# Assurance via model transformations and their hierarchical refinement

Zinovy Diskin, Tom Maibaum, Alan Wassyng, Stephen Wynn-Williams, Mark Lawford
McMaster University, McMaster Centre for Software Certification, Hamilton, Canada
{diskinz,maibaum,wassyng,wynnwisj,lawford}@mcmaster.ca

## ABSTRACT

Assurance is a demonstration that a complex system (such as a car or a communication network) possesses an importantproperty, such as safety or security, with a high level of confidence. In contrast to currently dominant approaches to building assurance cases, which are focused on goal structuring and/or logical inference, we propose considering assurance as a model transformation (MT) enterprise: saying that a system possesses an assured property amounts to saying that a particular *assurance view* of the system comprising the *assurance data*, satisfies acceptance criteria posed as *assurance constraints*. While the MT realizing this view is very complex, we show that it can be decomposed into elementary MTs via a hierarchy of refinement steps. The transformations at the bottom level are ordinary MTs that can be executed for data specifying the system, thus providing the assurance data to be checked against the assurance constraints. In this way, assurance amounts to traversing the hierarchy from the top to the bottom and assuring the correctness of each MT in the path. Our approach has a precise mathematical foundation (rooted in process algebra and category theory) — a necessity if we are to model precisely and then analyze our assurance cases. We discuss the practical applicability of the approach, and argue that it has several advantages over existing approaches.

## KEYWORDS

Assurance case, Model transformation, Block diagram, Decomposition, Substitution

**ACM Reference Format:**
Zinovy Diskin, Tom Maibaum, Alan Wassyng, Stephen Wynn-Williams, Mark Lawford. 2018. Assurance via model transformations and their hierarchical refinement . In *Proceedings of ACM Models conference (MODELS'18).* ACM, New York, NY, USA, Article 4, 11 pages. https://doi.org/10.475/123_4

## 1 INTRODUCTION

We understand *assurance* to be a demonstration that a complex system such as a car or a communication network possesses an important complex property such as safety or security with a sufficiently high level of confidence. We then write $S \models P_{\text{assr}}$, where $S$ stands for the system and $P_{\text{assr}}$ for the property; we say $S$ satisfies $P_{\text{assr}}$ or $P_{\text{assr}}$ holds for $S$. A technically more accurate formulation would say that $S$ satisfies $P_{\text{assr}}$ with acceptably high confidence if the system is used as intended. Exploiting the idiom of a *system of systems* for complex systems, we can say that assurance is about property-of-properties of system-of-systems.

A standard (and perhaps the only) practical engineering way to manage complexity is decomposition of the problem into subproblems; these subproblems are then themselves decomposed and so on, until a set of "atomic" problems whose solution is known is reached. The solution to the subproblems is then combined in a pre-determined way to solve the original problem. Different realizations of this idea for different contexts and in different terms are abundant in engineering, science, mathematics, and everyday life. Not surprisingly, the decomposition idea is heavily employed for building *assurance cases (ACs)* — documents aimed at demonstrating $S \models P_{\text{assr}}$, which are written by the manufacturer of $S$ and assessed by certifying bodies[1]. Building ACs based on decomposition is now supported by several notations and tools, primarily Goal-Structuring Notation (GSN) and Claims-Arguments-Evidence notation CAE. These methods and tools [1, 29] are becoming a *de facto* standard in safety assurance.

The combination of two ideas — delegating the assurance argument to the manufacturer and the decomposition approach outlined above — gave rise to the growing popularity of ACs, which have lately emerged as a widely-used technique for assurance justification and assessment (see, e.g., surveys [3, 25] on safety cases.). While we do believe in the power of both ideas, we think that the way of leveraging decomposition for assurance in GSN and CAE diagrams is confusing for two reasons. First, users of these notations typically intermix two decomposition hierarchies: functional/goal decomposition and logical decomposition, i.e., inference [5]. Second, data and dataflow, which we will show are crucially important for assurance, are left implicit in GSN/CAE-diagrams. While keeping dataflow implicit may (arguably) be acceptable for documenting design activities, it is definitely not acceptable in assurance and essentially diminishes the value of GSN/CAE-based assurance cases.

We propose another decomposition mechanism based on model transformations. The idea is based on three observations 1-3) described below.

1) We notice that saying $S \models P_{\text{assr}}$ means that data about the system relevant for assurance, $D_{\text{assr}}$, satisfy some relevant constraints, $C_{\text{assr}}$; that is, we ***define*** $S \models P_{\text{assr}}$ to be the statement $D^S_{\text{assr}} \models C_{\text{assr}}$, where $\models$ can be read as either *satisfies* or *conforms to* – a phrasing often used in the context when properties are seen as constraints. To simplify notation, below we will omit the superscript $S$ if it is clear from the context. Importantly, data $D_{\text{assr}}$ are to be computed from "raw" data about the system $D_0$ (think of physical parameters of a car, or technical parameters of a network) rather than being


*MODELS'18, Oct 2018, Copenhagen, Denmark*

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

[1] The latter can be an independent agency or a group of experts at the manufacturer disjoint from the AC writers.

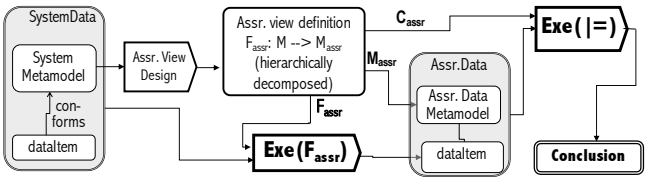


**Figure 1: MTBA Architecture of Assurance**

given immediately: $D_{\text{assr}} = f_{\text{assr}}(D_0)$, where $f_{\text{assr}}$ refers to an *assurance function* that inputs the raw system data and returns assurance data. Thus, the top assurance claim $S \models P_{\text{assr}}$ is rewritten as a data conformance statement $f_{\text{assr}}(D_0) \models C_{\text{assr}}$.

2) Next we notice a principal distinction between a definition and an execution of a function, which is important for assurance. Function $f_{\text{assr}}$ can be seen as an execution of a model transformation (MT) definition $F_{\text{assr}}$ for data $D_0$, i.e., $f_{\text{assr}}(D_0) = F^{\text{exe}}_{\text{assr}}(D_0)$. Also, MT are defined over metamodels and can even be specified as mappings $F_{\text{assr}}$: $M_0 \rightarrow M_{\text{assr}}$ (see [9]), where $M_0$ and $M_{\text{assr}}$ are metamodels for, respectively, system data and assurance data. Thus assurance can be seen as a special view of the system, where $F_{\text{assr}}$ is the view definition and $f_{\text{assr}}(D_0)$ is the view execution. Below we will often call transformation $F_{\text{assr}}$ an *assurance view*. Moreover, as metamodels typically include constraints, we can include assurance constraints $C_{\text{assr}}$ in $M_{\text{assr}}$ and thus reformulate the assurance problem $S \models P_{\text{assr}}$ as a typical MT problem: does the result of a transformation satisfy some predefined constraints encoded in the target metamodel?

The workflow described above is specified in Fig. 1 as a block diagram, whose nodes, shaped as directed rectangles, refer to processes/functions and rounded rectangles refer to (meta)data; as usual, data consist of a structure of data items that conforms to the metamodel (think of a data graph typed over the type graph so that the constraints are satisfied). Arrows show the dataflow in and out of processes. The diagram also shows a new Assr. Design Block discussed below.

3) In typical assurance domains such as safety, functions $f_{\text{assr}}$ are extremely complex and from different analyses (hazard and risk assessment and the like), whose results are subjected to complex verification and validation procedures. Hence, the correctness of $f_{\text{assr}}$'s definition $F_{\text{assr}}$ is a major issue. Here is where our third observation applies: the decomposition mechanisms successfully applied in those domains where functional block diagrams are used, such as signal processing and control theory (any block can be substituted by a block diagram with the same input and output ports), can be applied for MTs – indeed, functional blocks are transformations. Thus, we can decompose $F_{\text{assr}}$ into smaller and smaller components until we reach the level of simple functions whose correctness can be verified by simple means. If all decomposition steps are properly validated, and the execution of all transformation at the bottom of the hierarchy is properly verified, we can assure that the entire top transformation $F_{\text{assr}}$ has been properly executed and produced correct results $D_{\text{assr}}$. The final constraints check of Exe(|=), i.e., whether $D_{\text{assr}} \models C_{\text{assr}}$, is usually a simple procedure whose assurance is not problematic. [2] Thus, assurance can be viewed as establishing the correctness of a complex model transformation via its hierarchical decomposition – hence, the title of the paper. We will refer to the framework outlined above as the ***Model-Transformation Based Assurance*** (MTBA).

