Ensuring Software Quality in Information Systems

Michael Heather & Nick Rossiter
Department of Computer Science and Digital Technologies
Northumbria University, NE1 8ST, UK
michael.heather@trinity.cantab.net; nick.rossiter1@btinternet.com
http://www.nickrossiter.org/process/

Abstract

The work described here identifies facilities within category theory for guaranteeing quality in information systems. For relatively static aspects, the internal structure of the topos is explored further with particular emphasis on the nature of the pasted pullback, including the conditions for a pasting to be valid and the inherent recursive nature of pullback structures. For dynamic aspects, a banking example is explored, leading to the nature of the external processes acting upon the topos such as transactions. These processes are represented by monads, giving a three-level closure on the activity. The nature of monads is explored. The T-algebra enables changes to be made in the monad structure, giving the potential for adaptability. Monads, that have been strengthened by the Kleisli lift to the Cartesian form, can be composed naturally, facilitating the construction of quality large-scale information systems with the reliability required for transactions in the banking world.

Keywords: software quality, transaction, category theory, topos, monad
PACS: 02.10.-v 89.70+c

1 Introduction

A system has a purpose, viewed here as a controlled way for converting source inputs to target outputs. In the simplest case the system is a single automated function converting one input to a unique output. Typically though a system is much more complex, involving many mappings to represent relationships and complex algorithms to represent transformation rules. Relationships may be higher order for mapping one function onto another. The activity may be a mixture of the manual and the automatic. The quality of a system depends on how well its purpose is met. Quality cannot be readily measured and is often delivered with assurances rather than guarantees. Such assurances are stronger if the system has a formal basis, such as a mathematical theory. For example database systems are underpinned by a model employing a standard mathematical structure such as a set, graph or tree. The application of relational query languages such as SQL provides a quality assurance that results are reliable from the application of predicate calculus. Without such an assurance database systems would not be used to the extent they are for handling the information resources of organisations.

As technology advances assurances become less convincing than guarantees. It has been known for some time that some mathematical approaches are more reliable then others. Gödel for instance showed that sets and functions can be only modified and queried reliably in first-order calculus, and not in higher order logic [8]. Further in safe first-order calculus the closed-world assumption is applied, limiting the data considered to those actually occurring rather than to those permitted by the type.

An approach such as category theory represents an advance in technology because of its rigour, employed now in the Blockchain system for bank transaction processing [21]. Category theory is based on the arrow or morphism, capable of representing both the static and the dynamic; arrows may be simple functions or complex relationships between one collection of arrows and another, such as functors and natural transformations. Arrows are defined uniquely up to some level of isomorphism, natural in pure mathematics or metaphysical in the applied world. The fundamental categorical facilities identified for an application include the Topos as a structural data-type and the Monad for process. The application of the monad to a topos gives the operation of a process on data at the highest level, defined as a unique solution up to a metaphysical isomorphism. We will demonstrate such an application, explore how its performance relates to alternative techniques and discuss further work required.

The topos is based on the Cartesian Closed Category (CCC), a category with products, that is closed at the terminal object, and exhibits connectivity through exponentials between all objects. A CCC has an internal logic of the typed λ-calculus, an identity functor and the interchangeability of levels, with nodes being either objects or categories. A topos has additional properties beyond a CCC ([20], at p.106) including a subobject classifier, the internal logic of Heyting, that is intuitionistic logic, and a reflective subtopos for query closure.

The application of the topos to data has been established in recent papers [29, 30]. Structures developed as a topos include pasted pullbacks, to represent complex relationships, and recursive pullbacks, to represent detailed structures of nodes as pullbacks in their own right. The exact nature of the match, in the pasting operation, is still to be decided. The relationship of the topos structures to Fifth Normal Form (5NF) [13], also known as PJNF (Project-Join Normal Form) a challenging ultimate stage in relational database design, is of interest as it indicates the strength of the topos data structuring method. Other less powerful normalisation techniques are considered to be so set-based that any categorial approach would be categorification. The Cocartesian dual to the topos may offer further insights into the data structuring process.

The use of the allegories of Freyd [7] as a basis for data structures was attempted [30] but rejected because of their lack of naturality as set-based relations; the allegories may still have some use though in interoperability as a surrogate for relational databases. Internal queries on a topos are handled by the subobject classifier, which may be Boolean (0 or 1) or the more general powerobject. Both forms were illustrated in the two papers cited above [29, 30]. The provision of examples of Heyting intuitionistic logic for an application remains an objective. Internal queries are more akin to data searches, such as through Google, but do not provide a general process capability.

The application used was of student marks in a university context, which was adequate from the data structure viewpoint but limited from a data process angle. A more interesting application from the process perspective is banking, including the handling of transactions. This was first studied by us in 2006 [28].

Monadic design is a novel technique for handling the dynamic aspects of an application. Aspects to be investigated are the adjointness, inherent in the approach, the flavours of monad which are most suited to process applications and the T-algebra for modifying the adjunction.

The intention in this paper is therefore to introduce a new application, banking, which provides a more suitable test for an external process of a monad on a topos data structure. The mechanism of pasting is to be investigated in detail and the relationship of the topos to database normalisation is to be clarified. Monadic design will be developed for the topos.

2 Pullback: Single Relationship

The pullback category is an example of a topos. Figure 1 shows for the student application, introduced in [29], a simple pullback : $\mathbf{S} \times_{\mathbf{R}} \mathbf{M}$, the product of Student and Mark in the context of Result. The relationship between the product $\mathbf{S} \times_{\mathbf{R}} \mathbf{M}$ and $\mathbf{R}$ is adjoint, with the following logic condition holding: $\exists \dashv \Delta \dashv \forall$. The functor Δ selects pairs of $\mathbf{S}$ and $\mathbf{M}$ in a relationship in the context of $\mathbf{R}$, such that $\exists$ is left adjoint to Δ and $\forall$ is right adjoint to Δ. A diagram with such adjointness was termed by Lawvere as a hyperdoctrine [15].

Other arrows are defined as follows. Projections π are from the product onto its constituents, left π_l and right π_r, with dual arrows left π_l^* and right π_r^* respectively. Inclusions ι are into the sum $\mathbf{S} + \mathbf{M} + \mathbf{R}$ from its constituents, left ι_l and right ι_r, with

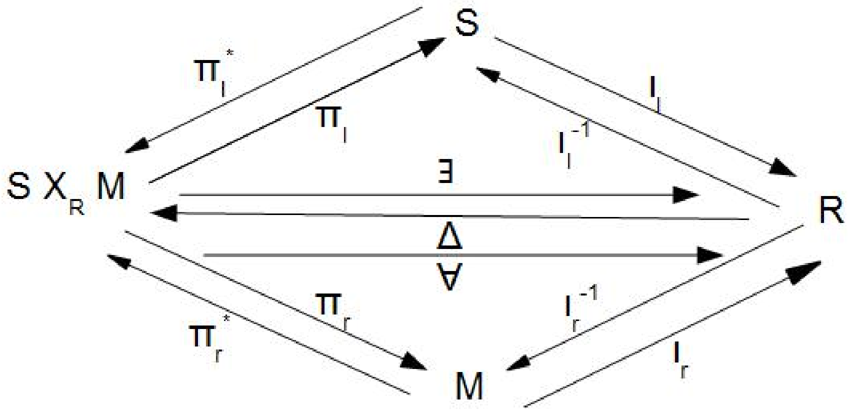

Figure 1: Pullback for a Single Relationship $\mathbf{S} \times_{\mathbf{R}} \mathbf{M}$; **S** Student, **M** marks, **R** Result

dual arrows ι_l^{-1} and ι_r^{-1} respectively.

