represented as a step function. It is beyond the scope of this paper to give a mathematical formula for computing the flash time or unit time cost; however these are explicitly studied by OEMs and data are easily available. The flash cost is included in the cost model, since design decisions, such as the choice of network controllers and software to hardware mapping, affect the flash time requirements of the architecture, thereby changing the total cost.

IN-SERVICE COST

From discussions with OEM experts, in-service cost has been only qualitatively related to architectural attributes. It may be more appropriate that this cost aspect be looked at on a scenario-specific basis. This modeling gap may motivate further research into quantitative links between architecture design attributes and quality and between reliability and maintenance costs. The following scenario illustrates why modeling the link between in-service cost and ECS architecture may become very important.

The cost of replacing a module has been shown to be much higher than reprogramming when ECS faults occur. Acknowledging this, an ECS architect considers two modules: a ROM-based, and a re-flashable EEPROM. If the ROM-based module has a problem (whether hardware or software) it must be replaced. On the other hand if software problems occur more often, the second module can be re-flashed in the service bay at a much lower service cost. Thus the choice between flashed EEPROM and ROM-based control modules will greatly affect the repair and subsequent warranty costs of an ECS system. With access to quality, warranty and vehicle maintenance data, it may be possible to derive a mathematical framework linking architecture design decisions to expected in-service costs.

CASE STUDY

The case study is based on an ECS architecture for steer-by-wire control (Figure 8). The basic concept of any steer-by-wire architecture is the removal of the mechanical linkage between the steering wheel and the wheel. The mechanical linkage is replaced by a set of steering wheel angle sensors, an electric motor that controls the wheel angle, road wheel angle sensors and some electronics that compute the closed loop control system. To maintain a realistic road condition feel for the driver, there is also a force feedback actuator on the steering wheel. The specific architectural alternatives used here have been studied using dependability as quality metric [4]. In this paper, we rank architecture alternatives on the basis of monetary cost attained from using the proposed cost model. One of the key assumptions we make in instantiating the cost model is that there is no design giveaway; i.e., all the architectural alternatives have sufficient IO capability to meet the functional requirements. The analysis is done by capturing the architectural attributes for each alternative, instantiating the cost model based on the cost elements that are impacted by the delta[5] difference in the architectural attributes of the alternatives, and then finally analyzing the level of the impact of the architectural differences on the instantiated cost model.

One of the cost elements we evaluated was the cost of redesigning a component. The approach we took is based on evaluating the degree of modification in terms of the delta change in software components, hardware components and hardware IOs with respect to the readily available components in the library or on the “shelf”.

The **baseline** architecture (referred to as B) is composed of three redundant steering angle sensors which are sampled by a set of supervisory ECUs in a quad redundant configuration (Figure 8). Each supervisory ECU has a direct CAN [5] link to a motor control unit (MCU), which in turn actuates the wheel angle motors. The supervisory ECUs communicate through a Flexray [6] communication bus. We will assume that for MCUs the available library unit has one CAN interface; for ECUs the available library element has one CAN and one Flexray interface. Any software module for co-ordination between the control units has to be designed. Based on this, for the base architecture

[5] The application of a delta-analysis approach should be proven mathematically or derived through a formal process. The delta-cost analysis approach and proof is outside the scope of this paper. We will assume that a delta analysis approach is valid in this case study.