Our plan for the paper is as follows. In the next section, we present an overall view of MTBA, and based on it, explain the content of the technical part of the paper (Sect. 3, 4 and 5). Sect. 6 is a discussion of the possible practical applicability of MTBA. Sect. 7 is Related and Future work, and Sect. 8 concludes.

## 2 MTBA IN A NUTSHELL

Many assurance techniques rely on requirement decomposition: the high-level assurance requirements for the system are decomposed into the corresponding requirements for subsystems and further on until we reach the level of elementary components. (E.g, in safety assurance, these high-level requirements are called safety goals, in privacy protection – standardized NIST controls, and in security, the set of system level requirements is specified in the Protection Profile—a document identifying security requirements for a class of security devices.) The decomposition is mainly based on design patterns and supported by mathematical models so that the assurance argument can be close to formal, if the mathematical complexity is manageable, or is supplemented by testing and/or model checking and similar techniques otherwise. We will refer to this part as *inferential assurance (IA)*.

Yet, however successfully we manage to decompose each system level goal, the system goals must themselves be validated to ensure that they are suitable (e.g., complete)—indeed, any formal procedure begins with assumptions taken for granted (cf. axioms of ancient Greeks). The only way to "prove" such assumptions is to rely on observational or experimental techniques. Indeed, an assurance case must have all assumptions validated through experimental evidence - no loose ends! Often, this experimental justification is validated by previous experience and expert opinion, but the use of such expertise has to be properly organized and documented, and is based ultimately on observational information. We will refer to this aspect of assurance as *procedural assurance (PA)*, since it consists of well organized and disciplined procedures (hazard analysis in safety assurance, security and privacy threat analyses) that can provide enough confidence in the result.

Finally, having validated the top set of requirements and their decomposition, to assure the correctness of the entire assurance view execution, we need to verify the correctness of computations at the very bottom level. E.g., we need to be sure that the library of linear algebra operations we used for building the case was properly tested, and the computer where this library runs does not have hardware problems that could affect the results. We will refer to this part of the procedure as *computational assurance (CA)*.

Separating assurance into three parts IA, PA, and CA is not absolute. They are interwoven, and procedural aspects are important

[2] If the last check result is negative, traceability links that are assumed to be maintained by all operations and processes involved, would allow us to locate the source of assurance violation.

in all assurance activities so that IA and CA are (or should be) embedded in PA. However, these terms are convenient for referencing specific aspects of assurance.

In the next section, we consider a simple example illustrating the notions discussed above but specifically tailored to show intricacies of the procedural assurance and the value and role of metamodeling. In Sect. 4, we consider the inferential assurance with a simple example of embedded software safety, and show how signal processing can be formalized in algebraic terms of tensor categories developed by the category theory community. Sect. 5 summarizes these ideas and shows how the hierarchical structure of assurance can be mathematically specified and reorganized on algebraic foundations borrowed from data refinement, program refinement and model transformation. We thus obtain a common mathematical foundation for assurance, making it amenable to solid computer support – after all, MT is a well developed technological domain.

## 3 PROCEDURAL ASSURANCE (PA)

The inferential assurance part of the example of this section is very simple, at least in our toyish view of the case, but the procedural part is interesting enough to demonstrate the nature of procedural assurance.

### 3.1 Getting started

Suppose a financial consulting agency advises its clients on how much money they should keep to sustain a desirable life level for a specified time period without a paycheck. To obtain a license, the agency needs to submit to a government regulator an AC demonstrating that policies the agency suggests do ensure financial security for its clients. A real AC would be a complex document specifying the procedure of computing a boolean value based on the data about the client including spending habits, about client's assets and expenses, and about the investment market.

We will consider an overly simplified model of such a case based on the metamodel in Fig. 2. The system of interest comprises a client (person) $X$ with $X$'s banking accounts and expenses, while its environment comprises employment and the investment market, which affect $X$'s assets, and other factors affecting the expense cost. Only three attributes of class Person are shown: rskType describes the financial behaviour type of a person — cautious, reckless, or medium (in-between); attribute edType evaluates the education level — high, low, or medium, which, supposedly, affects the employability of the client; and attribute crit(ical)Time provides the number of months the client wants to support her level of living without a paycheck. The two other classes and three associations in the metamodel are self-explainable (the type of account can be Checking, Saving, or Money Market).

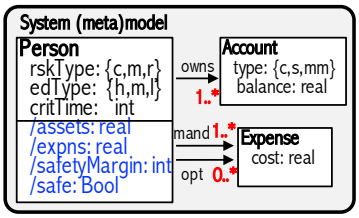


**Figure 2: Metamodel for the example**

The assurance function $f_{\text{assr}}$ consists of two functions, $f_{\text{asset}}$ and $f_{\text{exp}}$, which compute $X$'s assets and expenses in some way — more or less straightforward for assets and possibly complicated for expenses. (These functions are shown as operations 'assets' and 'expns' in the class diagram.) The agency evaluates the financial situation of client $X$ as acceptably secure if, for some some safety margin sMargin (e.g., 50%),

$$((X.\text{assets} - X.\text{expns})/X.\text{assets}) * 100\% > \text{sMargin} \qquad (C_{\text{assr}})$$

Thus, assurance data for $X$ is a triple of values

$$D_{\text{assr}} = (f_{\text{asset}}(D), f_{\text{exp}}(D), f_{\text{sMargin}}(D))$$

with $D$ being the system's data, which are subjected to the final conformance check w.r.t. assurance constraint $C_{\text{assr}}$ specified above.

Finding proper definitions $F_{\text{asset}}$, $F_{\text{exp}}$, and $F_{\text{sMargin}}$ for the corresponding functions above may be an issue, especially for $F_{\text{exp}}$ and $F_{\text{sMargin}}$. Decomposition based on previous experience and design patterns found are normally used in such inferential assurance problems – we will discuss this in more detail in Sect. 4. For the present section intended to show intricacies of the procedural assurance, we first notice that the rskType attribute of a person, which is an important input for the functions above, is actually not a raw data item and to be evaluated based on some non-trivial analysis rather than computed. Fig. 3 presents a simple model of such an analysis. Any client has to engage in an interview, in which her financial information is collected; in addition, the client fills-in a questionnaire about her financial behaviour, which is then carefully examined and evaluated and results in establishing the client's rskType and criticalTime. Perhaps, one or two more iterations are needed (note the feedback loop). These data then go to the computational/inferential blocks $F_{\text{exp}}$ and $F_{\text{sMargin}}$, which finally compute assurance data $D_{\text{assr}}$ typed over the metamodel $M_{\text{assr}}$. The assurance constraint $C_{\text{assr}}$ is specified as an invariant; in this way, the agency can monitor the situation (which can change as the environment changes) and make sure that all its client are financially safe (e.g., by setting triggers and actions on the violation of the constraint).

Note an important feature of the dataflow shown in Fig. 3: the four metamodels in different data-places are interrelated: the class Person they refer to is to be the same class, and there may be more complex relationships if more complex structures are involved – we will see that this is indeed the case in the next subsection.

### 3.2 Procedural assurance in practice

Within our model described above, the behavioural type of a person (modelled by rskType), i.e., whether she is financially cautious or reckless or in-between, is an important datum for assuring financial security. However, assigning such a type for a client $X$ may be a challenging issue as people behave differently in different contexts. Why did we assume that having three types is sufficient to compute a reliable assurance criterion, while perhaps, we need a 5-valued scale? Or maybe we need to consider the behavioural type as a compound attribute consisting of several subtypes and "measured" by a tuple of values? Whatever choices we have made for assigning such a type to $X$, its correctness cannot be formally proved. However, we can try to justify the assignment by referencing an established assignment procedure.

