Automated Extraction of Business Rules and Models from Code

Ira Baxter, Ph.D., CEO/CTO
idbaxter@semanticdesigns.com

Friday, November 4, 2016 (11:30 am – 12:30 pm)
Room: Florentine III & IV
Example: Chemical Plants
Change: Migrate Process Control programs

- **Problem**: Trusted plant controller software running in end-of-life industrial control computers
  - Hundreds of different plants
  - Migrate to modern controllers
  - Recover abstract process control from “assembly code”

- **Solution**: (in progress)
  - Define abstractions in terms of dataflows with conditional implementations
  - DMS matches legacy code via dataflows (“Programmer’s Apprentice”)
  - Find consistent matching sets of abstractions
  - Reify abstractions to new controller code

Some plants now converted
1. Define *abstraction* and how that is related to business rules

2. Show how computer code *implements* abstractions

3. Show how information flow *and pattern matching* can support finding abstractions in code enabling the extraction of business rules
Who am I?

Ira D. Baxter, Ph.D.

• Research on
  • Software Reusability
  • Theory for formal program design capture and modification

• Founder/CEO/CTO of Semantic Designs (automated tools company)

• 46 years building software tools
  • Operating Systems and Compilers for minis and micros
  • Program Analysis and Transformation Engines (DMS®)
  • *Usable* Parallel Programming languages (PARLANSE)

• Architect/Project lead on Automated Solutions to tough problems
  • Synthesis of parallel supercomputing codes for seismic simulation
  • Code generation of programs to run automobile factories
  • Translation of legacy mission software for B-2 Stealth Bomber
  • Automated architectural re-engineering of Boeing aircraft mission software
  • Impact analysis tools for large-scale mainframes: ANZ and US Social Security
  • *Model* ("business rule") extraction for Dow Chemical factories
The first model of the world might not be the right model

What can I do if I believe this world model?

“… most substances are compounds of these …”

Doh!
Early Ideas on Elements

Robert Boyle stated...

- A substance was an element unless it could be broken down to two or more simpler substances.

Air therefore could not be an element because it could be broken down into many pure substances.
Why do we care about “Business Rules” at all?

• Our organizations are large, complex entities
  • We run them with complex processes largely composed of software
  • The software itself is made from accidental technologies *du juor / d’hui(!)*
  • It is giant, complex, chaotic, and hard to understand
  • … and we forgot what it all does

• How does management …
  • know what the organization is doing?
  • state what they want the organization to do?

• How can we change the software to do what management wants?

• **Answer:** *abstract what the “complex process” is doing*
  • Hope the abstraction gets rid of accidental details
  • Hope the abstraction uses vocabulary management understands
Abstraction? What on earth does that mean?

The essence of abstractions is *preserving information that is relevant* in a given context, and forgetting information that is irrelevant in that context.

– John V. Guttag (MIT Computer Science)

This means

• teasing out what is important for a specific audience purpose
• from one system of vocabulary
• and translating it into another system of vocabulary
• easier to understand for that specific audience

It also means:

leaving out the details that don’t matter.

Insight: Most of the way our software is implemented involves details that ultimately don’t matter;

*all we want is the effect!*

… isn’t it a bit weird that managers seem know this, and programmers don’t?
So… what is a “Business Rule”?

- Requirements document (written in natural language [English])
- Z notation
- Semantics of Business Vocabulary and Business Rules (SBVR)
- Business Process Modeling Language
- Business Process Execution Language
- Decision Tables
- Decision Model
- Unified Modeling Language
- Flowcharts
- Domain Specific Languages
- Drools rules
- Random vendor result…?

How can we agree on what a BR is if there are so many kinds?
So… what is a Business Rule?

- Requirements document (written in natural language [English])
- Z notation
- Semantics of Business Vocabulary and Business Rules (SBVR)
- Business Process Modeling Language
- Business Process Execution Language
- Decision Tables
- Decision Model
- Unified Modeling Language
- Flowcharts
- Domain Specific Languages
- Drools rules
- Random vendor result…?

How can we agree on what a BR is if there are so many kinds?

Answer: They each focus on extracting certain information, abstracting away other information. Each is unique!

