IDIOM
PURE DECISIONING

The Case of the Missing Algorithm

And what has Haecceity got to do with it?

IDIOM Article

☎ +64 9 6308950 | @ idiomsales@idiomsoftware.com | idiomsoftware.com
✉ 2-2, 93 Dominion Road, Mount Eden, Auckland 1024

CONTENTS

Meet Quiddity and Haecceity

'Quiddity' and 'Haecceity' are two unusual words that aren't often used in daily conversation – especially IT conversations.

Which is a pity because they exactly describe a concept that is an important factor in successful delivery of commercial IT systems.

Let's see what the words actually mean (thanks to The Free Dictionary):

Quiddity: "The real nature of a thing; the essence"[1]

Haecceity: "The property that uniquely identifies an object."[2]

While the definitions are closely aligned, the slight difference is material to this conversation.

This time from **Wikipedia**:

"Haecceity may be defined in some dictionaries as simply the 'essence' of a thing, or as a simple synonym for quiddity. However, such a definition deprives the term of its subtle distinctiveness and utility. Whereas haecceity refers to aspects of a thing that make it a particular thing, quiddity refers to the universal qualities of a thing, its 'whatness', or the aspects of a thing it may share with other things and by which it may form part of a genus of things."[3]

To provide relevance to this conversation, we need to clarify what thing or object we are talking about – and that thing or object is the enterprise itself, in whole or in part.

There are a small number of core concepts that can be said to embody the essence of an enterprise. This article asserts that one such concept is the business algorithm, the unique combination of business logic, algebra, and rules that is used by the enterprise to convert real world data and events into useful outcomes that benefit all stakeholders – giving rise to happy customers, prosperous proprietors, and fulfilled staff!

In keeping with the definition of quiddity, the business algorithm will have numerous common constructs that are readily identifiable as being 'of the domain' and which are generally shared across all participants in the domain, so that (for instance) insurance systems have a sameness that is shared worldwide. As a specific example, the idea of 'earning premium' is universal and has the same implementation (more or less) in systems everywhere.

Then, in keeping with the uniqueness definition of haecceity, the general domain algorithm is extended by unique constructs that further and uniquely describe each specific enterprise. For instance, how risk is accepted and priced (underwriting and rating) is usually calculated according to the particular requirements of the owning enterprise.

---

[1] https://www.thefreedictionary.com/quiddity
[2] https://www.thefreedictionary.com/Haecceity
[3] https://en.wikipedia.org/wiki/Haecceity

The business algorithm is a unique and fundamentally important concept that no enterprise can function without. There are other aspects of the enterprise like brand or culture that may also claim to be 'of the essence', but the business algorithm is the only such concept that has a formal existence inside computer systems. The business algorithm is like the soul of the enterprise, uniquely defining the enterprise and giving it life via its systems. As such it has a unique claim to relevance as a first-order systems requirements artefact.

What is the Business Algorithm?

The expansive description of the business algorithm asserted by this article was not part of the rules lexicon when we started our decisioning journey in 2001. The concept of a single artefact that embodies all business logic, algebra, and rules across the depth and breadth of an enterprise is too big and too all-encompassing to seem plausible.

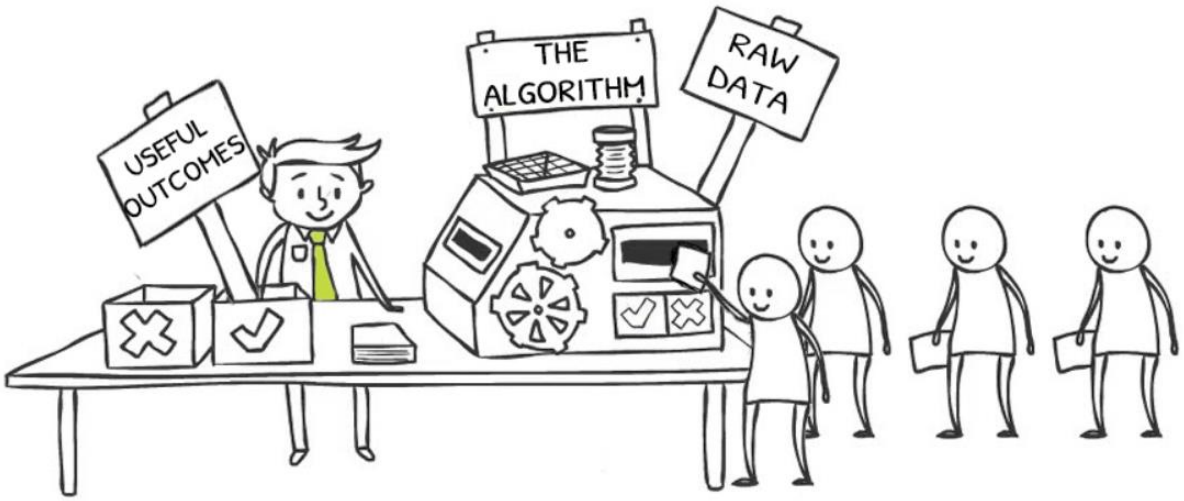
But plausible it is! Hands-on development of business algorithms at this scale over the past decade has empirically demonstrated the practicality of the concept, and furthermore, it has delivered outstanding ROI outcomes in doing so.