A simple model of such a procedure is shown in Fig. 4 as a hybrid diagram that combines workflow and structural metamodel-based

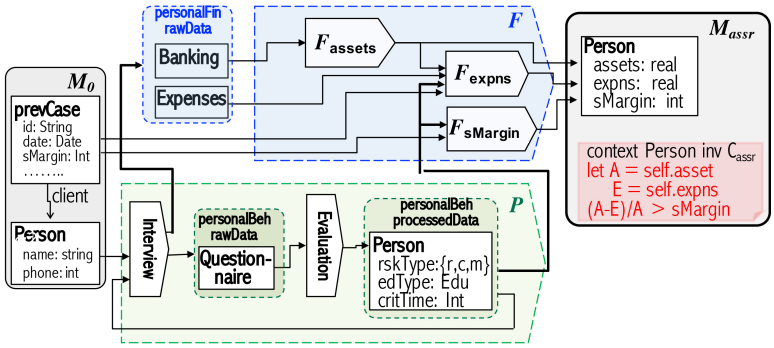


**Figure 3: The process in detail**

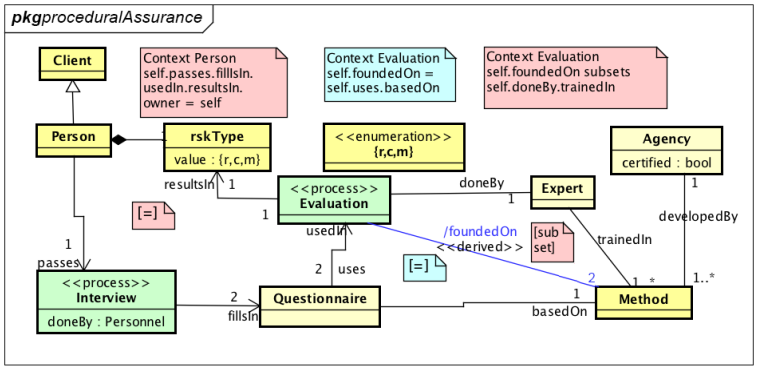


**Figure 4: Workflow and metamodels together**

specifications. Formally, it is a metamodel but classes annotated with stereotype 'process' (green with a color display) are actually instantiated with executions of the respective processes and can be considered as graphs of the respective "functions". Directed association into and from these classes show dataflow channels while purely structural associations are undirected. We consider that a client should only pass one Interview (for simplicity, feedback loops specified in Fig. 2 are not allowed), which is conducted by a trained Personnel, and fills-in two Questionnaires, each one being based on a Method developed by a certified Agency. The rskType assignment is the result of an Evaluation, and the commutativity constraint [=] declares that the resulting rskType is an attribute of that person who passed the interview – a precise description is given by the OCL constraint on the top-left.

The Interview is Evaluated by an Expert, who should be trained in the Methods used for developing the Questionnaires used in the Interview. We require that two questionnaires used in an interview were based on different methods. To specify this constraint, we first specify a derived association 'foundedOn' from class Evaluation to class Method as sequential composition of two associations as specified by the OCL query in the top-middle of the diagram (blue with a color display). Note that the derivable multiplicity of this association is to be 1..2 as two questionnaires can be based on the same method. To exclude such a case, we replace the derived multiplicity 1..2 (crossed-out in the diagram) by multiplicity 2. Finally, to ensure that the Expert is indeed trained in using the methods the questionnaires are based on, we add to the metamodel the constraint [subset], which states that for an evaluation *self*, we require that set of methods *self*.foundedOn (= *self*.uses.basedOn) to be a subset of *self*.doneBy.trainedIn (the OCL constraint in the top-right of the diagram). We see that an accurate description of a relatively simple procedural assurance activity needs complex metamodels with complex constraints.

A similar narrative is undertaken for justifying the assurance constraint $C_{\text{assr}}$ considered in Sect. 3.1. We need a method for setting the safety margin sMargin. We need an analysis of what constraints (assurance goals) we should use to make our assurance case more reliable. For example, we may need to require at least 75% of the client assets to be placed in low risk investment funds, which at once brings the issues of assuring that 75% is an acceptable fraction, and that the estimation of low-medium-high risks of the investment funds involved was done by a certified agency and/or is based on another reliable source. Yet another similar constraint is to require, say, 90% of the assets to be insured by a Federal Regulator. Then our set of assurance goals would consist of three constraints, but the issue of completeness of this set still persists and, of course, cannot

be proved formally. We can, again, only rely on the procedure that would lead us to yet another complex metamodel with complex constraints similar to that one discussed above.

Note also that Fig. 3 actually encodes a small decomposition hierarchy: we may make a step towards more abstract workflow by considering blocks $F$ and $P$ as black-boxes and correspondingly hiding the inner structure of their input and output metamodels. We can make one more step up by framing blocks $F$ and $P$ into one block "processRawData" with corresponding input and output metamodels consisting of single classes: "rawData" and "assuranceData".

## 4 INFERENTIAL ASSURANCE (IA)

We will consider the inferential part of ACs with a simple example, and show how our main machinery—decomposition of a top assurance claim into a hierarchy of refinement steps—works. Sect. 4.1 describes a simple example of goal decomposition done "manually" due to its simplicity, and demonstrate how convenient it is to use block diagrams (BDs) for system decomposition. In Sect. 4.2, we show how such a decomposition can be encoded algebraically by term substitution by using standard categorical techniques. In Sect. 4.3, we combine system and goal-requirement decompositions into a hierarchy of what we call *inferential steps* and demonstrate that wiring (and dataflow over it) is a crucial component of the entire inferential system (a.k.a. assurance argument flow), which makes an essential distinction from standard GSN/CAE approaches.

### 4.1 Design via block diagrams

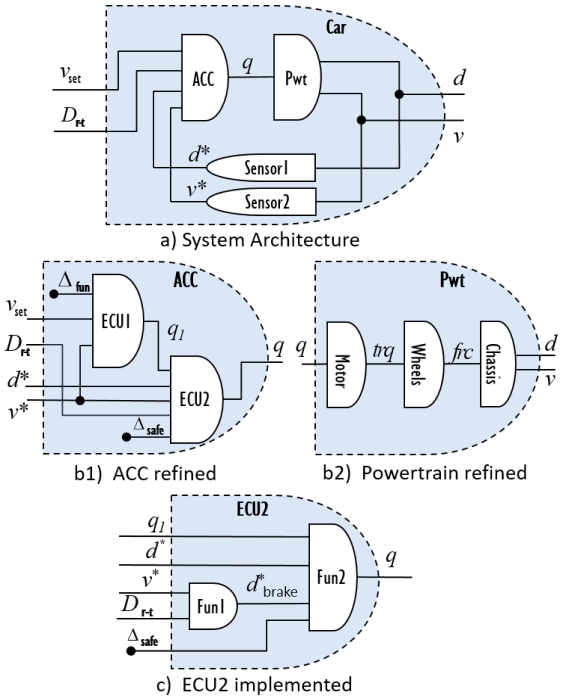


**Figure 5: Adaptive Cruise Control**

We will consider an oversimplified model of an embedded software system for Adaptive Cruise Control (ACC), designed to maintain the speed of the host car while remaining a safe distance away from the front (also called leading) vehicle. We choose to model a system that is intelligent enough to determine for itself the minimum distance required to come to a complete stop. The block diagram in Fig. 5(a) shows a high level view of the entire system (the host car in this case). The diagram consists of functional elements or blocks (e.g., ACC, Pwt, etc.), their interfaces (input and output ports), and the flow of data between them (wires). The sensors observe the environment and react according to the velocity $v$ of the vehicle (Sensor 2), and the distance $d$ to the nearest obstruction ahead of it (Sensor 1). The sensors produce their estimates of these values ($v^*$ and $d^*$ respectively) which are routed to the ACC system along with the desired speed $v_{\text{set}}$, and some data about the road and tires $D_{\text{r-t}}$ (road conditions, tire pressure, etc.). This information is used to determine an appropriate value for the electrical signal $q$ which is sent to the powertrain in order to accelerate (or decelerate) the car. The main functional requirement of ACC is to keep velocity $v$ equal to $v_{\text{set}}$ up to some acceptable margin as specified by condition ($R1_{v_{\text{set}}}$: ACC) below.

$$v \approx v_{\text{set}} \text{ with some functional margin} \qquad (R1_{v_{\text{set}}}\text{: ACC})$$

However, ACC is a safety critical system, and in addition to functional requirements, it must satisfy a number of *safety requirements*. For example, in order to avoid a collision with the leading car (e.g., if it suddenly and abruptly stops due to some *force majeure*) the distance kept by ACC is to be big enough to allow the full stop of the host car as specified by condition ($R1_{\text{br}}$ : ACC).

$$d \geq d_{\text{brake}} \text{ with some safety margin} \qquad (R1_{\text{br}} : \text{ACC})$$

To ensure the two conditions are satisfied, we decompose ACC into components, "decompose" the requirements into (sub)requirements for the components, and show that if the atomic components meet the atomic requirements, then ACC meets the "top" requirements above. The architecture of ACC is shown in Fig. 5(b1): it consists of two control units, ECU1 and ECU2.

