Decisions as First Class Citizens within Systems Development

European Business Rules Conference

Mark Norton

17 June 2004

IDIOM

Introduction

- This discussion is about Decisions, and about how elevating them to first class citizens within the systems development lexicon will be instrumental in returning control of business systems to business people.

- Let’s get started . .

What are decisions?

- We really need to be sure of what we mean by decisions, so we will start here. We are talking about a subset of ‘business rules’

- In particular:
  - Business rules that apply corporate expertise to domain objects - to quantify, qualify and classify them for subsequent processing according to the unique requirements of the organization.

- I want to highlight the point that uniqueness is principally associated with ‘quantifying, qualifying and classifying’ domain objects for the specific processing requirements of THIS, the rule owning, organization.

Domain Objects

- Domain objects is a term that we use to refer to the data subject areas that describe the problem domain in data terms.
- Because these describe real world entities and events, they tend to be more stable and generic – which is why we can anticipate industry specific domain object models such as MDDL, ACORD, HL7, etc.
- Traditional vendor applications count on a high degree of intra-industry consistency. For example, I am sure that most of you would describe the `customer' domain object in a very similar way – because customer is a tangible 'real-world' entity.

Domain Objects cont ...

- But rules are more ‘personal’ in a corporate sense - that is, how we quantify, qualify and classify customers will vary from one organization to another far more than the data that describes them.

- This particular subset of rules is what we will collectively call decisions.

Note : rules that describe the domain object itself, that is, rules that describe the data, are excluded. As a rule of thumb, if it can be described by a schema, it is a part of the Fact Model, not the Decision Model

---

Business Rules - According to Idiom

Idiom's view is that business rules:

- Apply **proprietary** business expertise to domain objects
  - Quantify, qualify or classify the object
  - Will change the object if the rule applies
- So that the domain objects can be further processed according to the unique requirements of **this** business
- Are only constrained by the 'Fact Model' that is defined for the domain*

We call these rules `decisions'

*__Business Rules and Information Systems: Aligning IT with Business Goals (Tony Morgan)__

- The fact model is composed of one or more domain objects
- These are defined using W3C schema definitions
- There are no timing, technology, structural or other constraints on the business' ability to set rules

Decisions

- Decisions are the concept that most directly implements business policy and practice.
- Reducing decisions to a single valued item is the key to making the decision model a practical and finite tree.
- This is critical to producing a level of granularity in our model of decision making behavior that is appropriate for formula building, and ultimately to our ability to implement and automate the decisions.
- We would now like to nominate some specific definitions.

Decisions are a Subset of Business Rules

- Decision: “a single value that is derived by the business to qualify, quantify, or classify a domain object so that it can be further processed by the business in accordance with its declared business policies and practices”
- Formula: “the logical expression that describes the transformation of named input variables into a Decision output value”
- We intend to show that **Decisions** are the key to quickly and accurately defining modern business systems
  - Automation of decision making is the primary concern for most new systems
  - Decisions drive both data and process in systems construction

Examples of Business Rules

- Let's look at some examples of what Idiom would consider to be business rules. These rules do not describe a static 'worldview' in the same way that the domain objects do as part of the Fact Model.
  - Gold card customers have a limit of 20000; Silver 10000
  - Imported turbo charged vehicles are a favorite of 'boy racers' and will be excluded from cover
  - Patients who spend more than 5 hours in surgery recovery will be considered more serious cases and will attract 10 more points of funding

Examples of Business Decisions

At 10.00am the executive committee met and agreed the changes on this slide which appear in green. And they said . . . ‘Please make this effective at 6pm on the 17th of June 2004!”:

- Gold card customers have a limit of 20000; Silver 10000
  - unless they have had an average daily balance of greater than 5000 for the previous 3 months, in which case 20000
- Imported turbo charged vehicles are a favorite of ‘boy racers’ and will be excluded from cover
  - only applies to second hand cars
- Patients who spend more than 5 hours in surgery recovery will be considered more serious cases and will attract 10 more points of funding
  - if mechanical ventilation is required

If we have a closer look at these changes ...

We can see that Decisions . . .

- Are owned by the business
- Are dynamic and have no time constraints – can change at any time
- Are technology and system independent

The constraints are no more nor less than if we were automating a decision point in the application to support a human decision maker – the parallel is clear – what we are doing is automating the decisions that implement corporate policy and practice.