$\mathbf{S}, \mathbf{M}, \mathbf{R}$ are each categories, with an optional internal pullback structure, giving a recursive pullback structure with potential unlimited depth, as shown in Figure 2. These diagrams show the objects present in each of the categories with the potential for each of the objects to be itself a category, as in a recursive structure. The categories shown at this level, the bottom of the data structure, are of the Dolittle type with a mapping from the data in the left-hand pullback object, a product, to equivalent data in the right-hand pushout object, a coproduct [10]. Such diagrams are also called Bicartesian squares [2] or pulation squares ([1] pp.205-206). The top and bottom objects are apparently the same, being the identifier for the data structure. However, we agree with Lambek & Scott ([14] pp.65-68) that they are different in purpose, with the top object being the intension (definition) and the bottom object the extension (values).

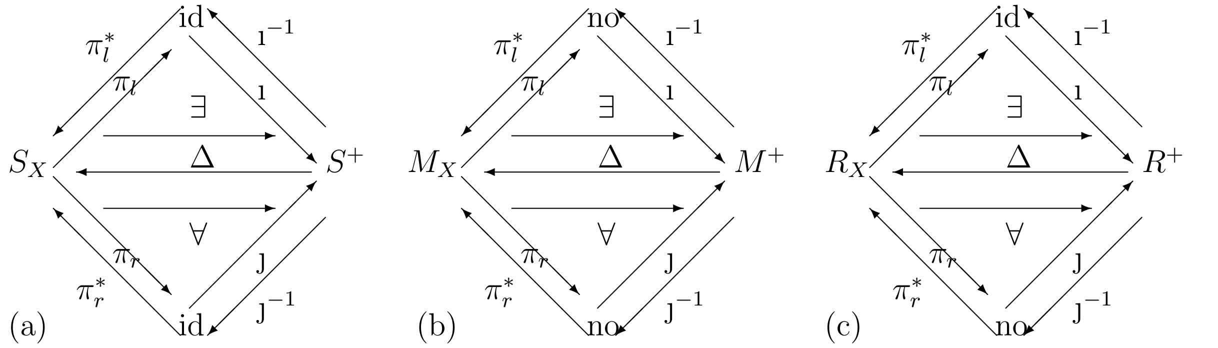

Figure 2: Internal Structure of Categories: a) The Pullback in $\mathbf{S}$. S_X is id $\times_{S+}$ id, S^+ is name $+_{\text{id}}$ address. b) The Pullback in $\mathbf{M}$. M_X is no $\times_{M+}$ no, M^+ is title $+_{\text{no}}$ grade, c) The Pullback in $\mathbf{R}$. R_X is id $\times_{R+}$ no, R^+ is mark $+_{\text{id}\times\text{no}}$ decision.

3 Banking Examples

3.1 Pullback: Single Relationship

We now introduce the Banking example, which is a more suitable subject for illustrating the action of process on a topos. The simple pullback is shown in Figure 3, defined as $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$, that is the product of Procedure and Account in the context of Transaction, with $\mathbf{P}$ the category Procedure, $\mathbf{A}$ the category Account, and $\mathbf{T}$ the category Transactions. An Account can belong to many users; the Procedure is the type of the transaction, for example: standing order, direct debit, ATM cash withdrawal; the transaction is a transfer of funds according to data processing requirements. $\mathbf{P}, \mathbf{A}, \mathbf{T}$ are categories, with internal pullback structure, giving recursive pullbacks as required, as in Figure 2 for the student example.

3.2 Pullback: Two Pasted Relationships

In pasted pullbacks two relations are joined together to form a square. Additional categories are introduced for User (customer) of $\mathbf{U}$. Each user may have multiple accounts across the banking network: there is a many-to-many (N:M) relationship between $\mathbf{U}$ and $\mathbf{A}$. The second pullback is the product of the subproduct of the first pullback $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$ with $\mathbf{U}$ in the context of $\mathbf{A}$, as shown in Figure 4. The resulting relationship is of account transactions by users. For the purpose of discussion, the pullbacks can be labelled $Pb1$ for the first square $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$ and $Pb2$ for the second square $(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U})$. By standard category theory ([20] pp.71-72) if the squares $Pb1$ and $Pb2$ are valid pullbacks, then the whole outer square is also a pullback $Pb2 \times Pb1$. We therefore have three pullback

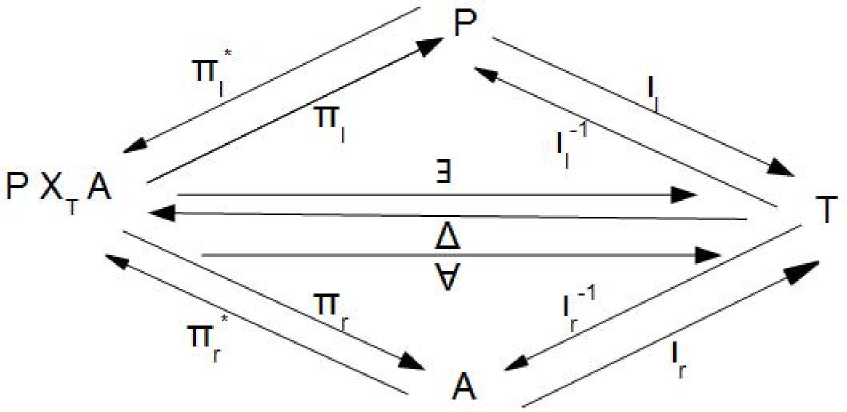

Figure 3: Pullback - Single Relationship: Bank Transactions by Procedure and Account

diagrams in a valid pasted relationship.

The vertical stacking of the pasted pullbacks, one above the other, in portrait form is suited to practical applications which could involve 5-10 relationships in a deep nested structure. In category theory text books, pasted structures are usually written in horizontal (landscape) form as in Figure 5, which is logically identical to that in Figure 4.

The aim of pasting in topology is to 'glue together' two continuous functions to create another continuous function. The specific pasting condition for the pullback $Pb2 \times Pb1$ is that $\iota_l' = \pi_r$ after Freyd's Pasting Lemma [7].

To make the application more realistic we add two further categories, those of **B** for Branch and **C** for (banking) Company. Branch:User is also a N:M relationship as each Branch has many Users and each User has many Branches but Company:Branch is a 1:N relationship: each Company has many Branches, each Branch is within one Company. The overall relationship is $(((\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U}) \times_{\mathbf{U}} \mathbf{B})$ with **C** in the context of **B** giving the pullback diagram shown in Figure 6. The representation of N:M and 1:N relationships is the same in terms of pullback structures, giving a useful symmetry in data design.