ECU1 provides fulfillment of ACC's functional requirement, which is formalized as shown in ($R2_{v_{\text{set}}}$: ECU1): ECU1 takes the required data and outputs the corresponding electric signal $q_1$ for the Motor. It may employ standard feedback control techniques (via Sensor2) to ensure condition ($R2_{v_{\text{set}}}$: ECU1) holds.

$$v_{\text{set}} - \Delta_{\text{fun}} \leq v \leq v_{\text{set}} + \Delta_{\text{fun}} \qquad (R2_{v_{\text{set}}}\text{: ECU1})$$

ECU2 is designed to ensure ACC satisfies its safety requirement ($R1_{\text{br}}$ : ACC) by satisfying the (sub)requirement ($R2_{\text{br}}$: ECU2), where $\Delta_{\text{safe}}$ is some predefined safety margin.

$$d \geq d_{\text{brake}} + \Delta_{\text{safe}} \qquad (R2_{\text{br}}\text{: ECU2})$$

To achieve this, ECU2 is itself decomposed into two functional blocks as shown in Fig. 5(c). Block Fun1 is responsible for computing an estimate of the braking distance $d^*_{\text{brake}}$ for the given velocity and road-tire conditions in such a way that the condition ($R3_{\text{br}}$:Fun1) is met (where $d_{\text{brake}}$ is the unknown real braking distance).

$$\text{if data } D_{\text{r-t}} \text{ accurate, then } d^*_{\text{brake}} > d_{\text{brake}} \qquad (R3_{\text{br}}\text{:Fun1})$$

Block Fun2 compares this distance with distance $d^*$ up to some safety margin $\Delta_{\text{safe}}$, and if $d^* \geq d^*_{\text{brake}} + \Delta_{\text{safe}}$, Fun2 does nothing and transmits the electric signal obtained from ECU1 without change, $q = q_1$. However, if the inequality above is violated, Fun2 overrides $q_1$ and outputs an alternate value ($Q_{\text{br}}$) which ensures the car will slow down. In this way, if the path of the car is obstructed, ECU2 prevents a collision (and sends an alarming signal to the dashboard, which we didn't include into the diagram to keep it simple).

$$\text{if } d^* \geq d^*_{\text{brake}} + \Delta_{\text{safe}} \text{ then } q = q_1$$
$$\text{else } q = Q_{\text{br}} \qquad (R4_{\text{br}}\text{:Fun2})$$

Finally, if the last two requirements, ($R_{\text{d}}$: Sen1), and ($R_{\text{v}}$: Sen2), for Sensors 1 and 2 are satisfied, an easy deduction over inequalities allows us to conclude that safety requirement ($R1_{\text{br}}$ : ACC) is satisfied (the assumption that if $v^*_1 > v^*_2$ then $d^*_{\text{brake}}(v_1) > d^*_{\text{brake}}(v_2)$ is taken for granted).

$$d > d^* \qquad (R_{\text{d}}\text{: Sen1})$$
$$v^* > v \qquad (R_{\text{v}}\text{: Sen2})$$

Thus, we have decomposed the system into subsystems and the top safety goal into (sub)requirements, and proved that the system meets its safety goal as soon as the subsystems satisfy their requirements.

## 4.2 The Algebra of Decomposition

Fig. 5 clearly shows the compositional idea behind block diagrams: a collection of connected components is itself a component, and in any context where a block is expected, a suitable block diagram may be used instead. For example on the system architecture level only the interface of ACC is known, however any block diagram satisfying that interface (e.g., Fig. 5(b1)) may be used in its place. By showing that the connection information present in block diagrams can be achieved by a set of fundamental operations on components, and that these operations form an algebra over components, this intuitive substitution can be formalized as algebraic term substitution in that algebra. The operations required to represent this connection information are now well understood, see, e.g., [2] or the book [7]; the latter shows that with a certain discipline of drawing block diagrams, intuitive graphical manipulations with them have precise formal algebraic meaning in the language of various specialized monoidal categories which we will refer to with the blanket term *tensor categories*.

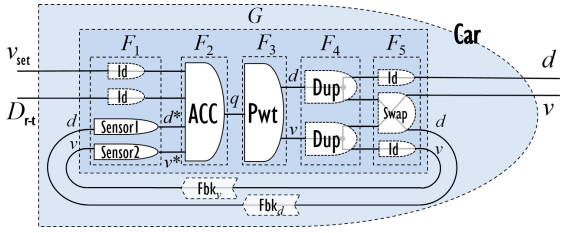


**Figure 6: Block diagrams as algebraic terms**

An example of representing a block diagram as an algebraic term is illustrated by Fig. 6. We represent the block (function) Car by combining its components using three operations: parallel composition, sequential composition, and feedback. These operations are used along with three predefined "signal routing" components (**Id**, **Dup**, and **Swap**) to describe the connections between atomic parts (ACC, Pwt, etc.). The signal routing components do not modify the values of the signals, and are therefore drawn in a semi-transparent way to show the wiring they aim to replace.

To represent the complex term for Car, it is broken down into sub-terms which are then combined. For example, the sub-term $F_1$ represents the composition of four blocks working in parallel and is written as: $F_1 = \mathbf{Id} \otimes \mathbf{Id} \otimes \text{Sensor1} \otimes \text{Sensor2}$, where symbol "$\otimes$" denotes parallel composition (or *tensor product* in the language of tensor categories). The interface of this composed block integrates the ports of the internal blocks. The sub-terms $F_4$ and $F_5$ are defined similarly.

The sub-term $G$ is the sequential composition of terms $F_i$: $G = F_1; F_2; F_3; F_4; F_5$ with symbol";" denoting sequential composition. In order for two blocks to be composed sequentially, the output interface of the first block must be compatible with the input interface of the second: indeed, each input port must accept the same type of data that the corresponding output port produces. The set of input ports of $G$ is exactly the set of $F_1$'s input ports, and the set of $G$'s output ports is the set of $F_5$'s output ports. Finally, the term Car is $G$ "plus" two feedback loops, which we need to specify algebraically.

We present an algebraic encoding for adding loops, following the ideas of the now classical paper [12]. We introduce one more algebraic operation, **Fbk**, which connects one output of a component to a compatible input. For some block $B$ where signal $x$ is both an input and an output, we write $\mathbf{Fbk}_x(B)$ to represent the block diagram where these two ports are connected. In our example, Car is written as the term $\mathbf{Fbk}_d(\mathbf{Fbk}_v(G))$, or (finally) as:

$$\begin{aligned}\text{Car} \quad = \quad & \mathbf{Fbk}_d(\mathbf{Fbk}_v((\mathbf{Id} \otimes \mathbf{Id} \otimes \text{Sensor1} \otimes \text{Sensor2}); \text{ACC};\\ & \text{Pwt}; (\mathbf{Dup} \otimes \mathbf{Dup}); (\mathbf{Id} \otimes \mathbf{Swap} \otimes \mathbf{Id})))\end{aligned}$$

In the language of tensor categories, operation **Fbk** is called *trace*. There are alternative algebraic encodings for feedback loops, many of which can be encountered in [2] and [27]. Each alternative brings some additional structure which may (or may not) be useful depending on the application domain.

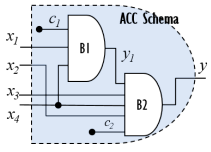


**Figure 7: Wiring schema**

For our purposes, it is sufficient to write block diagrams as terms in the following semi-formal way. For example, we write the block diagram in Fig. 5(b1) as a term ACC = $W_{\text{ACC}}$(ECU1, ECU2) where $W_{\text{ACC}}$ is a pattern (*wiring schema*) shown in Fig. 7: its elements are to be understood as variables of the corresponding types to be substituted by the corresponding constants, e.g., block ACC is obtained by substituting ECU1 for B1, ECU2 for B2 and $v_{\text{set}}$ for $x_1$ etc. Similarly, we can write Car = $W_{\text{Car}}$(ACC, Pwt, Sensor1, Sensor2), and so on. (This notation can also be accurately formalized as shown, e.g., in [28].)