- To be automated the decision
  - Needs the right input data to be available
  - Needs a system capable of responding to the decision outcome

When these conditions are met, decisions can be delegated entirely outside of the system boundary. Decisions become CONTENT, not permanent systems infrastructure

Are Decisions an IT Responsibility?

- So why do we analyze and build systems as if all decision rules are a series of static domain characteristics in the same way that the Fact Model is. Do all human actors who make decisions obey the constraints of a SDLC?
- Eventually we get to the point where we can contemplate whether decisions have anything to do with IT at all.
- Let's take a small diversion into modern IT analysis practice and look at this question a little closer. We will consider UML as a representative process that many would regard as important when building systems today.

Are Decisions an IT Responsibility?

- If we refer to version 1.5 of the UML Specification, we can see that business rules are explicitly excluded from the specification. After searching on ‘rules’ we find the following quote:
  - OMG UML V1.5, Section 3.22.3

    “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**). Most compartments are simply lists of strings. More complicated specifications are possible, but UML does not define them – they are a tool responsibility”

- We interpret this to mean: rules are not addressed by the standard, but may be supported by individual tool vendors

Rational Unified Process – a tool vendor approach

- RUP captures business rules in Business Modeling and Requirements workflows
- RUP **Business Rules Guidelines** indicate how to do this

Rule Type	Model	How to Express
Stimulus & Response	Activity	As decisions and conditional paths
Action Constraint	Activity	As conditional paths
Structure Constraint	Class	Constraints on associations between classes
Inference	Activity	As alternative paths and activities
Computation	Class	Methods and relationships between classes in the object mode

RUP and OO design

Tool vendors are allowed to implement rules – maybe RUP has addressed the issue. Or has it? Let's look at the two major workflows that RUP uses to address business rules. The first is OO based 'Business Modelling':

- In practice, using UML class diagrams to express business rules is not very effective. This is a problem of object-oriented design as much as a problem with UML.
  - Amongst the major design principles of OO design are the principles of abstraction and encapsulation.
  - A class is an abstract representation of some concrete entity (e.g. Customer) and encapsulates the data related to the abstraction in question, along with the methods or operations that manipulate that data.
  - But there are many instances of business rules that do not fit cleanly within a single class. This makes it harder to identify and then verify the correctness of the business rule.
- Another problem with using UML to capture business rules is that UML diagrams in themselves are not adequate to express all business rules.

RUP Requirements Modeling

The second major RUP workflow is the Requirements workflow.

- RUP mandates that functional requirements are captured using use case models. Use cases are documented using textual documents, which have only limited structure – if any.

- Use Cases capture requirements
  - "A sequence of actions that the system will perform to yield an observable result to a particular actor"
  - Can be a context or a 'process container' for decision making
  - Usually include one or more business rules

RUP Requirements Modeling

- BUT as with OO, we have the problem that there is no RUP concept which cleanly contains rules.
  - Certainly they can be many-to-many with Use Cases, which in itself implies a missing core entity from the RUP methodology – simple normalization of the meta-data implied by the Guidelines demonstrates this.
- The result is that rules are invariably found in poorly defined ‘supplementary documents’.
- So, again, with RUP it is hard to locate, verify, and manage rules

Summary of Issues for UML/RUP Management of Business Rules

- Lack of an explicit concept to define and implement rules
- Lack of a single, unified method of expressing business rules
- Business rules often apply to more than one requirement or class
- Definition of business rules using UML class diagrams results in fragmented business rule definitions which are difficult to verify
- Future changes to rules are constrained by the ‘SDLC’ – costing time and development effort
- Implementation of rules requires collaboration across many roles – leading to translation risk, increased workload and scheduling conflicts

This evidence suggests that traditional IT development does not give decisions the same importance that the business does!

Decisions aren’t static

- Unfortunately, even if the authors of UML decided to get serious about business rules, they would have a challenge architecting them into the existing community of UML ‘first class citizens’.

- If UML was to be revised so that it provides for Decisions as a first class citizen in the lexicon, what would it look like? Actually, we suggest that the question is redundant.
  - There is an entire SDLC in play that has as it’s most fundamental assumption that the specification provides an all encompassing but static view of the system at a point in time – a view that is translated into system activity only through the design, coding, and testing activities of a traditional SDLC.
  - This is not an appropriate development process for something as dynamic as we are suggesting decisions are!