Problem for users: what specific information do you want? Often chosen (poorly) for you by what a vendor happens to offer.
Key Background Concepts

(Business) State:
A set of facts true at a point in time about a (business) entity

State Model: The set of data describing a (business) entity
- Have purchase_orders
- Have warehouse_items
- Have unpaid_invoices
- Have available_cash

Data Model: The structure and meaning of the data: Vocabulary
- Structure == name and shape of the data
  (PO is tuple <customer,itemID>)
- Meaning is the set of operations on the data and their results
  Order(POs) → POs
    such that quantity (#) of POs increase
  Fulfill(POs,WIs,UIs) → <POs,WIs,UIs>
    such that #POs decreases, #WIs decreases, #UIs increase
  Payment(UIs,AC) → <UIs,AC>
    such that #UIs decreases, and amount AC increases
An algebraic (precise) specification (Spectrum) of business data

algebra Customer is String with
sorts: customer;
signatures:
    newCustomer(string) \rightarrow customer;
    name(customer) \rightarrow string;
axioms:
    name(newCustomer(string)) == string;
end

algebra ItemID is
sorts: itemID;
signatures: // discrete element algebra
axioms:
    end

algebra PurchaseOrder is Customer+ItemID+Integer with
sorts: purchaseOrder;
signatures:
    newOrder(Customer,ItemID,Natural) \rightarrow purchaseOrder;
axioms:
    orderingCustomer(newOrder(customer,itemID,integer)) == customer;
    orderedItemID(newOrder(customer,itemID,integer)) == itemID;
    orderedQuantity(newOrder(customer,itemID,integer)) == integer;
end

algebra WarehouseItem is ItemID+Integer with
sorts: warehouseItem;
signatures:
    quantityOnHand(itemID,natural) \rightarrow warehouseItem;
axioms:
    warehouseItemID(quantityOnHand(itemID,integer)) == itemID;
    quantityavailable(quantityOnHand(itemID,integer)) == integer;
end

Shape == Schema
Meaning == Constraints
On Operations
An algebraic specification (Spectrum) of *business state*

```
algebra BusinessState is Set<PurchaseOrder>, Set<WareHouseItems>,
    Set<UnpaidInvoice>, AvailableCash As Natural with
types: businessState

signatures:
  newCustomer(string) ➔ suffix(string)="Inc." ;
  newOrder(customer, itemID, desired) ➔ desired>0

currentState(Set<PurchaseOrder>, Set<WareHouseItems>,
    Set<UnpaidInvoice>, AvailableCash) ➔ businessState ;
purchaseOrders(businessState) ➔ Set<PurchaseOrder> ;
...  
businessStartUp() ➔ businessState ;
...

axioms:
purchaseOrders(currentState(POs,WIs,UIs,AC) = POs ;

businessStartUp() == currentState(empty,empty,empty,1000);  

currentState(newOrder(customer, itemID, quantity+Set<PurchaseOrder>,
    Set<WareHouseItems>,
    invoice(customer, anyprice)+Set<UnpaidInvoice>, cashonhand) ➔ false ;
  -- no new orders allowed until previous invoices are paid
```
An algebraic specification (Spectrum) of business actions (vocabulary)

algebra Company is BusinessState with

signatures:
order(businessState, customer, itemID, natural) ;
fulfill(businessState) \rightarrow businessState ;
collect(businessState, customer) \rightarrow businessState ;
restock(businessState, itemID, natural) \rightarrow businessState ;
...

axioms:

**fulfill** (currentState (newOrder (customer, itemID, desired) + Set<PurchaseOrder>,
quantityOnHand (itemID, available) + Set<WareHouseItems>,
Set<UnpaidInvoices>, cashonhand) ==
currentState (Set<PurchaseOrder>,
quantityOnHand (itemID, available-desired) + Set<WareHouseItems>,
invoice (customer, desired*price) + Set<UnpaidInvoices>,
cashonhand))
if available>=desired ;

collect (currentState (Set<PurchaseOrder>,
Set<WareHouseItems>,
Set<UnpaidInvoices>, cashonhand),
customer) ==
currentState (Set<PurchaseOrder>,
Set<WareHouseItems>,
Set<UnpaidInvoices>,
cashonhand+amount)) ;
...