Altogether, these formulas define the system design as a term (in a signature of tensor categories) given by its AST (abstract syntax tree) as shown in Fig. 8 – ignore the |=-half of the rectangle

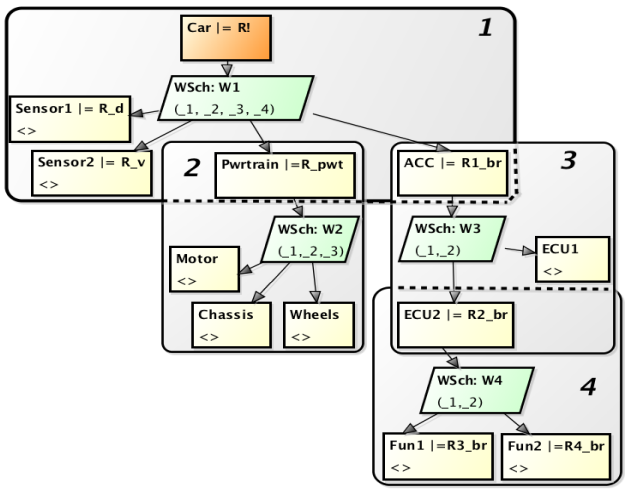


**Figure 8: System design as an algebraic term**

names for a while (e.g., read the label of the top node just as Car). Diamond nodes (green with a color display) refer to the schema of wiring the subsystems. The entire diagram is composed from four *decompositions steps* shown as grey round rectangles 1...4, and each steps can be identified by its top system $S_i$ so that $S_1$ = Car, $S_2$ = Pwt etc. It is easy to see that each decomposition step is just an algebraic equation

$$S_i = W_i(Sub_{i1}, ..., Sub_{in_i}) \qquad (\mathcal{DS}_i)$$

where $Sub_{ij}$ refer to subsystems of $S_i$. E.g., $S_3 = W_3(Sub_{31}, Sub_{32})$, where $S_3$ = ACC, $Sub_{31}$ = ECU1, $Sub_{32}$ = ECU2 and $W_3$ is given by Fig. 5(b1) (and can be specified as an algebraic term shown in Fig. 7).

The AST in Fig. 8 is almost exactly like ordinary ASTs, but with one important distinction: (rectangle) nodes in the tree correspond to arrows rather than objects and hence have source and target data places for data. The data side of the story will be important for us in Sect. 5.

## 4.3 From design to logic

*4.3.1 Logical flow via inferential steps.* The hierarchical diagram in Fig. 8 — if read with full labels on the rectangle nodes — accurately specifies the story of requirement decomposition described in Sect. 4.1. The labels of the rectangle nodes are assertions $S \models R$, where $S$ is a system and $R$ is a requirement, i.e., a condition $S$ must meet. Now each of the four decomposition steps is read as an inference — an *inferential step*

$$(Sub_{i1} \models R_{i1}) \wedge ... \wedge (Sub_{in_i} \models R_{in_i}) \Rightarrow S_i \models R_i \qquad (\mathcal{IS}_i)$$

and the entire diagram can be interpreted as a logical argument/flow built from four pieces. By making all substitutions (both structural and logical), we would obtain the following implication

$$(Sub^{\perp}_1 \models R^{\perp}_1) \wedge ... \wedge (Sub^{\perp}_{N_\perp} \models R^{\perp}_{N_\perp}) \Rightarrow S^{\top} \models R^{\top} \qquad (\mathcal{IS}^{\top})$$

where $Sub^{\perp}_j$, $j = 1...N_\perp$ are atomic subsystems (Sensor1,...,Motor, ..., Fun2 in our case, for which $N_\perp = 8$), $R^{\perp}_j$ are atomic requirement (but each $R^{\perp}_j$ can be a conjunction consisting of truly atomic sub requirements $R^{\perp\perp}_k$ ), and $S^{\top}$ is a huge term resulting from all subsystem substitutions according to the wiring. In our case, $S^{\top}$ = Car and

$$\begin{aligned} S^{\top} &= W_1(S1, ..., \text{Pwt}, ..., \text{ACC}) \\ &= W_1(S1, ...., W_2(M, C, W), ...W_3(\text{ECU1}, \text{ECU2})) \\ &= W_1(S1, ...., W_2(M, C, W), ...W_3(\text{ECU1}, W_4(Fun1, Fun2))) \end{aligned}$$

where $W_i$, $i$ = 1..4 are wiring schemas specified in Fig. 5, and we abbreviate component names to fit the term in one line (S1 is Sensor1, M is Motor etc). In this way, the diagram in Fig. 8 provides an inference *over the design* given by diagram Fig. 5.

Finally, we note that formula ($\mathcal{IS}^{\top}$) is not accurate as there may be several top requirements for the top system: $R^{\top} = R^{\top}_1 \wedge ... \wedge R^{\top}_{N^{\top}}$. Then ($\mathcal{IS}^{\top}$) should be refined by conjunction

$$\mathcal{IS}^{\top}_1 \wedge ... \wedge \mathcal{IS}^{\top}_{N^{\top}} \Rightarrow \mathcal{IS}^{\top} \qquad (\mathcal{IS}^{\top\top})$$

where $\mathcal{IS}^{\top}_i$ is the inferential step

$$Sub^{\perp}_{i1} \models R^{\perp}_{i1} \wedge ... \wedge Sub^{\perp}_{iN_\perp} \models R^{\perp}_{iN^i_\perp} \Rightarrow S^{\top} \models R^{\top}_i \qquad (\mathcal{IS}^{\top}_i)$$

(one step for every $R^{\top}_i$).

*4.3.2 MTBA vs. GSN.* In the current practice of safety assurance, the argument is typically presented via GSN diagrams [29]. The latter look like our diagram Fig. 8 (and indeed, we prepared our diagram with a GSN tool), but semantically different and sometimes essentially different. We use such a loose formulation as there is no accepted semantics for GSN diagrams and different users may (and do) understand them differently (cf. [6]). However, two essential deficiencies of the current practice of using GSN diagrams seem to be rather common: a) system/requirement decomposition and logical inference are intermixed in an unordered way (sometimes merged, sometimes interleaved) while our analysis shows they should go in parallel, b) wiring, and hence dataflow, are implicit: while this can be (arguably) okay for design, it is unacceptable for safety assurance!

# 5 DECOMPOSITION AS MODEL TRANSFORMATION REFINEMENT

Our mostly procedural assurance case in Sect. 3 comprised a relatively simple workflow and rich structural data specified by metamodels with OCL-constraints. In contrast, the inferential assurance case in Sect. 4 is a complex workflow specifiable by ordinary block diagrams over very simple data types. Indeed, while a signal is a complex dynamic entity, its static structure is very simple and is given by its type, e.g., a distance has type $\mathbf{R}^{\boldsymbol{T}}$ – the class of real-valued functions defined on set $\boldsymbol{T}$ understood as the set of time moments, e.g., in many cases, $\boldsymbol{T} = \mathbf{R}$. Thus, data typically managed by a block diagram have a very simple static structure – they are either signals or tuples of signals, e.g., have type $\mathbf{R}^{n\boldsymbol{T}}$, and hence the metamodeling machinery is not necessary. The goal of this section is to outline a specification frameworkin which our main machinery – decomposition – can be done uniformly for both types of assurance models. In other words, we need a general decomposition procedure that smoothly integrates data/structure refinement and behaviour refinement. Conceptually, the MT framework is a good fit – MTs are complex behaviors that operate on rich data specified by metamodels, but important technical details have to be found

and accurately specified. Particularly, we will see that treating a decomposition step as an MT refinement links our subject to the classical area of program refinement.

In the next subsection we will discuss how functional blocks can be specified as MTs, and in Sect.5.3 a unified framework is outlined.

## 5.1 From block diagrams to MTs

We will begin with a brief sketch of metamodeling to fix our notation and terminology. A *metamodel* is a pair $M = (G_M, C_M)$ with $G_M$ a graph of classes and associations (with a typical additional structure offered, say, by UML class diagrams) and $C_M$ a set of constraints (written in, say, OCL). We will often omit the subindex $M$ if it is clear from the context. A *data instance* or a *model* over $M$ is a pair $D = (G_D, t_D)$ with $G_D$ a graph of objects and links, and $t_D$: $G_D \to G_M$ a typing mapping (graph morphism) that gives each element in $G_D$ its type in $G_M$. Model $D$ is called *legal* or *valid*, if the constraints $C_M$ are satisfied. (A general formal specification of constraint satisfaction that works for a very wide class of constraint languages can be found in [16].)

To see how the BD2MT transformation works, let us begin with a simple BD shown in Fig. 5(b2). Each of the wires $q, trq, frc$ is replaced by a respective metamodel consisting of a single class, $M_x = A_x$ with $x \in \{q, trq, frc\}$ (we denote classes by A as letter C is used for constraints). The pair of wires $(d, v)$ gives rise to a metamodel consisting of two classes and we write $M_{d,v} = A_d \otimes A_v$. Now blocks can be seen as MT definitions: Motor: $M_q \to M_{trq}$ or Chassis: $M_{frc} \to M_{d,v}$. For more complex BDs, we can use the procedure we employed in Sect. 4.2. The structure of the term (wiring) reveals that, e.g., ACC block is the following MT definition: ACC: $M^{\mathsf{in}}_{\mathsf{ACC}} \to M^{\mathsf{out}}_{\mathsf{ACC}}$, where the source model is $M^{\mathsf{in}}_{\mathsf{ACC}} = A_d \otimes A_{v_{\mathsf{set}}} \otimes A_{D_{\mathsf{r-t}}} \otimes A_v$ and the target model is $M^{\mathsf{out}}_{\mathsf{ACC}} = A_q$.