Figure 6 involves six categories: **C** company, **B** branch, **U** user, **A** account, **P** procedure, **T** transaction, and ten pullbacks: $Pb4, Pb3, Pb2, Pb1$; $Pb4 \times Pb3, Pb3 \times Pb2, Pb2 \times Pb1$; $Pb4 \times Pb3 \times Pb2, Pb3 \times Pb2 \times Pb1$, $Pb4 \times Pb3 \times Pb2 \times Pb1$. The relations within

$$\begin{tikzcd}
(P\ X_T\ A)\ X_A\ U \arrow[r, "\pi'_r"] \arrow[d, "\pi'_l"'] \arrow[dr, phantom, "\text{Pb2}"] & U \arrow[d, "l'_r"] \\
(P\ X_T\ A) \arrow[r, "l'_l", "\pi_r"'] \arrow[d, "\pi_l"'] \arrow[dr, phantom, "\text{Pb1}"] & A \arrow[d, "l_r"] \\
P \arrow[r, "l_l"'] & T
\end{tikzcd}$$

Figure 4: Pullback: Two Pasted Relationships: Bank Transactions by User, in Portrait Layout

$$\begin{tikzcd}
(P\ X_T\ A)\ X_A\ U \arrow[r, "\pi'_l"'] \arrow[d, "\pi'_r"] \arrow[dr, phantom, "\text{Pb2}"] & (P\ X_T\ A) \arrow[r, "\pi_l"] \arrow[d, "l'_l"', "\pi_r"] \arrow[dr, phantom, "\text{Pb1}"] & P \arrow[d, "l_l"] \\
U \arrow[r, "l'_r"'] & A \arrow[r, "l_r"'] & T
\end{tikzcd}$$

Figure 5: Pullback: Two Pasted Relationships: Bank Transactions by User, in Conventional Landscape Layout

a banking system are shown in more conventional form in Figure 7(a) where each single-headed arrow represents a 1:N (one-to-many) relationship and each double-headed arrow represents a N:M (many-to-many) relationship.

For our purposes, a pasted pullback is only a valid pullback if all inner and outer diagrams are pullbacks. There are some theorems in pure category theory ([20] pp.71-72) which enable some deductions to be made based on partial knowledge: for example, with the diagram in Figure 5, if the inner diagrams are pullbacks then the outer diagram is a pullback, as stated earlier, and if the outer diagram and the right-hand diagram are pullbacks then the left-hand diagram is a pullback. Such deductions could be facilitated in any practical system but are a distraction from developing a simple robust solution.

As an example of an invalid pullback, consider the diagram in Figure 8 where the relationship diagram has been modified to that in Figure 7(b). There are seven valid

$$\begin{tikzcd}
(((P \times_T A) \times_A U) \times_U B) \times_B C \arrow[r, "l_l''''"] \arrow[d, "\pi_l'''"'] \arrow[dr, phantom, "Pb4"] & C \arrow[d, "l_r'''"] \\
((P \times_T A) \times_A U) \times_U B \arrow[r, "l_l'''", "\pi_r''"'] \arrow[d, "\pi_l''"'] \arrow[dr, phantom, "Pb3"] & B \arrow[d, "l_r''"] \\
(P \times_T A) \times_A U \arrow[r, "l_l''", "\pi_r'"'] \arrow[d, "\pi_l'"'] \arrow[dr, phantom, "Pb2"] & U \arrow[d, "l_r'"] \\
(P \times_T A) \arrow[r, "l_l'", "\pi_r"'] \arrow[d, "\pi_l"'] \arrow[dr, phantom, "Pb1"] & A \arrow[d, "l_r"] \\
P \arrow[r, "l_l"'] & T
\end{tikzcd}$$

Figure 6: Pullback: Three Pasted Relationships: Bank Transactions by User by Company Branch

pullbacks in the diagram: $Pb4, Pb3, Pb2, Pb1$; $Pb3 \times Pb2, Pb2 \times Pb1$; $Pb3 \times Pb2 \times Pb1$, but not all squares are pullbacks, for example $Pb4 \times Pb2$. Therefore the whole diagram is not a valid pullback.

For any valid pullback, the logic of adjointness holds for the outer square and all inner squares. Therefore for Figure 6 with its six valid pullback diagrams, the logic $\exists \dashv \Delta \dashv \forall$ holds across every diagram. An example of this logic is shown in Figure 9 for the outer square.

3.3 Subobject Classifier

The pasted structure is a Cartesian Closed Category (CCC) with products, terminal object and exponentials. Further it is a topos as a CCC with subobject classifier and internal Heyting Logic. The subobject classifier provides an internal query language for which a Boolean example is shown in Figure 10.

The subobject classifier facilitates simple database or information retrieval queries:

- $\Omega\{0, 1\}$ is the subobject classifier with subobjects classified as either 0 or 1

- χ_j is the characteristic function, a query mapping from the object S to $\{0, 1\}$, false or true

- 1_{topos} is the terminal object of the topos, giving a handle on the topos

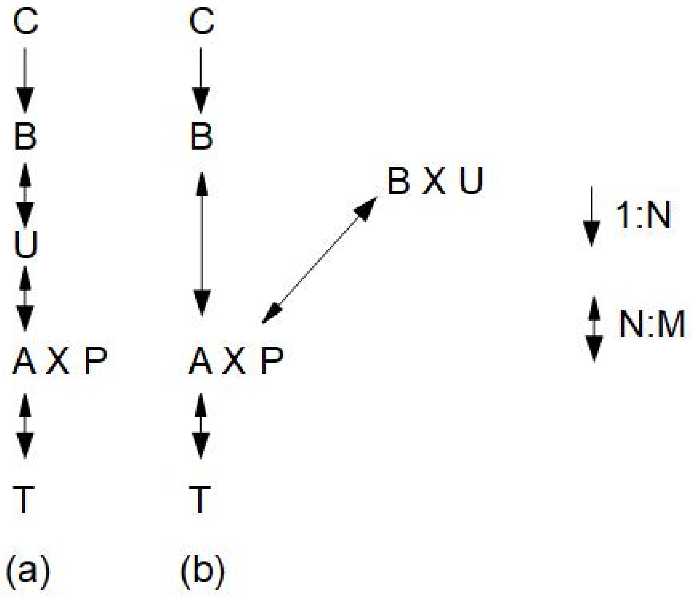

Figure 7: Relations within a Banking System corresponding to (a) Figure 6 and (b) to Figure 8. C is Company, B branch, U user, A account, P procedure, T transaction.

- $\jmath$ is the mapping from the subtopos U, the result of the query, to the object S
- U is the identity of the subtopos, giving query closure

The diagram may be viewed as a pullback of *true* along χ_j, with U as $1_{topos} \times_{\Omega\{0,1\}} S$.

4 Pasting Pullbacks: Discussion

To summarise, in a pasted diagram, all pullbacks as inner or outer squares must commute for the diagram to be a valid pullback as a whole. The structure is recursive in that a pullback node may itself be a pullback diagram. Two aspects are worthy of further discussion: how does the pullback diagram relate to data normalisation in conventional data structuring and can the pasting condition be expressed in other forms, drawing out the nature of the ’=’ condition?

$$\begin{tikzcd}
((P\, X_T\, A)\, X_A\, B)\, X_B\, C \arrow[r, "\pi_r''"'] \arrow[d, "\pi_l''"'] \arrow[dr, phantom, "Pb3"] & C \arrow[d, "\iota_r''"] & \\
(P\, X_T\, A)\, X_A\, B \arrow[r, "\iota_l''", "\pi_r'"'] \arrow[d, "\pi_l'"'] \arrow[dr, phantom, "Pb2"] & B \arrow[d, "\iota_r'"', "\iota_l'''"] & (B\, X_A\, U) \arrow[l, "\pi_l'''"'] \arrow[d, "\pi_r'''"] \arrow[dl, phantom, "Pb4"] \\
(P\, X_T\, A) \arrow[r, "\iota_l'", "\pi_r"'] \arrow[d, "\pi_l"'] \arrow[dr, phantom, "Pb1"] & A \arrow[d, "\iota_r"] & U \arrow[l, "\iota_r'''"] \\
P \arrow[r, "\iota_l"'] & T &
\end{tikzcd}$$

Figure 8: Invalid Pullback Diagram, corresponding to Relations in Figure 7(b)

4.1 Normalisation

Normalisation is the standard technique for evaluating a data design, in particular to determine how closely the logical design matches the physical world. A number of stages have been developed for the set-theoretic relational model: 1NF(First Normal Form), 2NF, 3NF, BCNF, 4NF, 5NF. The last and most demanding stage 5NF concerns us here, not just for its rigour but for its definition in category theory terms, indicated by its alternative name of Project-Join Normal Form (PJNF).