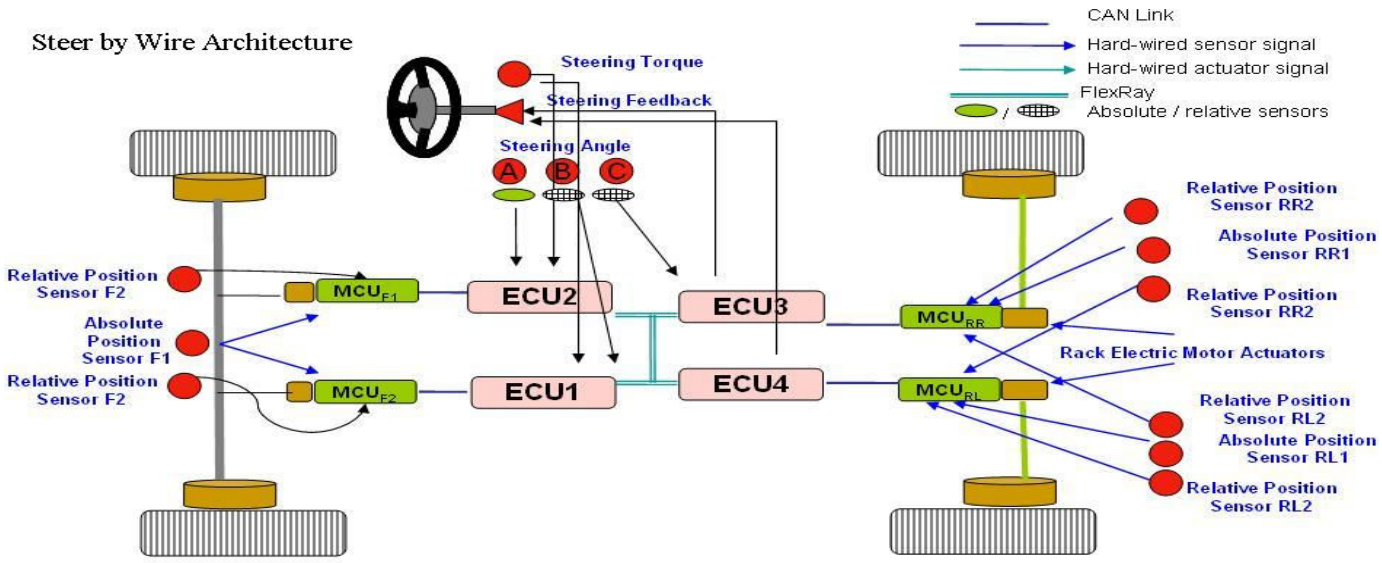

Figure 8 Steer-By-Wire Architecture

we need to design two software modules: one for the Flexray communication driver of ECUs and one to implement the protocol to mitigate asymmetric faults.

In the **first alternative** (referred as A_1) the CAN links (from the baseline architecture) are removed and all the nodes - both the quad redundant supervisory ECUs and the MCUs - are linked using the Flexray bus. This implies the development of a software module for an MCU-based Flexray communication driver along with a software module for an ECU-based Flexray communication and a module to implement the protocol to mitigate asymmetric faults. The MCUs have to be redesigned by 50% and a new part number has to be added. The ECUs have to be redesigned by 25% and a new part number has to be added. The flash time (it is assumed that all software is flashed inside the assembly plant) increases by 25% while the number of cut-leads (computed as the sum of cut-leads for CAN, Flexray and sensor, actuators) remains the same as the baseline.

In the **second alternative** (referred as A_2) CAN links are replaced by Flexray links, and only three supervisory ECUs in a triple redundant configuration are used. This causes three software modules to be designed: one for the Flexray communication driver for the ECU, one for the Flexray communication driver for the MCU and the third for the voting protocol by the three ECUs. The MCUs have to be redesigned by 50% and a new part number has to be added. Two of the ECUs have similar changes (25%) and the third one has a change of 37.5%; consequently there will be two new part numbers. The flash time and number of cut-leads remain comparable to the baseline.

Table 1: Architecture Attribute design Capture of SBW Architecture Alternatives

Architectural Attribute Comparison				
	B	A_1	A_2	A_3
# of Software Modules	2	3	3	2
# of New Parts	0	2	3	3
%redesign(MCU)	0	50	50	0
%redesign(ECU)	0	25	25+37.5	25+12.5
# of buses	5	1	1	5
# of Cut-leads	21	21	19	19
Flash Time (sec)	200	250	200	200

In the **third alternative** (referred as A_3) four ECUs are replaced by three ECUs (the communication scheme remains the same as the base). Two software modules need to be designed (one for the Flexray communication driver and one for the voting protocol). There is no redesign for the MCUs. Two of the ECUs are to be designed by 25% and the third one redesigned by 12.5%. The flash time and number of cut-leads remain comparable to the baseline.

Table 2 indicates the cost analysis result using the cost model. The mathematical details behind the analysis are omitted since they are not the main focus of this paper. The cost numbers are also omitted due to the proprietary nature of this information.