What about feedback looping? The latter can be understood as making semantics of a block relational rather than functional: a block with a loop specifies a relation between input and output signals, and semantics of executing this relation as a function is a special story often considered in terms of the least fixed point semantics (LFP) and its derivatives. In the MT world, relational semantics is well-known; a typical representative is QVT-R — an MT language directly based on the relational semantics (although we are not aware of a LFP semantics for QVT-R). Building a reasonable mathematical model for MTs with relational semantics (feedback) is our future work. Finally, an important distinction of MT from BD is that the source and target metamodels can be related by a traceability mapping (overlapped). Its accurate formalization is rather bulky and will appear elsewhere; it is based on the techniques used by the graph-term rewriting community (e.g., [8, 30]).

## 5.2 Inference as MT refinement

Consider an inferential step, e.g., $\mathcal{IS}_3$ in Fig. 8 (grey square 3). We treat block ACC as a transformation ACC: $M^{\mathsf{in}}_{\mathsf{ACC}} \to M^{\mathsf{out}}_{\mathsf{ACC}}$. Previously we considered decomposition as a substitution based on equality:

$$\mathsf{ACC} = W_{\mathsf{ACC}}(\mathsf{ECU1}, \mathsf{ECU2}),$$

but we can also consider it as a refinement

$$\mathsf{ACC} \sqsubseteq \mathsf{ACC}' = W_{\mathsf{ACC}}(\mathsf{ECU1}, \mathsf{ECU2}).$$

The idea is well-known, but to make it work properly, we need a rigorous specification of the relation $\sqsubseteq$: what exactly does refinement mean in our context?

Block ACC considered as a function $F_{\mathsf{ACC}}$, must satisfy certain Assume-Guarantee (AG) conditions: if the input signal satisfies some conditions (i.e., belongs to the domain of the function), then the output signal satisfies its own conditions too (belongs to the codomain). We can phrase this in the MT parlance as follows: if the input data satisfy the constraints $C^{\mathsf{in}}_{\mathsf{ACC}}$ declared in the source/input metamodel $M^{\mathsf{in}}_{\mathsf{ACC}}$ , then the output data satisfy the constraints $C^{\mathsf{out}}_{\mathsf{ACC}}$ in the metamodel $M^{\mathsf{out}}_{\mathsf{ACC}}$. We will write this in a general way for a component $X$

$$D^{\mathsf{in}} \models C^{\mathsf{in}}_X \Rightarrow F^{\mathsf{exe}}_X(D^{\mathsf{in}}_X) \models C^{\mathsf{out}}_X \qquad (\mathrm{AG}_X)$$

where $F^{\mathsf{exe}}_X$ is the procedure of executing the MT definition $F_X$ – this distinction between $F_X$ and $F^{\mathsf{exe}}_X$ is important. (We write $\mathsf{exe}_{F_X}$ in our figures because of the graphical editor limitations, and often skip letter $F$ and identify component $X$ and transformation $F_X$.) We will define refinement as a relation between constructs involved in conditions ($AG_{\mathsf{ACC}}$) and $AG_{\mathsf{ACC}'}$.

Substitution based on equality assumes that both ACC and ACC′ have the same input and output data, but it may be too restrictive for practice. For example, for the general architecture in Fig. 5(a), ACC may consider data $D_{\mathsf{r-t}}$ as a tuple/vector of unspecified dimension while for ACC′ in diagram Fig. 5(b1), block ECU2 may demand the metamodel (data type) to be refined and specify $D_{\mathsf{r-t}}$ as consisting of two tuples of unspecified dimension: one for the road data and the other for the tire data so that $D_{\mathsf{r-t}} = D_r \otimes D_{\mathsf{t}}$. Further down in the refinement chain, block Fun1 (a component of ECU2′), which makes a concrete computation of the braking distance based on data $D_{\mathsf{r-t}}$, demands the two tuples being refined as, say, a pair of road parameters and a pair of tire parameters, $D_r = \mathbf{R} \otimes \mathbf{R}$ and $D_{\mathsf{t}} = \mathbf{R} \otimes \mathbf{R}$.

The story above was presented in signal processing terms. In a structurally refined metamodeling setting, it would begin with a class Road-Tire with attribute rtData of unspecified type Any (i.e., formally, the union of all possible types). For ACC′, the metamodel is refined by making Road-Tire an abstract class partitioned into subclasses Road and Tire with, resp., attributes rData and tData of type Any. For ECU2′, the metamodel is further refined by specializing rData by, say, a pair of attributes: 'slipperyCoefficient' of type **R** and 'roadSurface' of an enumeration type «RoadSurface» (here we deviate from the signal processing version by considering types other than **R**), and similarly for tData. To make the metamodels even more interesting, let us assume a super-safe ACC, whose ECU2 computes the braking distance with a greater precision. Such ACC could require data about the road-tire contact area, which are attributed to the corresponding association (or several of them!) between classes Road and Tire.

Mathematically, the description above can be modeled by mapping $m^{\mathsf{in}}_{XX'}$: $M^{\mathsf{in}}_X \to M^{\mathsf{in}}_{X'}$ between the source metamodels of a component $X$ and its refinement $X'$, which maps an element in $M^{\mathsf{in}}_X$ to either an element in $M^{\mathsf{in}}_{X'}$ or to a query/operation over $M^{\mathsf{in}}_{X'}$, e.g., above we used operations of union (to form an abstract class Road-Tire as the union of classes Road and Tire in $M^{\mathsf{in}}_{X'}$ so that Road-Tire in $M^{\mathsf{in}}_{\mathsf{ACC}}$ is mapped to Road-Tire in $M^{\mathsf{in}}_{\mathsf{ACC}'}$) and tupling (so that attribute rData in $M^{\mathsf{in}}_{\mathsf{ECU2}}$ is mapped to the tuple formed by

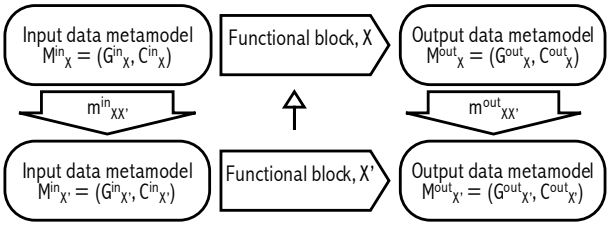


**Figure 9: MT refinement step**

the corresponding attributes in $M^{\mathsf{in}}_{\mathsf{ECU2}'}$ ). Such mappings are well known in algebra under the name of *Kleisli mappings* (see, e.g., their description tailored for the Models community in [9, 11]). A Kleisli mapping between metamodels gives rise to a data restructuring mapping between data instances in the opposite direction: if $D^{\mathsf{in}}_{X'}$ is a data instance (model) over $M^{\mathsf{in}}_{X'}$, it can be restructured to a data instance $D^{\mathsf{in}}_{X'}{\restriction_X}$ over $M^{\mathsf{in}}_X$ by, first, executing the query involved in $m^{\mathsf{in}}_{XX'}$ (e.g., create new objects and links by tupling given objects, or define a new abstract (super)class generalizing several given classes), and then renaming the results according to the mapping. A detailed discussion and more examples can be found in [9, 11].

An important condition such data refinement must satisfy is that for any data instance $D^{\mathsf{in}} \models C^{\mathsf{in}}_X$, there is a refined data instance $D^{\mathsf{in}}_* \models C^{\mathsf{in}}_{X'}$ such that $(D^{\mathsf{in}}_*)^* = D^{\mathsf{in}}$, where the upper star refers to the backward data translation that forgets the extra structure in $M^{\mathsf{in}}_{X'}$. We will refer to this condition as $(\mathsf{Ref}^{\mathsf{in}}_{XX'})$ and call it *precondition weakening*: indeed, it requires constraints for input data in $X'$ to be not too strong. In contrast, on the target side of the transformation story, the requirement flow needs *postcondition strengthening*: if a refined data instance $D^{\mathsf{out}} \models C^{\mathsf{out}}_{X'}$, then we require $(D^{\mathsf{out}})^* \models C^{\mathsf{out}}_X$ Finally, we require commutativity: for any data instance $D^{\mathsf{in}} \models C^{\mathsf{in}}_X$, we require $(F^{\mathsf{exe}}_{X'}(D^{\mathsf{in}}_*))^* = F^{\mathsf{exe}}_X(D^{\mathsf{in}})$. We call diagram in Fig. 9 a correct refinement step, if it satisfies all requirements above.