- Decisions are never static because businesses are never static– the business is not inherently constrained as to when they can be changed, therefore they are, conceptually at least, permanently in a state of change. In fact, they are Content not structure.

Treat Decisions as Content - don't lock them up in system code

- After the brief diversion via UML, we are back to the key point:
  It is time to re-think how we think about and model this type of business rule.

- The key issue is that decisions can be defined, modeled, implemented, and managed on an ongoing basis independently from the system itself. In fact, this activity can occur even if there is no system context at all – we only need the system when we want to automate the decisions.

- Decisions have a discrete and independent life cycle that is defined by the unique needs of decisions as "first class citizens"
  - The system does not need to know when or if a decision has changed or been added or deleted
  - The decision does not need to know how it will be implemented

End User Business - Benefits

Let's assume for a moment that we can achieve separation of decision making rules from the traditional IT development context. What would we gain?

- Relief from onerous IT development cycle times and costs
- Rule visibility and focus
  - Places business policy and practice to the forefront
  - Transparent and auditable (Basel, Enron)
- Retention of rules ownership by the "source of truth"
  - No obfuscation
  - No translation risk
- And when rules capability is fully developed
  - Business controls the response to new business demands

These benefits will be familiar to most of you, and we have only included them for completeness. But this small set of benefits make a compelling case.

Systems Development - Benefits

- System development task is reduced in size to the extent of the rules
  - 5-40% reduction in code, with 20% typical
  - Development effort reduces exponentially with size
  - Faster, cheaper, less risk
- A key requirements constraint is removed
  - Parallel development of rules
  - No milestone dependencies, including go live
- Major cause of future change and instability is removed – permanently
  - Gartner estimate 5-40% saving of IT costs before savings of labour
    [Gartner estimate from the Symposium ITXpo, Sydney Australia, November 2002]

Systems Development - Benefits

- The implied time and cost benefits for the business can be substantial. Many of these will come out of the development function itself.
- Note that Capers Jones studies on development performance have solidly established that a reduction in the size of the code base causes a much larger reduction in the overall development effort.
- So we are reducing time, cost, and risk simultaneously.
- To emphasise the dynamic nature of rules, and in particular their complete **dis-respect** for systems development agendas, I was witness when Allianz in Australia went live with their new electronic underwriting system - they needed to make a rule change on the very day that they went live. Fortunately, because they had achieved a high degree of separation, this was easily and safely done.

Vendor Software Industry - Benefits

- Another large contributor to overall business benefit is the commercial software sector. This sector and their customers in particular have much to gain from using rules components within their systems.

- These are detailed on the next slide but two items in particular stand out:
  - Some key benefits are faster development and cheaper applications that are more flexible across customers, regions, and time
  - There are also some more fundamental changes that we already see occurring in the nature and positioning of the vendor systems themselves. For instance, our demonstration is of a smaller rules-based niche system that would not have been viable as a traditional hard-coded SDLC developed application.

Vendor Software Industry - Benefits

- Externalising decision processes in vendor applications
  - Allows dynamic customization of decisions by end users
  - Vendor supplied customization and ‘pluggable’ options support regional or customer variations
  - Variations over time can be supplied as ‘knowledge components’
- A visible new sector for multi-party community applications that support more complex process requirements e.g. Health
- Standardized messages are leading to smaller, ‘niche’ vendor systems from a greater number of vendors
  - Mix and match decision making ‘best practice’ with application platform
- Faster development, cheaper applications that are more flexible across customers, regions, and time

Decisions are the core artefact of System Definition

Let's get to the core of the debate:

- By coincidence, our first two insurance customers, NZI in NZ and Allianz in Australia, both use Polisy - an old insurance application that came out of NZ in the early 80's - along with 200 like-minded companies.

- Imagine that these companies had identical implementations – similar data, similar process by definition. What would differentiate them?
  - We suggest that it would be the decisions they make and how they make them. About when and how they accept customers, accept risks, price insurance policies and pay claims
  - It is not the process but the **decisions** that drive the process that are important.

Decisions are the core artefact of System Definition

- While I was walking down Auckland's main street with the very first list of decisions for our angel customer, NZI (NZ's largest insurer at the time), I realized that I had never seen an inventory of business decisions in my previous 25 years in the industry!