Simply put, the business algorithm is all of the business logic, algebra, and rules that transform raw data into useful outcomes across the full extent of the subject enterprise – and yes, it is a big thing. What has made it viable today as a first order requirements artefact is use of a topology to break-down the complete business algorithm into manageable, business aligned units-of-work. Each unit-of-work should represent a distinct subset of the enterprise. There is no existing and agreed term for these units-of-work, so we will use our own term for them, viz. decision models. In aggregate, these decision models, each of which is a properly formed and separately executable sub-assembly of business logic, algebra, and rules within the overall business algorithm, describe a single, correct, consistent, and complete business algorithm that spans the enterprise.

We consider the business algorithm to be the supreme business requirement from a systems perspective because we can and should build the algorithm in its entirety without consideration of existing or future systems, and without regard to technology now or in the future. It is a purely business artefact that does not require any IT input to develop and test. It defines explicitly and exactly the value proposition of the enterprise independently of any systems that might implement it.

For the sake of clarity, the business algorithm that is the subject of this article manifests as a distinct 'thing' – it is a tangible, user accessible model of business behaviour, self-documenting and empirically testable.

When we have the fully formed business algorithm in our hands (so to speak), we can inspect it to infer the inbound data requirements (the externally sourced 'real-world' data that drives the algorithm); and similarly, we can inspect it to identify the outbound data that defines the 'useful outcomes', which are the purpose of the algorithm.



Note that this data is discovered as a natural consequence of defining the algorithm. However, the inverse is not true – the algorithm is never discoverable as a consequence of defining the data. In this way, the algorithm becomes the primary source of data requirements.

And when we understand the inbound and outbound data requirements, then we can further infer processes that are required to efficiently acquire the inbound data (in response to external events), and to respond to the useful outcomes. The business algorithm is therefore also the primary source of process requirements, albeit transitively via the data.

How Does this Differ from Traditional Approaches?

This contrasts with traditional approaches, which put the data and/or processes in a superior position to the business logic, algebra, and rules that together form the business algorithm. In fact, some versions of OMG's UML appear to specifically exclude the business algorithm from consideration. Reference version 1.5 section 3:22:3 viz.:

"Additional compartments may be supplied as a tool extension to show other pre-defined or user defined model properties (for example to show business rules…) ...but UML does not define them..."

The consequence of treating business logic, algebra, and rules as dependent properties of data and/or processes is that the enterprise knowledge and requirements that the business algorithm represents are scattered across systems, processes, and data in such a way that it is often infeasible, if not impossible, to accurately re-collate them. They are more or less lost to view, hence the title of this article.

Our own experience in harvesting the business algorithm from existing systems shows that the scattered logic is often hard to find and re-collate, and even when found is likely to include inconsistencies, missing edge cases, redundant implementations of the same thing, and/or outright conflict, wherein multiple implementations give different answers for the same underlying requirement. Even if full reconstruction of the logic that is embedded in existing systems is achieved, recourse to the business and/or first principles is usually required to clarify

and confirm these findings to arrive at the definitive correct, consistent, and complete business algorithm.

We can intuit that the business algorithm exists in every enterprise, because we know that enterprises convert raw data into useful outcomes every day. At this point you may be wondering whether, like Neo in the Matrix, you should take the red pill or the blue pill!

In this case the red pill is the one to take because we can show that the approach outlined herein is real and it works – we have been successfully building enterprise scale business algorithms for nearly a decade. This author has seen them and worked with them for enterprises on a large scale – for instance, a single omnibus algorithm deriving all billing line items from raw clinical data for 42 hospitals and 121 clinics[4]; or complete recalculation and remediation of an active $65billion defined benefits scheme for a million-member pension fund, right down to adjusting precision as we cross multiple systems implementation boundaries during the 35 years the scheme has been operating.

If enterprise scale algorithms are real as described, then the traditional IT approaches to requirements gathering, and systems architecture, design, and development, have just been offered a substantial challenge. Confirmed existence of the business algorithm means that we should invert the traditional development approach so that the data and process requirements are derived as dependents of the algorithm rather than the other way around. Our experience is that this can lead to (up to) an order of magnitude improvement in time, cost, and risk to produce equivalent systems.