In set theoretic terms, the definition of 5NF is that the structures resulting from the projections can be joined together to return the original structure without loss or gain of information [13]. Looking at the simple pullback diagram, as in Figure 3, the projections are the π arrows, π_l and π_r, and the join arrow is the diagonal Δ. PJNF holds through the adjointness in every pullback: $\exists \dashv \Delta \dashv \forall$. The arrows $\exists$ and $\forall$ involve the projections through the compositions: $\exists = \iota \circ \pi$ and $\forall = \iota \circ \pi$. In more complex data structures, the same logic applies. For instance in Figure 9 with six pullback squares (including undrawn inner ones), PJNF will hold if the whole structure and all inner squares are pullbacks with the logic: $\exists \dashv \Delta \dashv \forall$. Surprisingly pullbacks have rarely been used in normalisation studies, an exception being the work of Levene & Vincent [18] who briefly mention the pullback inference rule, following from the interaction between functional dependencies $\exists$ and inclusion dependencies ι.

It should be emphasised that the pullback is not categorification of the set-theoretic approach to normalisation of 5NF, as in earlier work with category theory and databases [12]. The form 5NF was a belated move by set-theoretic adherents to find a viable approach to normalisation after many earlier attempts had been only partially successful. The pullback follows basic category theory principles and is a *natural* choice for an effective

$$\begin{tikzcd}
(((\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U}) \times_{\mathbf{U}} \mathbf{B}) \times_{\mathbf{B}} \mathbf{C} \arrow[r, "\iota_l''''"] \arrow[d, "\pi_l'''"'] \arrow[dddr, "\exists"] \arrow[ddddr, "\forall"'] & \mathbf{C} \arrow[d, "\iota_r'''"] \\
((\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U}) \times_{\mathbf{U}} \mathbf{B} \arrow[d, "\pi_l''"'] & \mathbf{B} \arrow[d, "\iota_r''"] \\
(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U} \arrow[d, "\pi_l'"'] & \mathbf{U} \arrow[d, "\iota_r'"] \\
(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \arrow[d, "\pi_l"'] & \mathbf{A} \arrow[d, "\iota_r"] \arrow[uuul, "\Delta"] \\
\mathbf{P} \arrow[r, "\iota_l"'] & \mathbf{T}
\end{tikzcd}$$

Figure 9: Adjointness Holds for all Pullbacks: $\exists \dashv \Delta \dashv \forall$

data structure.

4.2 The Pasting Condition

The Pasting Condition is $\iota_l' = \pi_r$, that is the left-inclusion of the outer square equals the right-projection of the inner square. On the surface this looks rather set theoretic, where the '=' would be without context, but in category theory the '=' is defined naturally as unique up to natural isomorphism, through the adjointness inherent in the pullback category.

Moreover any pullback can be represented as an equalizer [26], as in Figure 11, which is equivalent to Figure 3. In the equalizer diagram the product of **P** and **A** in the context of **T**, $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$, maps onto the product $\mathbf{P} \times \mathbf{A}$ which in turn maps onto **T** where the two paths, $\iota_l \circ \pi_l$ and $\iota_r \circ \pi_r$, converge.

Equalizer diagrams can also be constructed for pasted pullbacks, as in Figure 12, which is equivalent to Figure 4. In the equalizer diagram the product of $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$ and **U** in the context of **A**, $(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U}$, maps onto the product $(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times \mathbf{U}$ which in turn maps onto **A** where the two paths, $\iota_l \circ \pi_l \circ \pi_l'$ and $\iota_r \circ \iota_r' \circ \pi_r'$, converge.

$$\begin{tikzcd}
U \arrow[r] \arrow[d, "\jmath"'] & \mathbf{1_{topos}} \arrow[d, "\text{true}"] \\
S \arrow[r, "\chi_j"'] & \Omega\{0,1\}
\end{tikzcd}$$

Figure 10: Pullback Square for Boolean Subobject Classifier: Definition of Characteristic Function $\chi_j : S \longrightarrow \Omega\{0,1\}$

$$\begin{tikzcd}
\mathbf{P} \times_{\mathbf{T}} \mathbf{A} \arrow[r] & \mathbf{P} \times \mathbf{A} \arrow[r, shift left, "\iota_l \circ \pi_l"] \arrow[r, shift right, "\iota_r \circ \pi_r"'] & \mathbf{T}
\end{tikzcd}$$

Figure 11: Pullback in Figure 3 Represented as an Equalizer

5 External Process

The concept of process is underpinned by metaphysics, as defined in the writing of authors such as Alfred North Whitehead, in his book Process and Reality [31]. For any entity in the universe, the actions possible upon it and the rules for such actions are a critical part of the whole system. First we look at the technical features within category theory for representing process. We next look at the requirements for the real world and review the facilities of the theory that appear to be most relevant.

5.1 Process in Category Theory

An internal process is a morphism (arrow) within a topos, such as $p : A \longrightarrow B$, where the process p takes object A to object B in the same topos. Such arrows play a natural role in the category construction. An external process is activity on a topos $\mathbf{E}$, taking it to another topos $\mathbf{E}'$, such as provided by a functor F with $F : \mathbf{E} \longrightarrow \mathbf{E}'$. Both $\mathbf{E}, \mathbf{E}'$ must conform to the natural rules for topos construction. Constraints on the transition between $\mathbf{E}$ and $\mathbf{E}'$ are enforced through adjointness between F ($\mathbf{E} \longrightarrow \mathbf{E}'$) and its dual G ($\mathbf{E}' \longrightarrow \mathbf{E}$), such that $F \dashv G$ and the 4-tuple $< F, G, \eta, \epsilon >$ exists where η is the unit of adjunction $\eta : 1_E \longrightarrow GFE$, ϵ is the counit of adjunction $\epsilon : FGE' \longrightarrow 1_{E'}$ and E, E'

$$\begin{tikzcd}
(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U} \arrow[r] & (\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times \mathbf{U} \arrow[r, shift left, "\iota_l \circ \pi_l \circ \pi_l'"] \arrow[r, shift right, "\iota_r \circ \iota_r' \circ \pi_r'"'] & \mathbf{A}
\end{tikzcd}$$

Figure 12: Pasted Pullback in Figure 4 Represented as an Equalizer

are objects in categories $\mathbf{E}$ and $\mathbf{E}'$ respectively. The pair of adjoint functors FG may be written as T and the dual GF as S.