- But more importantly, I realized that if I have an inventory of business decisions (and the formulas that support them), then I can:
  - derive all of the important data;
  - infer the processes;
  - and I can even infer the culture of the organization.

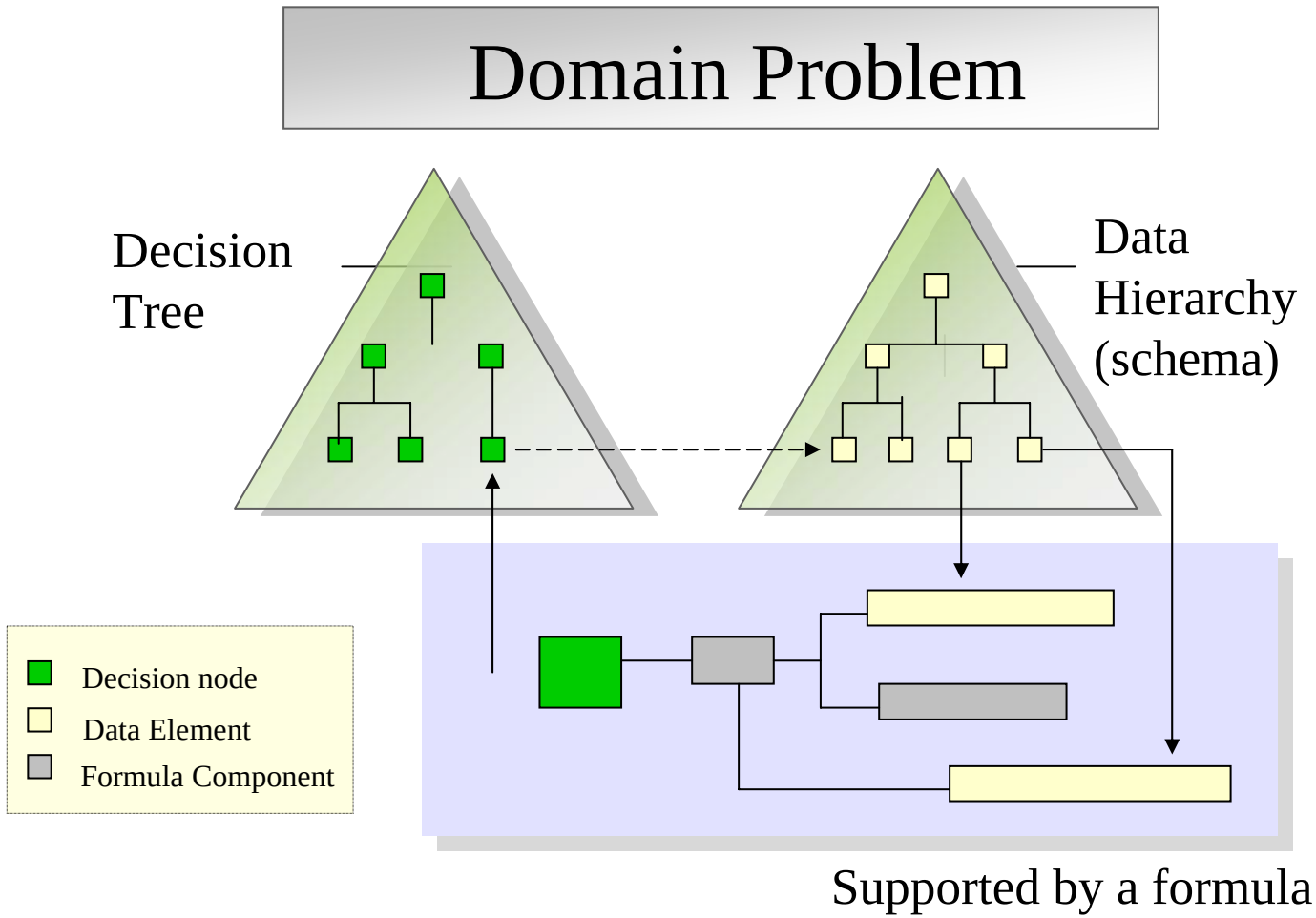
Decisions are the core artefact of System Definition

- From our first inventory of decisions we observed that we could reverse engineer:
  - Required data, including structure
  - Implied processing model
  - Implied culture
  - In short, the entire business operation!
- But the reverse is not true for data and/or process models - Decision making cannot be inferred.
- Therefore decision making is a more complete description of the business than either data or process models
- And, codified and automated decision making is a primary reason for the development of modern systems