**Meaning == Constraints On Operations**
Two Fundamental Flavors of Rules

Constraints: sets of conditions over (business state) which must always be true

Already have some these in basic vocabulary

• Data Model:
  The structure and meaning of the data (in a state) elements, and any constraints on those data elements

• State Model:
  The set of data describing an (business) entity including constraints on the state of the business

Can state business actions in term of pre- and post- action constraints
  Essence of “nonprocedural”

Procedures: reactions to new events to change business state in desired way

To be useful, these reactions must honor business constraints
Classic Flowchart (Procedural Rule)

1. Wait for Purchase Order
2. Invoice Valid?
   - No: Wait for Part in Stock
   - Yes: Wait for Invoice Payment
3. Update Cash
4. Mark Invoice Paid
5. Wait for Part Restock

End Flowchart

Order Part Restock
Classic Flowchart Simulation 1

Execution token

Wait for Purchase Order

Invoice Valid?

No

Wait for Part in Stock

Ship Part

Send Invoice

Order Part Restock

Wait for Invoice Payment

Update Cash

Mark Invoice Paid

Wait for Part Restock
Classic Flowchart Simulation 3

1. Wait for Purchase Order
2. Invoice Valid?
   - No
   - Wait for Part in Stock
   - Ship Part
   - Send Invoice
   - Order Part Restock
3. Wait for Invoice Payment
   - Update Cash
   - Mark Invoice Paid
   - Wait for Part Restock
Classic Flowchart Simulation 6

1. **Wait for Purchase Order**
   - **Invoice Valid?**
     - Yes: **Wait for Part in Stock**
       - **Ship Part**
         - **Send Invoice**
         - **Order Part Restock**
     - No: **Wait for Part in Stock**

2. **Wait for Invoice Payment**
   - **Update Cash**
     - **Mark Invoice Paid**
     - **Wait for Part Restock**
Classic Flowchart Simulation 7

1. Wait for Purchase Order
2. Invoice Valid?
   - No
   - Wait for Part in Stock
   - Ship Part
     - Send Invoice
     - Order Part Restock
   - Yes
   - Wait for Invoice Payment
     - Update Cash
     - Mark Invoice Paid
     - Wait for Part Restock
Send Invoice

Invoice

Valid?

Wait for Purchase Order

Invoice

Valid?

Wait for Part in Stock

Ship Part

Send Invoice

Order Part Restock

Wait for Invoice Payment

Update Cash

Mark Invoice Paid

Wait for Part Restock
So why not use FlowCharts?

Actions are in terms of... what? (“Order Part Restock”)

Better if uses abstract, well-defined business actions (fulfill)

Overconstrained Sequence of Events

Why Wait for Invoice Payment
then Wait for Part Restock?

We need more abstract event sequencing

Synchronization is not handled well

What does “wait for ...” mean?
Classic Petri Net 1
Synchronization with Multiple Tokens

Place: A holder of zero or more (black) tokens
Token: Marker in place representing state-is-active
Outgoing Arc: Connection from an (input) Place to Transition
Incoming Arc: Connection from Transition to (Output) Place
Transition: An intermediary between places
that consumes tokens from input places synchronously
and generates a token in output places

Parts in Warehouse
Purchase Order
Part Available for Purchase
Shipments
Invoices
Place: A holder of zero or more (black) tokens
Token: Marker in place representing state-is-active
Outgoing Arc: Connection from an (input) Place to Transition
Incoming Arc: Connection from Transition to (Output) Place
Transition: An intermediary between places that consumes tokens from input places synchronously and generates a token in output places
Classic Petri Net Simulation 3

Client Need

Parts in Warehouse

Purchase Order

Part Available for Purchase

Invoices

Restock Order

Restock Part Available

Shipments

Shipment Expected

Cash

New Part From Factory

Expected

Available
Classic Petri Net Simulation 5

- Client Need
- Parts in Warehouse
- Purchase Order
- Part Available for Purchase
- Invoices
- Shipments
- Shipment Expected
- Restock Order
- Restock Part Available
- New Part From Factory
- Cash
BPMN: A special kind of (classic) Petri Net