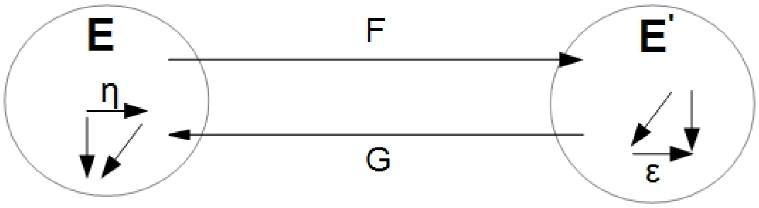

Figure 13: Multiple 'Cycles' $FGFGFG(T^3)$ for adjointness $< F, G, \eta, \epsilon >$

The adjointness T can be enhanced by performing it three times, T^3, to achieve closure. Such a construction is termed a monad, with its dual S^3 termed a comonad. The functors and their constraints are illustrated in Figure 13. The monad is a generalisation of the single-level monoid, which has a single operation, binary multiplication $M \times M \longrightarrow M$, and the identity $1 \longrightarrow M$, for an object M.

5.2 Real-world Requirements

The process is represented in information systems by the transaction, which has been the subject of intense study because of its criticality to applications such as banking and internet-based commerce. However, the concept is a very general one, applying for instance to drafting where a transaction may last several days as a technical drawing is modified from one consistent state to another, or maybe months, as a legal document is modified similarly. The notion of transaction in a categorial context was developed in an earlier paper in 2011 [11], and in considerable detail in 2006 [28]. The principles of the transaction are summarised as ACID: Atomicity, Consistency, Isolation, Durability.

Atomicity ensures that the process, however complicated, is viewed as a single arrow. Consistency ensures that all rules have to be satisfied before the transition is made. Isolation ensures that any intermediate results in the process are not revealed. Durability ensures that once a transaction is performed, the results persist until changed by another transaction. The transaction is a logical technique for controlling the real world.

5.3 Applicability of the Three Cycles

A transaction is viewed naturally as three 'cycles' of adjointness [28]. The first cycle performs the actual work required; the second checks for any errors or inconsistencies resulting from the first cycle; the third cycle consolidates the changes made provisionally in the first cycle and checked in the second cycle. The 'cycles' are not separate stages; all three cycles are performed as a single snap: the prehension, or grasping, of Whitehead [31]. This single snap satisfies the atomicity and isolation requirements. The second cycle satisfies the consistency requirement, through review against the rules. The third cycle satisfies the durability requirement, through consolidating the results. If adjointness does not hold in any cycle, the transaction is abandoned. We now look at the application of the monad in more detail.

$$\begin{tikzcd}
T^3 \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
IT \arrow[r, "\eta T"] \arrow[d, "="'] & T^2 \arrow[d, "\mu"] & TI \arrow[l, "T\eta"'] \arrow[d, "="] \\
T \arrow[r, "="'] & T & T \arrow[l, "="]
\end{tikzcd}$$

(b)

Figure 14: (a) Associative Law for Monad $< T, \eta, \mu >$; (b) Left and Right Unitary Laws for Monad $< T, \eta, \mu >$

5.4 Technical Details of the Monad Approach

For a monad, the diagrams for the associative laws and unitary laws are shown in Figure 14. These diagrams provide the formal basis for the approach. Figure 14(a) shows the relationship between T^3, T^2 and T where T is the endofunctor $GF : \mathbf{X} \longrightarrow \mathbf{X}$, $\mathbf{X}$ being

$$\begin{tikzcd}
S \arrow[r, "\delta"] \arrow[d, "\delta"'] & S^2 \arrow[d, "S\delta"] \\
S^2 \arrow[r, "\delta S"'] & S^3
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
S \arrow[d, "="'] & S \arrow[l, "="'] \arrow[r, "="] \arrow[d, "\delta"] & S \arrow[d, "="] \\
IS & S^2 \arrow[l, "\epsilon S"] \arrow[r, "S\epsilon"'] & SI
\end{tikzcd}$$

(b)

Figure 15: (a) Associative Law for Comonad $< S, \epsilon, \delta >$ (b) Left and Right Unitary Laws for Comonad $< S, \epsilon, \delta >$

any category. An endofunctor is a functor with the same source and target. A pair of adjoint functors F and G is an endofunctor as the source of $F : \mathbf{X} \longrightarrow \mathbf{Y}$ is X and the target of $G : \mathbf{Y} \longrightarrow \mathbf{X}$ is also $\mathbf{X}$. The unit or identity of the monad is $\eta : 1 \longrightarrow T$ from Figure 14(b) and the multiplication of the monad is $\mu : T^2 \longrightarrow T$ from Figure 14(a). We therefore write the monad T as the object $< T, \eta, \mu >$, with the category $\mathbf{X}$, on which the monad is based, omitted as it is inferred from the functors involved. However, it is not wrong to write the monad as the object $< \mathbf{X}, T, \eta, \mu >$ where the nature of $\mathbf{X}$ has a bearing on the arguments being made. Further it is often useful to say on which category the monad is based.

For a comonad, the dual of the monad, the diagrams for the associative laws and unitary laws are shown in Figure 15. Figure 15(a) shows the relationship between S, S^2 and S^3 where S is the endofunctor $FG : \mathbf{Y} \longrightarrow \mathbf{Y}$, $\mathbf{Y}$ being any category. The counit or identity of the comonad is $\epsilon : S \longrightarrow 1$ from Figure 15(a) and the comultiplication of the comonad is $\delta : S \longrightarrow S^2$ from Figure 15(b). We therefore write the comonad S as the object $< S, \epsilon, \delta >$ or $< \mathbf{Y}, S, \epsilon, \delta >$.

Figure 16 shows the two triangular identities for the monad in the category $\mathbf{X}$, derived by applying the interchange law to Figure 14(b). Through commutativity Figure 16 defines the arrow $G\epsilon F : GFGF \longrightarrow GF$. This arrow is the multiplication of Figure 14, that is $\mu : T^2 \longrightarrow T$. Therefore we can rewrite the monad $< T, \eta, \mu >$ as $< T, \eta, G\epsilon F >$ for an alternative view, based on the units and counits of adjunction η and ϵ respectively.

$$\begin{tikzcd}
I_X GF \arrow[r, "\eta GF"] \arrow[dr, "="] & GFGF \arrow[d, "G\epsilon F"] & GFI_X \arrow[l, "GF\eta"'] \arrow[dl, "="'] \\
& GF &
\end{tikzcd}$$

Figure 16: The Monad in the category **X**: Triangular Identities defining ϵ

5.5 Historical and Present Usage of the Monad Term

According to Hippolytus (170−235 AD), the worldview was inspired by the Pythagoreans, who called the first thing that came into existence the *monad*, from which came the dyad, triad, tetrad, etc. [4]. Gnosticism is a modern term for a multitude of Jewish religious ideas and systems from the first and second century AD, with the highest God, Supreme Being or the One, termed the Monad. The Syrian-Egyptian school depicts creation as coming from a primal monadic source, finally resulting in the creation of the material universe.