Decision-centric development

- **First declare the domain problem**
  - It must be finite and achieve a business goal. For example “Rank and select a suitable organ recipient” as in our demonstration
- **Then develop the decision tree**
  - The decisions are the primary link to business policy and practice – which is usually why the system is being built at all
  - Break down the tree into more and more granular decisions until ‘single valued’ decisions are found – this terminates the decomposition.
- When all domain objects i.e. the data, are also decomposed to the attribute level, and decisions describe a single valued outcome, then the granularity is appropriate to start on the formulas – the domain problem has been decomposed into manageable pieces.
- It is at this point that traditional systems development approaches can be re-considered – use cases may be needed to gather the data, but this is not the concern of the rules builder.

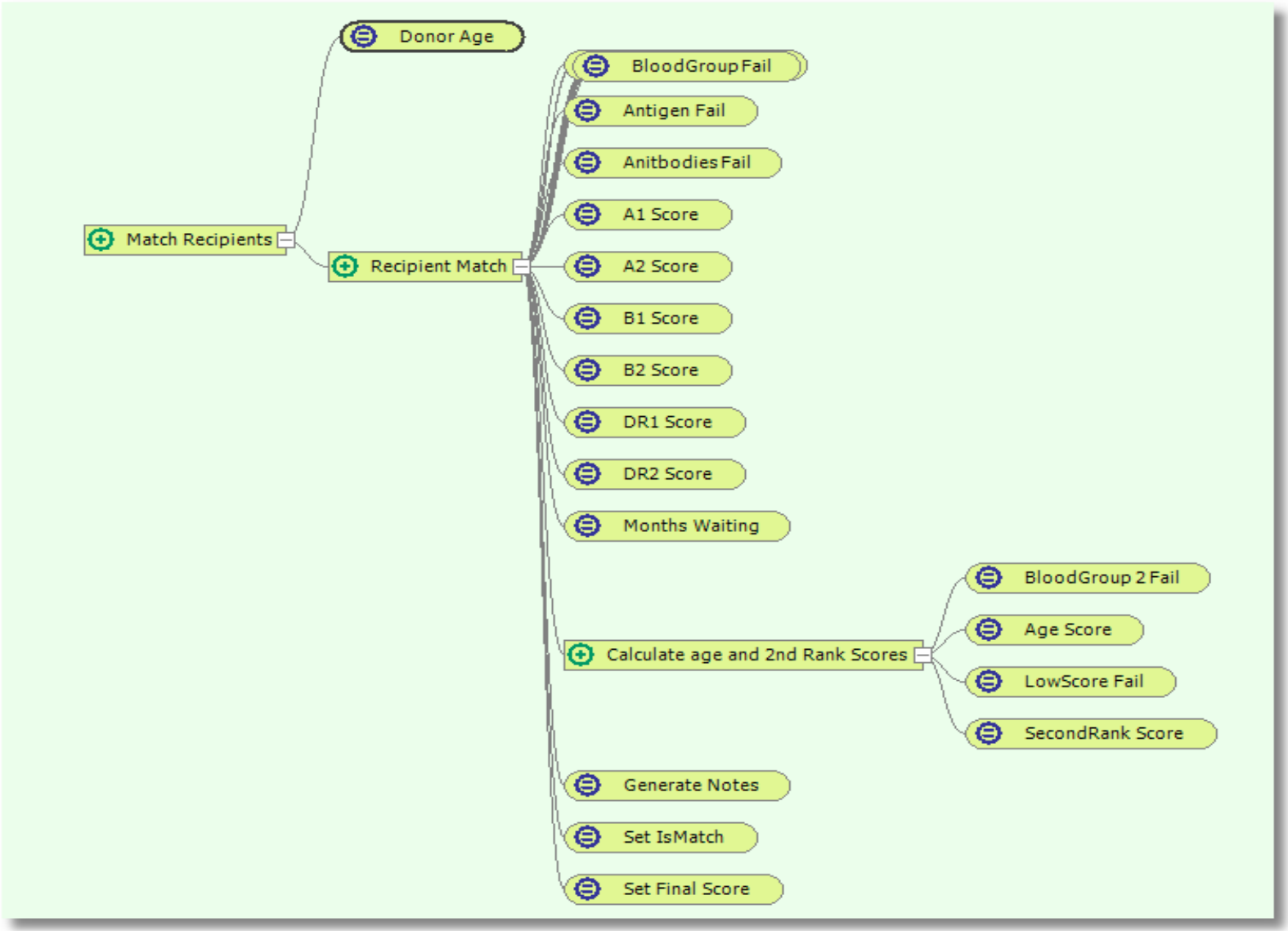
Decision-centric Development



Creating the Decision Tree

- Ask of the ‘source of truth’
  - ‘what decisions do you need to make in order to [resolve the domain problem]’
  - Examples: Underwrite a fleet of vehicles; Admit a patient
- Use ‘mind-mapping’ techniques to decompose the high level decisions down to single valued, simple decisions
  - These are now suitable for translation into formulas
- Reconcile continuously with the Fact Model (one or more data schemas) that is supplying the ‘decision vocabulary’
- The decision tree itself can be presented using a ‘mind-map’ metaphor. Here is an actual decision tree from our demonstration application

The Healix Decision Tree

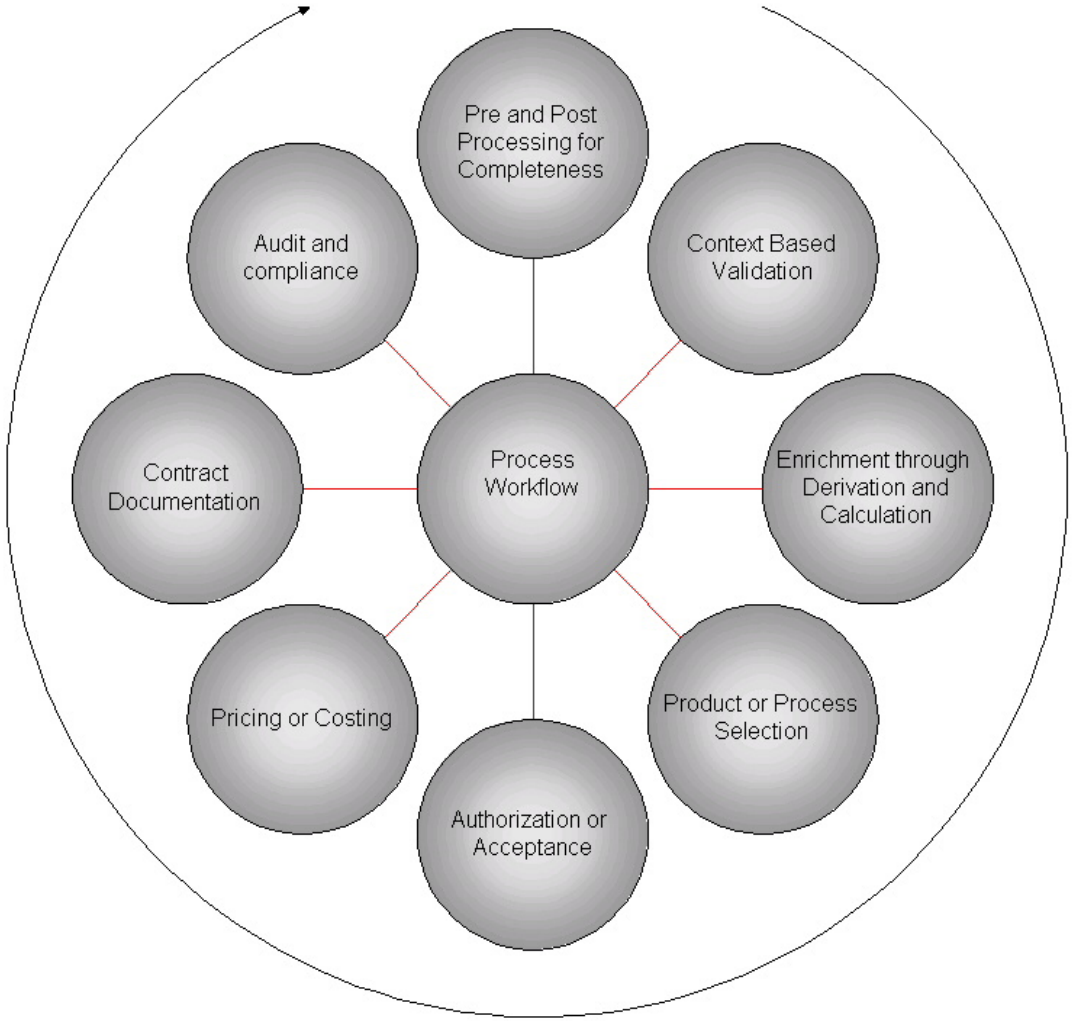


Typical applications for decision trees

The resulting decision trees can perform a wide range of useful macro level tasks . Here are some typical examples:

- Codifying and automating complex contracts such as financial instruments, insurance policies, health funding contracts
- Cleaning, consolidating, transforming data
- Controlling system behavior through dynamic configuration
- Replacing human decision makers within business transactions
  - And if we look at the typical example where decisions replace human actors, we can see a ‘pattern’ for high level decision decomposition that fits many transactions (next slide)

Common Decision Categories within a Transaction



Decisions should not constrain Architecture

- Decisions do not need to constrain architectural options.
- The now completely externalized decisions should be able to be ‘plugged in’ to any system – and quite possibly, more than one system on more than one platform. There is no need for a decision centric approach to constrain the preferred systems architecture.
- As an aside, most of IDIOM’s implementations are as local calls, operating in the address space of the calling application.

Decisions should not constrain Architecture

There are many integration options . . .

- Local call in the language of choice - Java, .NET, C++
- Web Services
- Messaging Service (MSMQ, MQSeries, Sonic)
- COM Wrapped DLL
- Proprietary containers such as:
  - IBM's business rule beans; Crossworlds container; MQSI Node
  - Sonic Software ESB containers
  - Microsoft BizTalk Server

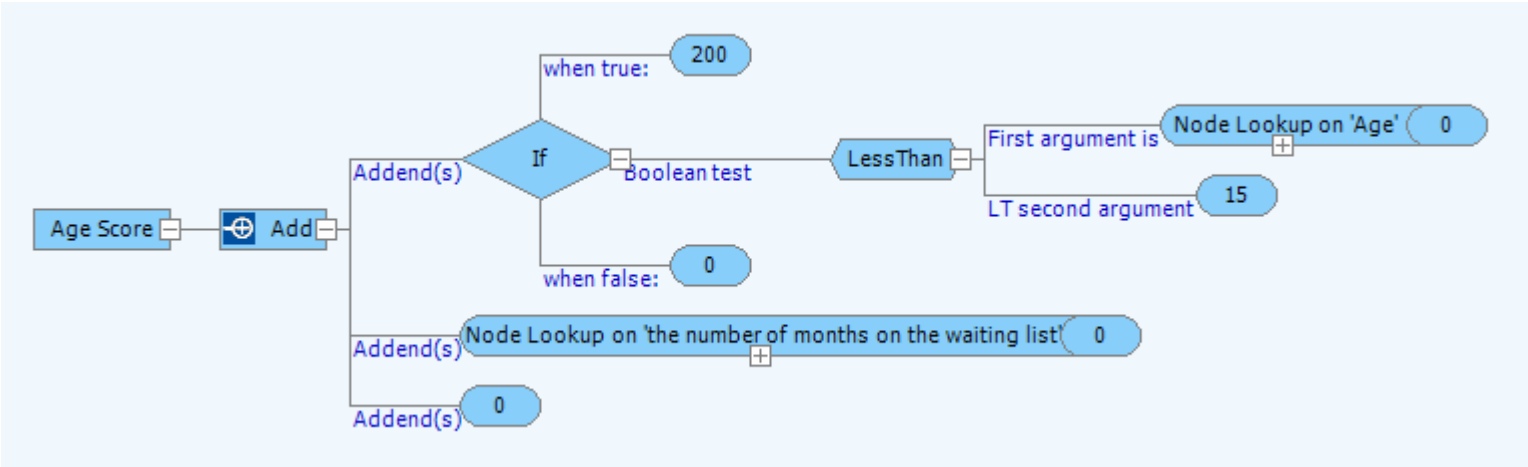
Healix Demonstration

We will now move on to a demonstration that I hope will relate to every slide we have shown. Everything that I have spoken about has been done and is being done. Let me show you now using a tool to demonstrate some of these decision characteristics in action:

Donor Recipient Matching System (National Kidney Allocation System)

- A simple system comprised only of rules and data handling adapters
- Specific decision structures are used to select the Recipient including
  - Social policy criteria
  - Clinical criteria
  - Criteria are scored to select the most suitable Recipient
  - Rules were built in a few hours
- Rules are maintained personally by the Medical Director, without technical or BA support