Company

Purchase Order

Wait for Part

Request Part from Warehouse

Wait for Payment

Send Invoice

Order Part Restock

Ship Part

Wait for Part

Send to Warehouse

Update Cash

Customer

Invoice

Part
**BPMN Simulation 3**

**Company**
- Purchase Order
- Wait for Part
- Request Part from Warehouse
- Ship Part
- Wait for Part
- Send Invoice
- Order Part Restock
- Wait for Restock
- Update Cash
- Send to Warehouse

**Customer**
- Invoice
- Part
BPMN Simulation 8

Company

Purchase Order

Wait for Part

Request Part from Warehouse

Ship Part

Send Invoice

Wait for Payment

Order Part Restock

Wait for Restock

Update Cash

Send to Warehouse

Customer

Invoice

Part
BPMN Simulation 9

Company

Purchase Order

Wait for Part

Request Part from Warehouse

Wait for Part

Ship Part

Send Invoice

Wait for Payment

Send to Warehouse

Update Cash

Order Part Restock

Wait for Restock

Customer

...

Invoice

Part
Weaknesses of BPMN?

**Actions are in terms of… what?**

(“Order Part Restock”)

Better if uses **abstract**, well-defined business actions (**fulfill**)

**Data being exchanged isn’t clearly defined**

Better if uses **abstract**, well-defined business data (**PurchaseOrder**)

**Synchronization often occurs with data transmission**

BPMN requires (active) event and (passive) data

Better to combine synchronization signals with data
“Colored” Petri Nets (Timed, Hierarchical, …)

Place: A holder of zero or more (colored) tokens
Token: Marker in place representing available event with data
Outgoing Arc: Connection from an (input) Place to Transition
Incoming Arc: Connection from Transition to (Output) Place
Transition: An intermediary between places that consumes tokens from input places synchronously and generates a token in output places
Computes new data values to new tokens using business actions

Parts in Warehouse → Part Available for Purchase → Shipments
Purchase Order → Part Available for Purchase
Invoices

Token color represents abstract business data (Can use name instead of color)
Colored Petri Nets Simulation step

Place: A holder of zero or more (colored) tokens
Token: Marker in place representing available event with data
Outgoing Arc: Connection from an (input) Place to Transition
Incoming Arc: Connection from Transition to (Output) Place
Transition: An intermediary between places that consumes tokens from input places synchronously and generates colored tokens in output places
Computes new data values for new tokens using business actions
Colored Petri Nets
Slight Notation Change

Place: A holder of zero or more *(colored)* tokens
Token: Marker in place representing *available event with data*
Outgoing Arc: Connection from an *(input)* Place to Transition
Incoming Arc: Connection from Transition to *(Output)* Place
Transition: An intermediary between places that consumes tokens from input places synchronously and generates colored tokens in output places
Computes new data values for new colored tokens *(business actions)*

Parts in Warehouse → Purchase Order

Compute invoice content

Part Available for Purchase → Invoices, Shipments

This style of computation is called *data flow*
So… what specific information do you want from Business Rules?

There are many possible uses of BR across an organization.

• Narrowing to one focus enables simpler BR, but limits use.
• Widening focus necessitates more complex BR vocabulary

This creates tension in choice.

So how can you choose?

Consult hierarchy of BR types.
Two hierarchies of program models: code and data

Computation

- Constraint Systems with Time
  - Timed, Colored Petri Nets
    - Petri Nets (BPMN)
      - Flowcharts/Procedures
      - Decision Tables
        - IF-THEN rules
      - Data Flow Diagram
    - Call Graphs

Data

- Algebraic Specifications
  - Entity Relationship Diagrams
    - Relational Data Models
      - Fixed Set of Data types
      - Record Structures
    - Class Diagrams
      - Typed Data Blobs
      - Named data blobs
Model and Business Rule Extraction Using Pattern Matching
Model extraction using Pattern Matching

Goal:

*Extract models and business rules from legacy systems*

Method:

1. BA/Programmer identify code idioms representing business actions
2. BA/Programmer define a pattern representing idiom
3. BA/Programmer may define other patterns for same idiom
4. Tool analyze source code to find data flows
5. Tool analyze pattern to find data flows
6. Tool matches pattern to source code using data flows as guide
7. Tool records idiom name as code abstraction

Benefits:

*Matched code fragments are instances of business actions*
*Code variations equivalent to pattern are found*

Not discussed: *How to build business vocabulary or data models*
int fibonacci(n)
{ unsigned int fl= 0, fh = 1, i;
  if (n <=1 )
    fh = n;
  else
    for (i= 2; i<=n; i++) {
      int tmp = fh;
      fh =fl + fh;
      fl = tmp;
    }
  print ("Fib(%d) = %d\n", n, fh);
  return n;
}

Big example wouldn’t fit on football field…

Insight:
Maybe we can abstract away this detail

accumulate(fib#s)
(Analysis): Data Flow patterns:
Matching code with dataflows, *not syntax*

---

**default base domain C~ISO9899c1990.**

**public data flow pattern**

```java
classify_bank(bank_number:IDENTIFIER<~, bank_code:IDENTIFIER~>):statement_seq
= "if (\bank_number > 10 & \bank_number <= 25) \bank_code = 3; // bank of ethel else \bank_code = 0; // unknown bank number ".
```

---

**Data flow pattern for idiom**

---

**Data flow match of idiom woven into code**

---

"bank classification" idiom
Representing a business computation
COBOL tax computation Patterns

data flow pattern ComputeTax(TaxRate:Constant, Total:IDENTIFIER) :
  <HowTaxed:  TaxStyle>:
  StatementSequence
  case HowTaxed: TaxStyle:
    StatementSequence
    when 'Added'
      ComputeTax_by_adding(TaxRate,Total)
    when 'Multiplied'
      ComputeTax_by_multiplying(TaxRate,Total)
  esac;

data flow pattern ComputeTax_by_adding(TaxRate:Constant, Total:IDENTIFIER) :
  StatementSequence
  Temp:IDENTIFIER
  "MULTIPLY \Total BY \TaxRate GIVING \Temp."
  ADD \Temp TO \Total"
  if Value(TaxRate)>0.0 and Value(TaxRate)<1.0

data flow pattern ComputeTax_by_multiplying(TaxRate:Constant, Total:IDENTIFIER) :
  StatementSequence
  "Compute \Total = 1.0 + \TaxRate"
  if Value(TaxRate)>0.0 and Value(TaxRate)<1.0;

COMPUTE-TOTAL.
MULTIPLY QUANTITY BY PRICE GIVING TOTAL-AMOUNT.
IF TOTAL-AMOUNT > DISCOUNT-THRESHOLD
  MULTIPLY TOTAL-AMOUNT BY DISCOUNT-PERCENT
  GIVING DISCOUNT-AMOUNT
  DIVIDE 100 INTO DISCOUNT-AMOUNT
  SUBTRACT DISCOUNT-AMOUNT FROM TOTAL-AMOUNT.
  ADD ONE TO VAT-RATE GIVING TAX-ADJUSTMENT.
  MULTIPLY TAX-ADJUSTMENT INTO TOTAL-AMOUNT.
  DISPLAY COMPANY-NAME.
  DISPLAY "Total: ", TOTAL-AMOUNT.

COMPUTE-INVOICE.
MULTIPLY AMOUNT BY VAT-RATE GIVING TAX.
Compute INSURANCE = INSURANCE_RATE * AMOUNT.
ADD TAX TO AMOUNT.
ADD INSURANCE TO AMOUNT GIVING INVOICE_TOTAL.
Model Based Migration

(Dow Chemical)

AS-IS

Analysis

Guidance

DCS Model: Process Control Concepts applied to specific factory

Rules from Model to ST

Translation (Abstraction)

Rule Compiler

Describe RLL

Description of RLL

Description of Model

Description of Model

Description of ST

Translation Rules from RLL to DCS concepts

DMS

Parse

Analyze

Transform

Format

if(ST4)
then Timer(T42,4sec);
if(ST2)
then Timer(t41,4sec);
ST1X :=
(ST1 ! ST4 & T42.dn)
& (~ST1 ! ~ S1 ~ S2)
& first_scan );
ST2X :=
(ST2 ! ST1 & S1 & ~S2 )
& (~ST2 ! T41.DN )
& STX :=
(ST3 ! ST2 & T41.DN) & (~ST3 ! S1 ! ~S2 )
ST4X :=
(ST4 ! ST3 & ~S1 & S2 ) & (~ST4 | T42.DN );

Temp

Filter

Safe?