I recently met a senior Government figure who has been charged with rebuilding his country's entire welfare payments system, which distributes hundreds of billions of dollars annually. The current environment is spread across 250 legacy systems, with no overarching data, process, or rules design. The underlying business algorithm is not visible or even apparent.

So where would we start? We could acquire and normalise the data across all of the systems to give a data centric viewpoint. We could analyse the external events, with their inputs and outputs, to develop a process centric architecture. But in either case, we would still not have the business algorithm, so how would we calculate the entitlements that are the purpose of the system? Furthermore, without the algorithm, we do not have any independent means to validate that these re-derived data or process maps are correct or relevant. At best, we could say that they mirror the legacy environment, but if that is our objective, we haven't advanced much – I am sure the Government will be hoping for more.

However, if we build the business algorithm first and foremost, then we can clearly infer the data; and when we have the data we can devise optimal processes to support it. These new data and process models are likely to look rather different compared to the legacy influenced maps described above. And using this approach de-risks the overall project, and substantially reduces time and cost to deliver.

---

[4] This use case is the basis for our submission to the Business Rules Excellence Awards 2018, available on our website http://idiomsoftware.com/DOCS/Download/d3058f0c-ce4c-4322-8c93-914378f8e178.pdf

We suggested earlier that discovering and re-collating the business algorithm from existing systems can be infeasible, sometimes impossible. However, we have developed techniques and approaches that help mitigate this problem.

Building the Business Algorithm

The key to building an enterprise wide algorithm is the topology. The topology is a breakdown of the business algorithm into logical units-of-work (which for convenience we have called 'decision models') that allows us to normalise the business logic, the algebra, and the rules within. In this context, 'normalise' means to reduce the algorithm to its simplest form, which requires that it be described using the smallest number of parts, without overlap or redundancy, that is required to achieve the outcomes.

There are several ways to break-down the entire algorithm so that we can address it piecemeal. None can be based on systems, data, or processes, which are artificially derived constructs that do not inherently define the enterprise, and in the context of a new solution, they should not even yet exist (and if they do, we must challenge the basis on which they were created, lest we simply remake the mistakes of the past). Of course, the real-world raw data exists somewhere 'out-there', but we do not yet know which of that data is relevant to the mission of this enterprise – only the algorithm can tell us that.

This is the critical point of this paper (assuming that you have taken the red pill) – if the business logic, algebra, and rules are scattered across the existing system components as properties of data and/or process, then by definition:

a) those data and/or process components must already exist, rightly or wrongly (and there is no way to empirically assess rightness or wrongness prior to the algorithm, which has been 'lost' by virtue of its scattered, piecemeal implementation); and

b) no competing classification scheme for normalising the business logic, algebra, and rules in the algorithm is possible, since they are already declared to be properties of data and/or process.

This means that the specification of the business algorithm is compromised by having to adhere to, and to be normalised in accordance with, an artificial topology that is an historical accident at best. And for those with a general knowledge of normalisation, having to normalise the internal elements of the algorithm around predefined data and process constructs can only constrict and compromise the definition of those elements. Essentially, normalisation cannot be achieved around a fixed topology without forcing the meaning of the thing being normalised, potentially leading to redundancy, errors, and omissions. The implication is that this approach is detrimental to the correct normalisation and definition of the algorithm, and by extension, of the enterprise itself.

By following the traditional approach, we are implicitly locked into an historical data and process architecture in which the algorithm has neither status nor visibility. And if the data and/or processes already exist, by what means were they discovered and validated without the business algorithm. By definition, the only purpose for both data and processes is to support the algorithm, which is the source of the 'useful outcomes'; without the algorithm,

they have no purpose. No system should be collecting and processing data without prior understanding the 'useful outcomes' that are required.

Therefore, we need a new topology. Our new topology concept should precede the definition of data and processes, and will in fact ultimately define them both. It should be based on existing and fundamental business structures including:

- **First principles.** In our Government payments example, the first principles will be found in legislation and regulations – for instance, we have currently executing decision models that exactly implement key pieces of legislation. In financial organisations like banking and insurance, the first principles extend beyond legislation and regulation to include the customer promises and obligations that arise from product disclosure statements, etc.
- The inherent subdivisions of **responsibilities** and **expertise** within the enterprise provides another useful dimension, for instance product ownership, finance, customer management, etc. Each of these subdivisions of enterprise governance should be represented by relevant proxies in the business algorithm.
- The underlying four step **algorithm lifecycle**, from raw data to useful outcome. This includes inherent and logical subsets of the business algorithm that manage:
  a) Data acquisition and validation; do I have enough data of suitable quality for the calculation of the useful outcomes?
  b) Data transformation into the idiom of the enterprise. The calculation of outcomes is always defined using an idiom that is proprietary to the enterprise, unless it is a commodity business that has no 'haecceity' – that is, no unique characteristics. Perhaps surprisingly, transformation to the idiom is often the major part of the business algorithm's calculation effort.
  c) Calculation of the useful outcomes that are the proximate purpose of the business algorithm, and by extension the enterprise itself. This usually means approval for and then value of each specific transaction context (in our Government example, is the person entitled to be paid, and if so, how much? In insurance, this would include underwriting and rating, and/or claims adjudication and payment.)
  d) Workflow. What do we do next? What, when, and how do other systems and processes need to be advised of the useful outcomes?