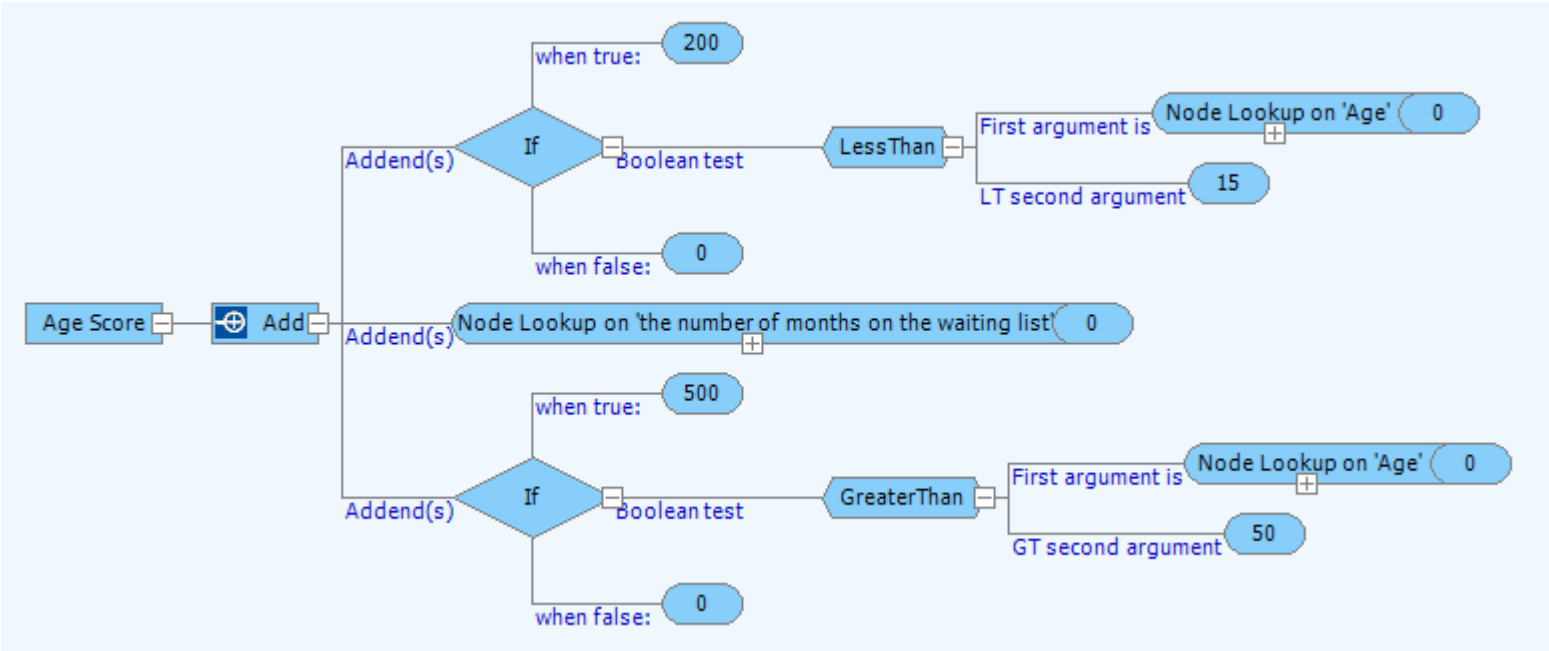
Formula for Age Score (original)



The formula "Age Score" is defined as follows.

Return A + the number of months on the waiting list + 0
- A: When Age < 15
  - THEN return 200
  - ELSE return 0

Formula for Age Score (updated)



The formula "Age Score" is defined as follows.

Return A + the number of months on the waiting list + B

A: When Age < 15
- THEN return 200
- ELSE return 0

B: When Age > 50
- THEN return 500
- ELSE return 0

Summary

- What we have seen
  - Decisions are THE first class citizens of the business systems community
  - Software developers now need to recognize and respond to this
- Why is it important to you?
  - Aligning software development to this reality improves many outcomes: “Greatly increased business agility - faster, at lower cost, and with less risk”
- How can you benefit?
  - Start thinking about businesses in terms of the decisions they make and how they make them – it is what makes them, and you, succeed or fail!