Table 2: Monetary Cost Analysis of SBW Architecture Alternatives

Cost Computation in ,000$				
	B	A_1	A_2	A_3
Software Design	225	315	288	198
Hardware Design	150	142.5	198.75	206.25
Part Fabrication	2524	3684	3226	2066
Integration	200	380	320	320
Net	3099	4521.5	4032.7	2790.2

Next the cost for each row of the table is analyzed:

1. A combination of Flexray and CAN is less expensive than using only Flexray. The main reason is the availability of CAN modules in the library and lower cost associated with CAN parts (in fact, the cost of microprocessors and network controllers for Flexray is much higher than CAN-based components).

2. The net cost of the base is comparable to that of A_3. The removal of one ECU certainly reduces the part fabrication cost; however introducing part numbers and redesigning hardware modules offsets the cost advantage gained in part cost.

3. The software and hardware development cost is not related. While software development costs of A2 and A3 are significantly more than the other two; the gap is much smaller for the hardware development cost. This is due to the fact that a higher number of buses significantly increases the system validation effort.

4. If the system-wide cost is considered, the result shows that the base and A_3 are comparable. This observation leads us to consider a new possibility: architectures may be chosen and shared among product families rather than targeting one single vehicle.

5. The cost study shows the most sensitive cost points of the architecture alternatives. For example, in the case of A1, if the part cost can be reduced significantly, the design will emerge as a winner.

In summary, this scenario clearly shows the value of the cost model, not only as an estimation tool, but also as a useful way to give hints to the architects as to what cost impacts their design decisions have. Instead of using an ad-hoc method that may miss significant cost impacts (e.g., packaging costs are commonly overlooked) or apply a cost metric that is not fairly biased towards one alternative, the proposed cost model provides not only a better understanding of the overall cost differentiation, but also a quantitative measure to choose an architecture in a systematic manner.

COST EVOLUTION

Until now the cost model does not have any notion of time. Based on our interviews, some of the major architectural decisions are driven by scalability requirements. For example, the ECU may be over-designed with multiple Flexray links in order to accommodate future redundancy requirements. Naturally this has a giveaway cost for vehicle configurations that do not require this additional hardware. However, given a long-term architectural evolution towards greater levels of safety criticality, this additional scalability will enable a less costly evolution in comparison to the complete overhaul of the architecture.

There are two forms of evolution that needs to be considered within the automotive domain:

1. *Technology evolution*. This is defined as the evolution in the capability and manufacturing efficiency of a specific technology element within the architecture. For example, the communication bus in the electrical architecture is evolving from a pure CAN-based architecture to a mix of CAN, LIN and Flexray. Figure 9 illustrates this concept, specifically in terms of the evolution of verification and software development cost relative to the serial data bus technology.

2. *Architecture evolution*. This is defined as the incremental design stages from a baseline architecture to the ideal state architecture. For example, if the long-term architecture strategy is a highly centralized architecture and the current architecture is a decentralized version, an evolution strategy needs to be defined. Figure 10 illustrates this concept.

Part 1 can be reflected within the cost model by including the time dimension. For example, the operational and architectural cost parameters are no longer constants, but a function of time. This function would be dependent on factors such as process improvements, manufacturing improvements and production volumes.

Part 2, on the other hand, has to be reflected as part of the architectural alternative. Previously we defined an ECS architecture alternative as a static composition of ECUs, buses and other elements. To account for architectural evolution, these elements are instantiated on a specific time step. The time steps define the discrete points in time where the architecture is incrementally modified in order to move towards the ideal state architecture.

Both these concepts are susceptible to uncertainty, since they try to estimate a future outcome. In general, this problem might be too difficult to provide a quantitative analysis methodology. Nevertheless, providing the architect with an influence diagram as to what considerations need to be given on specific architectural

decisions can provide a qualitative measure of goodness.