## 5.3 Summary

Interpreting a decomposition step as a refinement step (Sect. 5.2) rather than an inferential step (Sect. 4.3) includes logical inference (and thus nothing is lost with this transition) but also has an essential advantage. Now the data flow that was implicit in the "pure" logical inference becomes an explicit and important ingredient of the entire model. The latter includes both data refinement given by vertical mappings in Fig. 9, and function/process refinement given by the commutativity of the refinement diagram. This is an accurate mathematical model of the assurance argument flow, and it brings to assurance formal methods and techniques developed in two domains – correctness of program refinement [19, 26] and MT analysis [21, 22].

Thus, each of the four inferential steps in our hierarchy in Fig. 8 can be interpreted as an instantiation of the general refinement pattern in Fig. 9 for $X \in \{\mathsf{Car}, \mathsf{Pwt}, \mathsf{ACC}, \mathsf{ECU2}\}$. We recall that the direction of data refinement goes bottom-up – we restructure refined data towards the pre-refined metamodel. Note that the model we developed is conceptually multi-dimensional: our hierarchy of refinement steps is a hierarchy of arrow squares, in which each arrow (being a block diagram) is actually a planar object. This is a much richer conceptual and structural landscape than offered by basically two dimensional GSN/CAE diagrams.

# 6 APPLICABILITY AND IMPACT

It is premature to provide a definitive answer to the question of how practical MTBA will prove to be in practice. In this section, we envision how practical MTBA could be if it transitions from a mathematical model for ACs to a method, and thus its potential as a basis for developing practical AC methods and tools. Because of the comprehensive approach to the problem of assurance, there is a real opportunity to support AC development with automated tools.

The authors of survey [6] conducted in-depth interviews about the practical use of safety ACs (SACs) with nine experts in software-intensive safety-critical domains, including automotive, railway, avionics and medical devices. Participants' experience in the safety-critical systems ranged from six to 25 years. Seven participants were from industry (both SACs writers and assessors) and two from universities, but with extensive experience in creating industrial-grade SACs. The survey identified seven major challenges in using SACs (introduced below by direct quotes), and we will consecutively discuss how MTBA can help to address each of them.

***1 Scalability.*** *The most significant challenge from our participants' perspective is that of navigating and comprehending large SACs, especially when presented using graphical form.*

MTBA directly addresses the issue by structuring the entire AC as a hierarchy of refinement steps. The latter then appears as an elementary unit for both AC building and assessing. This was the goal of GSN and CAE notations, but they did not fully realize it due to a) the absence of dataflow in AC (given by wiring in MTBA), and b) mixing requirement decomposition and logical inference in an unordered way (sometimes merged, sometimes interleaved) while MTBA shows they should go in parallel.

***2 Managing change.*** *Changes in software, especially in software requirements, can result in changes in SACs.*

Kokaly *et al* have shown in [14, 15] how model management techniques can help to address the issue. MTBA based on MT is directly portable to model management platforms both theoretical and practical. Moreover, traceability—fundamental for change management—is a first class citizen in MTBA as a) the dataflow is explicit and b) the MT model employed is traceability-based [9].

***3 Requiring special skills to create.*** *The graphic notations of SACs are usually easy to understand. However, our participants considered that creating a convincing, well structured safety argument requires special skills and considerable experience.*

It is known that formal reasoning needs a special and extensive mathematical training and is usually difficult for typical engineers (including software engineers). With the current SAC tooling based on, mainly, GSN and CAE, the task becomes even more challenging as they force the user to flatten the multi-dimensional body of the inferential assurance (see Sect. 5) and "dress" it into the tight one-dimensional attire of purely logical inference. MTBA can help by a) a clear separation of concerns into structural and logical, b) explicit modeling of mapping by arrows, and c) bringing to assurance the block diagram (BD) notation, which has "won" several test-of-time awards in different engineering domains (signal processing, control

theory, electrical engineering)—it appears that engineers can build and understand BD much more easily than logical formulas.

***4 Complexity of the system.*** *...Since most safety-critical systems are innately complex and many systems are interconnected, capturing the safety concerns in those systems becomes increasingly challenging. In addition, because developing safety-critical systems often involves experts in various disciplines, multidisciplinary collaboration in arguing system safety is also important and challenging.*

As mentioned in the introduction, we normally manage complexity via decomposition, which is at the heart of MTBA. In addition, the notational basis of MTBA is BDs, which can serve well as a common language for different engineering domains and thus facilitate multidisciplinary communication and collaboration. Another facilitator provided by MTBA is a formal semantics based on well-established mathematical patterns — a discussion and references on how and why it works can be found in [10, Sect.2].

***5 Uncertainty, trust, confidence.*** *Many participants voiced concerns about the fact that system safety always involves issues related to uncertainty, trust, and confidence. Capturing these 'intangible' issues and establishing trust and confidence in the safety arguments was considered as a challenge.*

None of the methodologies can make assurance a fully automatable technological discipline, and the intangible issue above will always be in the way. However, mathematical models for assurance, towards which we have made an important step, can make these intangibles more tangible and better manageable.

***6 Too "flexible".*** *[*As SACs are written by product's manufacturers, the SAC technique*] may be subject to confirmation bias and/or conflicts of interest of the manufacturers.*

This is a general issue inherited in ACs and MTBA can offer nothing radical here. However, because MTBA has a precise semantic basis, we are not randomly searching through "woolly" material, using wishful thinking; we have a solid, well defined structure to examine for such issues as confirmation bias. Participants in the survey would say that MTBA can help to "frame thinking about system safety", and they graded the average perceived importance (IMP) of this factor of using SACs by 3.86 [6, Sect.3] (with the whole being equal 10). We will discuss the three other factors below.

***7 Incomplete information.****Due to the limited integration of SAC management in the software development process, our participants discussed the challenges involved in* gathering sufficient and accurate information *for safety arguments. This issue primarily originated from flawed safety requirements, insufficient test coverage, and incomplete* traceability *across software artifacts.*

MTBA offers three mechanisms to mitigate the above. First, traceability is at the heart of MTBA see discussion on Challenge 2. MTBA also brings a more technological view of assurance than the current approaches as it separates assurance into blocks organized into a high-level workflow, which can further be decomposed. In this way, procedural assurance is managed "inferentially" — the goal originated with the GSN/CAE approaches, but is now based on a proper mathematical basis. Finally, MTBA's focus on dataflow is implicitly a focus on a disciplined and organized gathering of information.

Besides the *Framing of thinking* factor of SAC usage mentioned above, the participants identified three other factors. Two of them, *Communication* with IMP=3.86 and *Easiness to engage with notation* (IMP=3.14) are very well supported by MTBA with its reliance on BDs. The last factor, *Fills the gap for new systems* (IMP=3.71) is a functionality of the AC approach as a whole, and if MTBA serves it well, it serves this factor well too.

## 7 RELATED AND FUTURE WORK

MTBA has several immediate technical sources. A series of seminal papers [14, 18, 20], brought the ideas of metamodeling and model management into assurance, but the main focus of this line of work is on data conformance rather than dataflow. Our MT refinement machinery is based on a) understanding transformations as Kleisli mappings between metamodels [9], and b) Program Refinement – a classical area with an enormous literature (e.g., [26]). Finally, there is the recent work on string diagrams (including signal flow graphs, block diagrams, circuits) in category theory [2, 4, 7]. We are not aware of any prior applications from domains in the later two areas to assurance.

The influence of GSN/CAE methods and tools on assurance practice and research is difficult to overestimate [13, 17, 23, 25]. For the present paper, GSN/CAE were inspirational sources due to their strong emphasis on the decomposition and the hierarchical structure of ACs, which is a cornerstone of MTBA. They have also been influential due to our feeling that GSN/CAE do not fully realize the potential of the decomposition idea and should be fixed. Several ways of formalizing ACs were suggested by Rushby in [24], but his focus is more on the argument structure than on the very basis of assurance.

The following issues are important for future work. Most importantly, the model of assurance we presented is *a posteriori* with respect to the design: we assumed that the functional system design is given and assurance just needs to check it for the assurance property. Clearly, it would be much more effective to do assurance concurrently with design, which is indeed done in practice but is not reflected in our model. The latter is to be extended with a model of functional design and its interaction with assurance. Two other issues are more technical: we need a) an accurate notation for specifying rich workflows over rich data structure, and b) a precise but manageable mathematical framework for specifying and verifying hierarchies of MT refinement.