The monad entered metaphysics as the *Monadology* of Leibniz, written from 1712-1714 as *Principes de la nature et de la grâce fondé en raison*, which has since been published in various forms and languages [17]. Leibniz allows just one type of element in the building of the universe, which is given the name monad or entelechy, and described as a simple substance, which has no parts, hence indivisible. Monads are elementary particles with blurred perceptions of one another and have been described as eternal, indecomposable, individual, subject to their own laws, un-interacting, and each reflecting the entire universe in a pre-established harmony; monads are centres of force; substance is force, while space, matter, and motion are merely phenomenal. Like atoms, monads are irreducible but differ in their complete mutual independence, and in their following of a preprogrammed set of instructions peculiar to itself, so that a monad ‘knows’ what to do at each moment. Each monad is like a little mirror of the universe.

The monad term is also used in music, where it is a single note, with a dyad being 2 notes, a triad 3 notes, etc., and in biology where it is a unicellular organism.

In functional programming, the monad is an increasingly popular construction as an abstract data type, with promising developments in the language Haskell [9, 19], named after Haskell B Curry, who developed the transformation of functions through currying in the $\lambda-$calculus. The monad in Haskell is formally classified as an extension of the monad developed in category theory, involving the notion of a strong monad [23, 25]. Such a monad is defined in higher-order category theory as a bicategory construction. In more concrete terms a strong monad is defined as a (categorial) monad with strengthening with respect to products and idempotency. The strengthening with products leads to the concept of a Cartesian monad, where if the underlying categories are pullbacks, the monad T preserves pullbacks, and μ and η are Cartesian, then the monad is Cartesian. Such a construction facilitates the use of T in transformations where a Cartesian type is expected. The strengthening with idempotency provides resilience as further operations are performed. So with the underlying category for the monad $\mathbf{X}$ being Cartesian with the object $A \times B$, there is a natural transformation $\tau_{A,B}$ from the Cartesian operation $(A \times TB)$ to $T(A \times B)$ such that strengthening with the identity I is immaterial, consecutive applications of strength commute, and strength commutes with monad unit and multiplication [24]. Further details of the Cartesian monad are found later in this paper in Section 7, in the work by Mulry [25] and in Appendix C of Leinster's book *Higher Operads, Higher Categories* [16].

Category theory is regarded as a unifying force so might be able to provide an insight into all of the above notions of the monad. The notion of unit applies to all the various usages and this is continued into the categorial version with the unit in the monad definition $< T, \eta, \mu >$ of $\eta : 1 \longrightarrow T$ and the counit in the comonad definition $< S, \epsilon, \delta >$ of $\epsilon : S \longrightarrow 1$. The monad of Leibniz is similar to the categorial version in respect of their following a preprogrammed set of instructions with each monad being a little mirror of the universe. However, there is a major difference – Leibniz's monad is a particle and the categorial monad is a process – emphasising the set-based nature of Leibniz's work. The use of the term monad in music appears to reflect the physical reality of a single note. From a more constructive point of view, musical units, and hence monads, might also include chords and other logical combinations of notes. An application of the categorial monad to music is under active consideration. The use of the term monad for a unicellular organism has lapsed, maybe because the general term was confusable with its use for specific unicellular organisms, the *Monas*. The comparison between the monad of functional programming and that in category theory is the most useful: this shows that the Cartesian monad selected for functional programming is indeed the type of monad needed for information systems as the underlying Haskell category has products, in particular pullbacks, which form the basis of our structural approach.

6 Process on a Topos

The monad and comonad processes are applied to a topos, defining the structure of the data, to perform the transactions. The design of the processes ia therefore termed Monadic Design. We write the process on a topos as:

$$T : \mathbf{E} \longrightarrow \mathbf{E}'$$

where T is the Cartesian monad $< T, \eta, \mu >$ for a category $\mathbf{E}$ with endofunctor T, that is $GF : \mathbf{E} \longrightarrow \mathbf{E}$, unit of adjunction $\eta : 1 \longrightarrow T$ and unit of multiplication $\mu : T^2 \longrightarrow T$. The source topos is $\mathbf{E}$ and the target topos is $\mathbf{E}'$, with the topos based on pullbacks, including the pasted variety, as described in Section 3. The type (intension) of the source and target is the same but the data values (extension) will vary. Closure is achieved as the type is preserved.

For the running bank example, the Cartesian monad T is the banking system transaction, the source information system is $\mathbf{E}$ and the target information system is $\mathbf{E}'$. There may be more than one adjunction for a monad T, based on a category $\mathbf{E}$. For instance $< F, G, \eta, \epsilon >$ may be one adjunction for $\mathbf{E} \longrightarrow \mathbf{E}'$ with another of $< F_A, G_A, \eta_A, \epsilon_A >$ for $\mathbf{A} \longrightarrow \mathbf{E}$, where $\mathbf{A}$ is a subcategory of $\mathbf{E}$. So a variety of adjunctions may be handled by a single monad, over various subcategories of a particular category. This gives flexibility in handling different data-sets with the same underlying structure.

For the process there will also be a comonad:

$$S : \mathbf{E}' \longrightarrow \mathbf{E}$$

where S is the Cartesian comonad $< S, \epsilon, \delta >$ for a category $\mathbf{E}'$ with endofunctor S, that is $FG : \mathbf{E}' \longrightarrow \mathbf{E}'$, counit of adjunction $\epsilon : S \longrightarrow 1$, counit of multiplication $\delta : S \longrightarrow S^2$.

Categories of algebras can be defined over the monad and comonad. From the algebraic perspective, there are two approaches employing the monad/comonad as the underlying categories. The category of algebras over a monad is traditionally called its Eilenberg-Moore category [6] ([20] at pp. 139-142). Dually, the Eilenberg-Moore category of a comonad is its category of coalgebras. The subcategory of free algebras is traditionally called the Kleisli category of the monad, as is its dual the subcategory of co-free co-algebras of the comonad ([20] at pp. 147-148). The Kleisli category of a monad transforms a monad into a form more suitable for implementation in a functional language such as Haskell. Compared to the pure mathematics form of Mac Lane, Kleisli strength generalises the notion of commutativity and guarantees that products lift to the corresponding Kleisli categories [25]. From the point of view of products, the monads developed to Kleisli strength are applicable in a much wider range of computing applications. Kleisli categories are discussed in more detail in Section 7.

6.1 The T-algebra

The T-algebras are one of the algebraic forms resulting from the work of Eilenberg and Moore [6]. Such algebras facilitate changing the definition of a monad and therefore permitting fundamental changes to the operand of our process. For any category **X**, which in our case is a topos **E**, the T-algebra produces a new consistent state of adjunction for a modified intension.