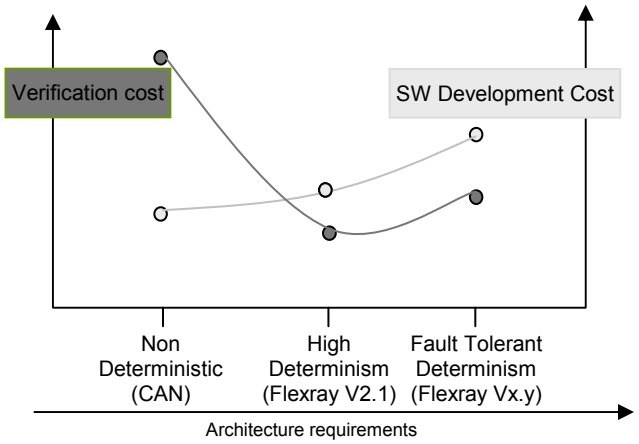

Figure 9: Automotive Serial Data Technology Evolution

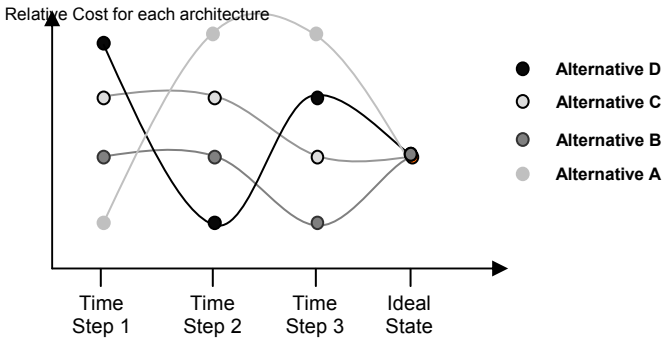

Figure 10: Architecture Evolution with Respect to Monetary Cost

CONCLUSION

We have presented a preliminary cost model for automotive ECS architectures that provides a system level view of the effects of design decisions on monetary cost. The model is based on influence diagrams which provide an effective visualization of relations. The model being modular in nature (any input can be refined to a more refined cost model without affecting the rest of the model), supports abstraction (in case of lack of input data in early design phases) and helps in understanding the appropriate cost metric scope to evaluate fairly the cost for two or more architectural alternatives. The case study illustrates the value of having such a cost model by indicating the cost factors that are impacted based on specific architectural decisions and by providing a fair way to evaluate architecture alternatives. In terms of future work, we have proposed the development of cost models that capture the notion of technology and architecture evolution.

ACKNOWLEDGMENTS

This work was supported in part by General Motors, in part by GSRC grant 2003-DT-660 and in part by the Center for Hybrid and Embedded Software Systems (CHESS) at UC Berkeley, which receives support from the National Science Foundation (NSF award #CCR-0225610), the State of California Micro Program, and the following companies: Agilent, DGIST, General Motors, Hewlett Packard, Infineon, Microsoft, and Toyota.

REFERENCES

1. R. Kirchain and F. Field. *Process-Based Cost Modeling: Understanding the Economics of Technical Decisions.* Encyclopedia of Material Science & Engineering, 2001.
2. J. Axelsson. *Cost Models for Electronic Architecture Trade studies.* In Proc. 6th International Conference on Engineering of Complex Computer Systems, Tokyo, Sep 1995.
3. B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby. *Cost Model for future Software Life Cycle Processes: Cocomo 2.0.* In Special Volume of Software Processes and Product Measurement, Annals of Software Engineering. J. C. Baltzer AG, Science Publishers, 1995.
4. C. Pinello. *Design of Safety-Critical Applications, a Synthesis Approach*. PhD thesis, Electrical Engineering and Computer Sciences, University of California, Berkeley.
5. *Road Vehicles - Interchange of Digital Information* – Controller *Area Network (CAN) for High-Speed Communication.* International Standards Organization (ISO). ISO Standard -11898, Nov 1993.
6. *Flexray Communications System Specifications 2.1*, 2005.
7. *SEI Product Line Definition*, http://www.sei.cmu.edu/productlines/glossary.html

CONTACT

Arkadeb Ghosal is with University of California, Berkeley, CA 94720 USA. (e-mail: arkadeb@ eecs.berkeley.edu).

Sri Kanajan is a senior research engineer at General Motors Research, Warren, MI, 48088 USA. (e-mail: sri.kanajan@gm.com).

Alberto Sangiovanni-Vincentelli is with University of California, Berkeley, CA 94720 USA. (e-mail: alberto@ eecs.berkeley.edu).

Randall Urbance is a former employee of General Motors, Warren, MI, 48088 USA. (e-mail: randall.urbance@gm.com)