Each of these three major perspectives (including the four lifecycle steps of the algorithm) is an original source of requirements that will only change when the business itself changes at a fundamental level. Each may require us to redirect analysis effort back to existing systems in order to understand the current implementation of the algorithm, so that we do usually end up inspecting legacy systems in detail. However, the legacy implementations are only a reflection of somebody else's interpretation of requirements at some earlier point in time. They may be a corroborating source, but they do not fundamentally define the requirements.

If we aim to normalise the algorithm, then we need a classification structure that truly reflects the quiddity and the haecceity of the enterprise, and which will last the test of time. This ideal structure is not based on data or processes, which are artificial and temporal constructs.

Each of the three major perspectives above are important to constructing a long-term classification structure (i.e. the topology) for the algorithm, and their intersection may give rise to many decision models, so that a complete topology might include tens to hundreds of decision models.

Deriving the specific useful outcomes for any particular transaction will likely require adjudication by many of these decision models within the boundary of the transaction being executed. This gives us an orchestration problem within the overall business algorithm, which we resolve by building a meta (decision) model to manage the orchestration of the many topographical models required.

The net result is that any process that is triggered by an external event simply presents its raw data to the business algorithm for adjudication and conversion into the useful outcomes. Not only does the external process not understand what decision models will process its request, it must not know. If it explicitly requests the execution of specific models, then it presumes an internal knowledge of the business algorithm, which by definition means it has ceased to be a business only artefact – it is now inextricably bound to the process implementation.

For the sake of clarity, we are proposing that all business logic, algebra, and rules can be logically defined into the business algorithm, on a fully normalised basis so that there is only ever one representation of any logical requirement within the business algorithm, and the algorithm in aggregate is correct, consistent, and complete across the enterprise.

The Algorithm in Operation

However, this is not to be confused with implementation – the entire algorithm, or distinct subsets of it, can be deployed into multiple execution locations so that it can run 'in process' wherever it may be required. This allows for fast execution.

In our own case, this is usually achieved by generating the complete set of decision models into code and loading the executables (dll's, JARs) into the database. Any location (back-end processes, web servers, centralised services, etc.) can access the algorithm's public interface ('in process' or via a connector), which then loads and caches the executables as required from the database. The relevant set of models required for any particular location is defined by the nature of the transactions that are visible at that location, and may not include all decision models that comprise the business algorithm. This approach is currently being upgraded to utilise more recent deployment strategies including Docker, AWS Lambda, microservices, etc.

We normally expect <10 milliseconds for standard calls to the algorithm, but some extreme cases that process large amounts of data inside the transaction can extend this by an order of magnitude or more.

What is a large amount of data for a single transaction? Anywhere from 1million-100million data nodes. Anything less than 1million data nodes per transaction is considered business as usual. At 100million nodes we suggest a change in approach to the definition of transactions.

To get some sense of the scale of the decision models and algorithms that we are talking about, and of the set that may be distributed: Our largest single decision model generates

~300,000 lines of code (Java and/or C#); our largest set of decision models executed in a single request is several dozen; and our largest business algorithm has <100 decision models in the enterprise wide set, so that the complete business algorithm for an enterprise can easily extend to millions of lines of code.

Conclusion

The above discussion describes an approach that captures the quiddity and the haecceity of each unique enterprise by capturing its own particular 'business algorithm'. Of great importance, the business logic, algebra, and rules in this algorithm must be normalised (similar to how we normalise data to get a sound data design). Managing and navigating this normalisation process requires a topology that is derived from the enterprise's fundamental structures. This specifically excludes systems – the business algorithm is captured and normalised independently of systems considerations to ensure that it is not contaminated by artificial or historical technology or methodology constraints. And in an inversion of current practice, the business algorithm can then be used to re-derive both data and processes from the first principles that are provided by the algorithm. This approach reduces time cost and risk to develop systems by substantial margins.

Finis

IDIOM
PURE DECISIONING

Mark Norton | CEO and Founder | Idiom Limited

☎ +64 9 630 8950 | +64 21 434 669 | After hours +64 9 817 7165 | Australia free call 1800 049 004
✉ 2-2, 93 Dominion Road, Mount Eden, Auckland 1024 | PO Box 60101, Titirangi, Auckland 0642, New Zealand
@ mark.norton@idiomsoftware.com | idiomsoftware.com | Skype Mark.Norton