In more detail, applying the T-algebra to a topos **E**, in the monad with adjunction $< GF, \eta, \mu >$, yields a new monad adjunction $< G^T F^T, \eta^T, G^T \epsilon^T F^T >: \mathbf{E} \longrightarrow \mathbf{E^T}$; that is a new monad adjunction $F^T \dashv G^T$ is defined to accommodate the changed category $\mathbf{E^T}$. A T-algebra is $< e, h >$ where e is an object in **E**. The structure map of the algebra is $h : Te \longrightarrow e$ such that the diagrams in Figure 17 commute. Beck's Theorem provides rules on which categorial transformations in the T-algebra $X \longrightarrow X^T$ are valid [3]. This is sometimes called PTT (Precise Tripleability Theorem).

$$\begin{tikzcd}
T^2e \arrow[r, "Th"] \arrow[d, "\mu_e"'] & Te \arrow[d, "h"] \\
Te \arrow[r, "h"'] & e
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
e \arrow[r, "\eta_e"] \arrow[dr, "1"] & Te \arrow[d, "h"] \\
 & e
\end{tikzcd}$$

(b)

Figure 17: T-algebra: (a) Associative Laws, (b) Unitary Laws

7 Application

The categorial monadic approach is being used for the Blockchain [21], a transaction system, adopted by Bitcoin, for keeping hundreds or even thousands of copies of each transaction record, using multiple transaction logs. The monadic design pattern provides a broad range of transactional semantics with composition the key to scaling any system. The blockchain approach is drawing interest from the established banking industry, where a blockchain is viewed as a shared, encrypted 'ledger' that cannot be manipulated, offering promise for secure transactions [27]. Meredith indicates that compositionality is the key to reliability but offers few details on how this is achieved in the monad. Compositionality is a cornerstone of category theory, defined as a minimum up to some level of isomorphism. In monad/comonad definitions there is the choice of the Mac Lane or Kleisli algebras as introduced above in Section 6. It is the approach owing to Heinrich Kleisli that has elevated compositionality to a higher level, through the Kleisli lift, described for instance by Mulry [25]. In the diagram in Figure 18, H is a monad $< H, \eta, \mu >$ in $\mathbf{X}$ and K is a monad $< K, \gamma, \rho >$ in $\mathbf{Y}$. The Kleisli categories, representing the free algebras, are $\mathbf{X_H}$ and $\mathbf{Y_K}$. The Kleisli lift of functor F is the functor $\bar{F} : \mathbf{X_H} \longrightarrow \mathbf{Y_K}$ such that the diagram in Figure 18 commutes. Associated with this diagram is the definition of the lifting natural transformation $\lambda : FH \longrightarrow KF$ in Figure 19, derived through applying the interchange law to the component functors and natural transformations in the two monads defined above.

$$\begin{tikzcd}
\mathbf{X_H} \arrow[r, "\bar{F}"] & \mathbf{Y_K} \\
\mathbf{X} \arrow[u, "I_H"] \arrow[r, "F"'] & \mathbf{Y} \arrow[u, "I_K"']
\end{tikzcd}$$

Figure 18: Kleisli Lifting of Functor $F : \mathbf{X} \longrightarrow \mathbf{Y}$ to $\bar{F} : \mathbf{X_H} \longrightarrow \mathbf{Y_K}$

So far the Kleisli lift applies to any category, giving what is termed Kleisli prestrength. We now need to consider the Kleisli lifting of a bicategory, one involving a product of two categories. This is essential if the products are to be well defined for compositional purposes as indicated in Section 6. The lifting gives rise to what is termed Kleisli strength, forming the basis of the Cartesian monad, a term introduced earlier in our overview of the Haskell programming language in Section 5.5. The terms Cartesian monad and strong monad encountered in the literature are for our purposes interchangeable. The enhanced compositionality is achieved firstly by defining a natural transformation $\tau_{A,B} : A \times TB \longrightarrow T(A \times B)$ for objects A, B, C in the category $\mathbf{X}$ with monad $< T, \eta, \mu >$ such that the diagram in Figure 20 commutes. A further natural transformation $\lambda_{TA} : I \times TA \longrightarrow TA$ is also defined, as shown in the commuting diagram in Figure 21, to reinforce the interchange laws employed in Figure 19. Both the diagrams defining the Cartesian monad involve the Cartesian product, the most relevant for information systems, but the theory is actually more general covering the tensorial (outer) product $A \otimes B$, which may have more relevance

$$\begin{tikzcd}
F \arrow[r, "F\eta"] \arrow[ddr, "\rho F"'] & FH \arrow[r, "F\mu"] \arrow[dd, "\lambda"] & FHH \arrow[d, "\lambda H"'] \\
(\bar{F}C) & (\bar{F}D) & KFH \arrow[d, "K\lambda"'] \\
& KF \arrow[r, "\gamma F"'] & KKF
\end{tikzcd}$$

Figure 19: Kleisli Lifting of Functor F to $\bar{F}$: the lifting natural transformation $\lambda : FH \longrightarrow KF$

for studies involving vectors. Further diagrams are required when the product is tensorial, rather than Cartesian, involving multiplication through the arrow $\mu_{A\times B}$ and associativity though the arrow $\tau_{A,B\times C}$. A major advantage of Kleisli strength monads is that they can, in general, be composed naturally, unlike monads of weaker strength. Such composability increases reliability and scalability, both of which are vital for large scale information systems. Kleisli strength facilitates the discovery of distributive laws.

$$\begin{tikzcd}
A \times B \arrow[r, "1_A \times \eta_B"] \arrow[dr, "\eta_{A\times B}"] & A \times TB \arrow[d, "\tau_{A,B}"] \\
& T(A \times B)
\end{tikzcd}$$

Figure 20: Cartesian Monad: Diagram defining the natural transformation $\tau_{A,B}$

Meredith [21] envisages that the monadic design patterns, providing a broad range of transactional semantics, would have a front-end data sublanguage of the applied $\pi-$calculus, a compositional process calculus developed for concurrent programming by Milner [22]. However, other presentational techniques from category theory are available, such as bigraphs, and should also be evaluated before a choice is made.

$$\begin{tikzcd}
I \times TA \arrow[r, "\tau_{I,A}"] \arrow[dr, "\lambda_{TA}"] & T(I \times A) \arrow[d, "T(\lambda_A)"] \\
& TA
\end{tikzcd}$$

Figure 21: Cartesian Monad: Diagram defining the natural transformation λ_{TA}

In the functional programming language Haskell, monadic design patterns are employed. The design pattern for a category $\mathbf{C}$ is $\mathbf{H} =< H, \eta, \mu >$ where $\mathbf{H}$ is the monad with type constructor H, η is a return function, $\mu : HHA \longrightarrow A$ is a join function. In more conventional monad terminology H is the endofunctor, η the unit of adjunction and μ the multiplication [25]. If the monad is of the *Maybe* type, there are facilities for exception handling. To facilitate monad composition, the monad is lifted into a Kleisli category, with the power of a strong monad or a Cartesian monad. A monad composition operator, also known as the Kleisli composition operator, is available for composing one monad with another naturally [5].

Returning to our banking example we can see that composition of processes is readily available if our monads are Cartesian, with the Kleisli lift. So for two monads $\mathbf{T} =< T, \eta, \mu >$ and $\mathbf{U} =< U, \gamma, \rho >$, we can write UT for the composite process, where say T is the banking transaction with checks for its feasibility and U is a task establishing remote mirror facilities, as in distributed data recovery systems, for recording the results persistently. Such compositionality could be enforced over a large distributed systems by involving many individual monads. So monads can be used either in the small individually in a local environment or, through composition, in the whole universe of the information system. The efficacy of the monad approach can be proven through category theory, thereby increasing the reliability and robustness of a system, where every transaction is critical. Further the monad can be directly implemented in the programming language Haskell, enabling experimental results to be derived. An approach has therefore been developed for guaranteeing quality in an experimental environment.

8 Summary

The combination of the topos, as the underlying data-type, and the monad, as the process or transformer, guarantees the quality of experimental information systems. The topos is based on pullbacks, which can be nested recursively or pasted together for complex relationships. Data normalisation arises naturally through the rules of pullback construction. The subobject classifier of a topos facilitates internal queries on the information system. The monad is defined as three components for operations on a category: an endofunctor that is often an adjunction, the unit of adjunction and the unit of multiplication. There are two main approaches for applying the monad as an algebra: Eilenberg-Moore and Kleisli. The Kleisli approach finds favour, with its lift to Cartesian monads handling products, providing compositionality across a succession of monads and a route for a quality implementation in the experimental environment of Haskell.