## 8 CONCLUSION

We proposed a novel architecture and model for assurance cases focused on data and dataflow. The model integrates system decomposition and goal/requirement decomposition into an inferential hierarchy, which is then rearranged into a hierarchy of MT refinement steps. The model brings to assurance established techniques developed in Program Refinement and MT Analysis. We also discussed possibilities of transitioning the model into a method of building reliable assurance cases. Future work will include expansion of the model, as well as developing practical methods for ACs built using the model.

## REFERENCES


[1] Adelard [n. d.]. *Claim, Argument, Evidence Notation.* Adelard. Available at http://www.adelard.com/asce/choosing-asce/cae.html.

[2] J. C. Baez and J. Erbele. 2014. Categories in Control. *ArXiv e-prints* (May 2014). arXiv:math.CT/1405.6881

[3] Robin E. Bloomfield and Peter G. Bishop. 2010. Safety and Assurance Cases: Past, Present and Possible Future - an Adelard Perspective. In *Making Systems Safer - Proceedings of the Eighteenth Safety-Critical Systems Symposium, Bristol, UK, February 9-11, 2010*, Chris Dale and Tom Anderson (Eds.). Springer, 51–67. https://doi.org/10.1007/978-1-84996-086-1_4

[4] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. 2014. A categorical semantics of signal flow graphs. In *International Conference on Concurrency Theory*. Springer, 435–450.

[5] Valentin Cassano, Thomas Maibaum, and Silviya Grigorova. 2016. A (Proto) Logical Basis for the Notion of a Structured Argument in a Safety Case. In *Formal Methods and Software Engineering - 18th International Conference on Formal Engineering Methods, ICFEM 2016, Tokyo, Japan, November 14-18, 2016, Proceedings (Lecture Notes in Computer Science)*, Kazuhiro Ogata, Mark Lawford, and Shaoying Liu (Eds.), Vol. 10009. 1–17. https://doi.org/10.1007/978-3-319-47846-3_1

[6] Jinghui Cheng, Micayla Goodrum, Ronald A. Metoyer, and Jane Cleland-Huang. 2018. How Do Practitioners Perceive Assurance Cases in Safety-Critical Software Systems? *CoRR* abs/1803.08097 (2018). arXiv:1803.08097 http://arxiv.org/abs/1803.08097 there is a better ref to a workshopp CHASE'18.

[7] Bob Coecke and Aleks Kissinger. 2017. *Picturing Quantum Processes. A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press.

[8] Andrea Corradini and Fabio Gadducci. 1999. An Algebraic Presentation of Term Graphs, via GS-Monoidal Categories. *Applied Categorical Structures* 7, 4 (1999), 299–331. https://doi.org/10.1023/A:1008647417502

[9] Zinovy Diskin, Abel Gómez, and Jordi Cabot. 2017. Traceability Mappings as a Fundamental Instrument in Model Transformations. In *Fundamental Approaches to Software Engineering - 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science)*, Marieke Huisman and Julia Rubin (Eds.), Vol. 10202. Springer, 247–263. https://doi.org/10.1007/978-3-662-54494-5_14

[10] Zinovy Diskin, Harald König, Mark Lawford, and Tom Maibaum. 2017. Toward Product Lines of Mathematical Models for Software Model Management. In *Software Technologies: Applications and Foundations - STAF 2017 Collocated Workshops, Marburg, Germany, July 17-21, 2017, Revised Selected Papers (Lecture Notes in Computer Science)*, Martina Seidl and Steffen Zschaler (Eds.), Vol. 10748. Springer, 200–216. https://doi.org/10.1007/978-3-319-74730-9_19

[11] Z. Diskin, T. Maibaum, and K. Czarnecki. 2012. Intermodeling, Queries, and Kleisli Categories. In *FASE (Lecture Notes in Computer Science)*, Juan de Lara and Andrea Zisman (Eds.), Vol. 7212. Springer, 163–177.

[12] André Joyal, Ross Street, and Dominic Verity. 1996. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society* 119, 3 (1996), 447–468. https://doi.org/10.1017/S0305004100074338

[13] Tim Kelly. 1998. *Arguing Safety – A Systematic Approach to Managing Safety Cases*. Ph.D. Dissertation. University of York.

[14] Sahar Kokaly, Rick Salay, Valentin Cassano, Tom Maibaum, and Marsha Chechik. 2016. A model management approach for assurance case reuse due to system evolution. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, October 2-7, 2016*, Benoit Baudry and Benoît Combemale (Eds.). ACM, 196–206. https://doi.org/10.1145/2976767

[15] Sahar Kokaly, Rick Salay, Mehrdad Sabetzadeh, Marsha Chechik, and Tom Maibaum. 2016. Model management for regulatory compliance: a position paper. In *Proceedings of the 8th International Workshop on Modeling in Software Engineering, MiSE@ICSE 2016, Austin, Texas, USA, May 16-17, 2016*. ACM, 74–80. https://doi.org/10.1145/2896982.2896985

[16] Harald König and Zinovy Diskin. 2017. Efficient Consistency Checking of Interrelated Models. In *Modelling Foundations and Applications - 13th European Conference, ECMFA 2017, Held as Part of STAF 2017, Marburg, Germany, July 19-20, 2017, Proceedings (Lecture Notes in Computer Science)*, Anthony Anjorin and Huáscar Espinoza (Eds.), Vol. 10376. Springer, 161–178. https://doi.org/10.1007/978-3-319-61482-3_10

[17] Adelard LLP. 1998. *Adelard Safety Case Development Manual*. Technical Report. http://www.adelard.com/resources/ascad/.

[18] Yaping Luo, Mark van den Brand, Luc Engelen, John M. Favaro, Martijn Klabbers, and Giovanni Sartori. 2013. Extracting Models from ISO 26262 for Reusable Safety Assurance. In *Safe and Secure Software Reuse - 13th International Conference on Software Reuse, ICSR 2013, Pisa, Italy, June 18-20. Proceedings (Lecture Notes in Computer Science)*, John M. Favaro and Maurizio Morisio (Eds.), Vol. 7925. Springer, 192–207. https://doi.org/10.1007/978-3-642-38977-1_13

[19] T. S. E. Maibaum. 1997. Conservative Extensions, Interpretations Between Theories and All That!. In *TAPSOFT'97: Theory and Practice of Software Development*. 40–66.

[20] Sunil Nair, Jose Luis de la Vara, Mehrdad Sabetzadeh, and Lionel C. Briand. 2014. An extended systematic literature review on provision of evidence for safety certification. *Information & Software Technology* 56, 7 (2014), 689–717. https://doi.org/10.1016/j.infsof.2014.03.001

[21] Bentley James Oakes, Javier Troya, Levi Lucio, and Manuel Wimmer. 2015. Fully verifying transformation contracts for declarative ATL. In *18th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MoDELS 2015, Ottawa, ON, Canada, September 30 - October 2, 2015*, Timothy Lethbridge, Jordi Cabot, and Alexander Egyed (Eds.). IEEE Computer Society, 256–265. https://doi.org/10.1109/MODELS.2015.7338256

[22] Lukman Ab. Rahim and Jon Whittle. 2015. A survey of approaches for verifying model transformations. *Software and System Modeling* 14, 2 (2015), 1003–1028.

[23] David J Rinehart, John C Knight, Jonathan Rowanhill, and Dependable Computing. 2015. *Current Practices in Constructing and Evaluating Assurance Cases With Applications to Aviation*.

[24] John Rushby. 2010. Formalism in safety cases. In *Making Systems Safer*. Springer, 3–17.

[25] John Rushby. 2015. Understanding and Evaluating Assurance Cases. SRI-CSL-15-01 (2015).

[26] Steve Schneider. 2001. *The B-Method: An Introduction*. Palgrave Macmillan.

[27] Peter Selinger. 2010. A survey of graphical languages for monoidal categories. In *New structures for physics*. Springer, 289–355.

[28] David I Spivak. 2013. The operad of wiring diagrams: Formalizing a graphical language for databases, recursion, and plug-and-play circuits. *arXiv preprint arXiv:1305.0297* (2013).

[29] The GSN Working Group 2011. *Goal Structuring Notation*. The GSN Working Group. Available at http://www.goalstructuringnotation.info/.

[30] Uwe Wolter, Zinovy Diskin, and Harald König. 2018. Graph Operations and Free Graph Algebras. In *Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig (Lecture Notes in Computer Science)*, Reiko Heckel and Gabriele Taentzer (Eds.), Vol. 10800. Springer, 313–331. https://doi.org/10.1007/978-3-319-75396-6_17