Control

Heat

Modern Controller Code

(ST)
Summary

Wide variety of “business rule” schemes are unified by Colored Petri Net models augmented by formal data models

Two major components:
- Data and business rule computation models
- Process/data flow with synchronization

Differs from BPMN:
- How it models data transfers as part of synchronization
- How it compute business actions based on formal data model

Key to capturing business rules from code
- By recognizing dataflow idioms as business abstractions
- Major success in migrating must-not-fail factory
- Technology at early-adopter stage

www.semanticdesigns.com
Automated Extraction of Business Rules and Models from Code

Abstract

Everybody talks about "business rules", and how to extract them from code. The definitions vary wildly, and the procedure to extract them are largely informal. This confuses everybody about the nature of business rules and what exactly happens as they are extracted.

This talk will be a synthesis of ideas from the program analysis, reverse engineering, model extraction and business rules extraction community. We will discuss the concept of abstraction as a unifying principle that ties these ideas together in a coherent framework, showing how decision tables, BPEL-style notations, models and domain-specific languages are all variations on a theme.

We will discuss the kind of technology that is required to enable the analysis of code by tracing information flows from system inputs through code to system outputs, and reverse engineering from code idioms with interactive guidance of the process by a business rule analyst. As a case study, we will discuss how we were able to extract reliable models of a factory control process from extremely low level code for Dow Chemical industrial plant controllers using pattern matching technology to recognize common code idioms and design choices. Finally, we will discuss how this technology is likely to evolve for use in the broader business rule community.

<table>
<thead>
<tr>
<th>Learning Objective 1 *</th>
<th>Learning Objective 2 *</th>
<th>Learning Objective 3 *</th>
</tr>
</thead>
<tbody>
<tr>
<td>What are abstractions, and how are they related to business rules.</td>
<td>Show how computer code implements abstractions</td>
<td>Show how information flow and pattern matching can be used to support extraction of business rules and abstraction</td>
</tr>
</tbody>
</table>
## Abstract

<table>
<thead>
<tr>
<th>Presentation Level</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Advanced</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kind of Presentation</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1-hour presentation</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Select a General Focus</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>? The Evolving Analysis and Design Landscape</td>
<td></td>
</tr>
<tr>
<td>? Leveraging Technology</td>
<td></td>
</tr>
<tr>
<td>? Fast Forward</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Select a Special Focus</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>? Business Rules and Decisions</td>
<td></td>
</tr>
</tbody>
</table>

### Additional focus or positioning of content

<table>
<thead>
<tr>
<th>Keyword 1</th>
<th>Keyword 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>abstraction</td>
<td>information flows</td>
</tr>
<tr>
<td>design recovery</td>
<td>automation</td>
</tr>
<tr>
<td>pattern matching</td>
<td></td>
</tr>
</tbody>
</table>

### How do you plan on conducting the session?

Traditional presentation
Ira Baxter, Ph.D., has been building system software since 1969. After founding a microprocessor software house in the 1970s, he returned to graduate school at UC Irvine to study reuse of knowledge supporting software maintenance and evolution. On completing his Ph.D. in 1990, he joined Schlumberger as research scientist automating the generation of supercomputer programs for oil field exploration. In 1995, he founded Semantic Designs, where he has been architect/implementer of the Design Maintenance System(R), providing automated program analysis and transformation to large-scale legacy systems.

He has been project lead on a variety of massive code migration and re-architecting projects, including work with Dow Chemical to automate the extraction of models from factory control code. Dr. Baxter has been active in Software Engineering and Maintenance and other conferences since 1983, including co-chairing of the International Conference on Software Maintenance.
Thank You

Contact
Randal Matthias
Business Development Manager
Rmatthias@semanticdesigns.com
1-512-250-1018 x172
www.SemanticDesigns.com