9 Acknowledgements

We are very grateful to Michael Brockway of Department of Computer Science and Digital Technologies, Northumbria University, for helpful discussions on the theory of pasted pullbacks and monads.

References

[1] Adámek, Jiří, Herrlich, Horst; Strecker, George E, Abstract and concrete categories, John Wiley (1990). Recent edition at http://katmat.math.uni-bremen.de/acc (2005).

[2] Banach, R, Regular relations and Bicartesian Squares, Theoretical Computer Science **129**(1) 187-192 (1994). https://doi.org/10.1016/0304-3975(94)90086-8

[3] Beck, Jonathan Mock, Triples, Algebras and Cohomology, Reprints in Theory and Applications of Categories, Columbia University PhD thesis, 2: 159, MR 1987896, originally published 1967 (2003). http://www.tac.mta.ca/tac/reprints/articles/2/tr2abs.html

[4] Bunsen, Christian Karl Josias, Freiherr von; Hare, Julius Charles & Bernays, Jacob, Hippolytus and his Age, published Longman, Brown, Green, & Longmans, London 577 pp (1854).

[5] Diehl, Stephen, Monads made difficult. http://www.stephendiehl.com/posts/monads.html

[6] Eilenberg, Samuel, & Moore, John C, Adjoint functors and triples, Illinois J Math **9**(3) 381-398 (1965). http://projecteuclid.org/euclid.ijm/1256068141.

[7] Freyd, Peter, & Scedrov, Andre, Categories, Allegories. Mathematical Library **39** North-Holland (1990).

[8] Gödel, Kurt, Die Vollständigkeit der Axiome des logischen Funktionenkalküls, Monatshefte für Mathematik und Physik, **37** 349-360 (1930). Reprinted in Feferman, S, ed. Gödel Collected Works, volume 1, publications 1929-1936, Oxford, p.102-122 (even numbers) original; p.103-123 (odd numbers) translators Bauer-Mengelberg, Stefan, & Heijenoort, Jean van (1986).

[9] $\lambda-$Haskell: an advanced, purely functional programming language (2017). https://www.haskell.org/

[10] Heather, Michael, & Rossiter, Nick, Logical Monism: The Global Identity of Applicable Logic, Advanced Studies in Mathematics and Logic **2** 39-52 (2005). http://nickrossiter.org.uk/process/advstudiesmathsmonism.pdf

[11] Heather, Michael, & Rossiter, Nick, The Process Category of Reality, ANPA 31, Cambridge 224-262 (2011). http://nickrossiter.org.uk/process/anpa0911.pdf

[12] Johnson, M & Rosebrugh, R, Sketch data models, relational schema and data specifications. Electron Notes Theor Comput Sci **61** 51-63 (2002). http://www.mta.ca/~rrosebru/articles/sdmrsds.pdf

[13] Kent, William, A Simple Guide to Five Normal Forms in Relational Database Theory, Communications of the ACM **26**(2) 120-125 (1983). http://www.bkent.net/Doc/simple5.htm

[14] Lambek, J, & Scott, P J, Introduction to Higher Order Categorical Logic, Cambridge (1986). https://github.com/Mzk-Levi/texts/blob/master/LambekJ.,ScottP.J.IntroductiontoHigherOrderCategoricalLogic.pdf

[15] Lawvere, F W, Adjointness in Foundations, Dialectica **23** 281-296 (1969).

[16] Leinster, Tom, Higher Operads, Higher Categories, London Mathematical Society Lecture Note Series 298, Cambridge (2004)

[17] Leibniz G W, Monadologie 1714; translated by Nicholas Rescher, 1991. The Monadology: An Edition for Students. University of Pittsburgh Press. Ariew and Garber 213, Loemker 67, Wiener III.13, Woolhouse and Francks 19. Online translations:

Jonathan Bennett's translation; Latta's translation; French, Latin and Spanish edition, with facsimile of Leibniz's manuscript at the Wayback Machine (archived July 4, 2012); further editions établie par E Boutroux, Paris LGF 1991; Lamarra, A, Contexte GènGètique et Première Réception de la Monadologie, Revue de Synthese 128 311-323 (2007).

[18] Levene, Mark, & Vincent, Millist W, Justification for inclusion dependency normal form, IEEE Transactions on Knowledge and Data Engineering **12**(2), pp. 281-291 (2000). http://eprints.bbk.ac.uk/196/1/Binder1.pdf

[19] Lipovača, Miran, Learn You a Haskell for Great Good!, A Beginner's Guide, William Pollock, San Francisco (2011).

[20] Mac Lane, Saunders, Categories for the Working Mathematician, 2nd ed, Springer (1998).

[21] Meredith, Lucius Greg, Monadic design patterns for the Blockchain, DEVCON1, Ethereum Developer Conference, Gibson Hall, London, 9-13 Nov (2015). https://www.youtube.com/watch?v=uzahKc_ukfM&feature=youtu.be

[22] Milner, Robin, Communicating and Mobile Systems: The $\pi-$calculus, Cambridge (1999).

[23] Moggi, Eugenio, Computational Lambda-Calculus and Monads, Proceedings of the Fourth Annual Symposium on Logic in Computer Science 1423 (1989).

[24] Moggi, Eugenio, Notions Of Computation And Monads, Information And Computation **93** 5592 (1991).

[25] Mulry, Philip, Notions of Monad Strength, Banerjee, A, Danvy, O, Doh, K-G, Hatcliff, J, (edd.) David A. Schmidts 60th Birthday Festschrift, EPTCS 129, 6783, doi:10.4204/EPTCS.129.6 (2013). https://arxiv.org/pdf/1309.5132.pdf

[26] ncatlab, Pullback as an Equalizer https://ncatlab.org/nlab/show/pullback

[27] Phys Org, Bitcoin's 'blockchain' tech may transform banking, http://phys.org/news/2015-12-bitcoin-blockchain-tech-banking.html

[28] Rossiter, B N, Heather, M A, & Sisiaridis, D, Process as a World Transaction, Proceedings ANPA 27 Conceptions, 122-157 (2006). http://nickrossiter.org.uk/process/anpa064.pdf

[29] Rossiter, Nick, & Heather, Michael, Formal Natural Philosophy: Top-down Design for Information & Communication Technologies with Category Theory, ANPA 35, Explorations, Grenville J Croll, Nicky Graves Gregory (edd.), 155-193 (2015). http://nickrossiter.org.uk/process/anpa-2015-a5-Latex.pdf

[30] Rossiter, Nick, & Heather, Michael, Abstract Relations and Allegorical Categories, ANPA 36, Explorations II, Anton L. Vrba (ed.) 103-134 (2016). http://nickrossiter.org.uk/process/Rossiter-ANPA-PROC-36updated.pdf

[31] Whitehead, Alfred North, Process and Reality: An Essay in Cosmology, Macmillan, New York (1929); corr.ed., eds. David Ray Griffin and Donald W. Sherburne, New York: Free Press (1978). https://monoskop.org/images/4/40/Whitehead_Alfred_North_Process_and_Reality_corr_ed_1978.pdf