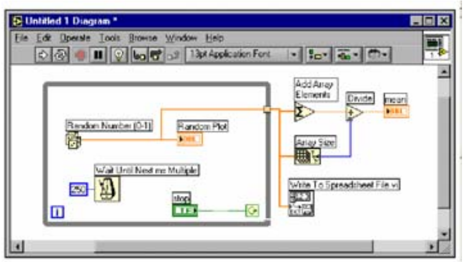
# Domain Specific VLs

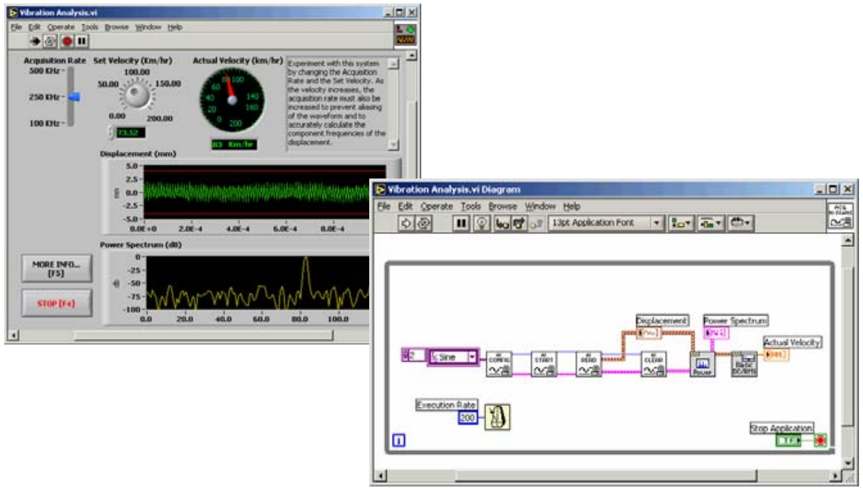
- A domain specific visual language is one where the notation is customised for a particular problem domain
- Have trade off between generality of language (ie range of problems able to be solved) and terseness of notation and closeness of mapping
- Look at:
  - A couple of widely used DSVLs
  - One that is likely to be widely used
  - Four locally developed DSVLs

# LabView

- LabView uses a visual dataflow metaphor like Prograph, but is a domain specific language rather than a GP one
  - Domain is lab instrumentation: access and analysis of sensor data attached to computer
  - Processing elements include math data transformations (eg FFTs, integrators, differentiators)
- Very successful commercial Domain Specific VL http://www.ni.com/labview/



# Labview example



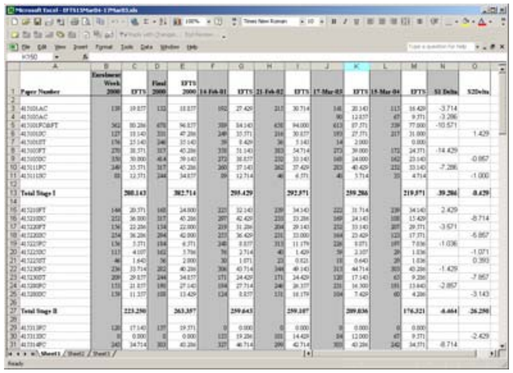
Experiment with this system by changing the Acquisition Rate and the Set Velocity. As the velocity increases, the acquisition rate must also be increased to prevent aliasing of the waveform and to accurately calculate the component frequencies of the displacement.

# Labview Success

- Metaphor used – dataflow wiring plus computation blocks – has high closeness of mapping
  - End users are electronic engineers – very familiar with circuit wiring
- Modularity via blocks – again very similar to electrical circuit concepts hence low abstraction gradient for end users and hidden dependencies are of a sort that end users are familiar with
- Problems of high viscosity due to layout reorganisation not an issue with user audience – familiar with these problems from circuit design tools
- Language relatively terse at one level (general concepts) but quite diffuse at another (many predefined operations with their own iconic representation)
- Attention to front end – ability to create realistic looking virtual instrument front panel

# Spreadsheets

- Very successful DSVL – so successful spreadsheets have become a more general tool
- Original target – financial and other numeric calculations
- Metaphor – financial tables + calculator

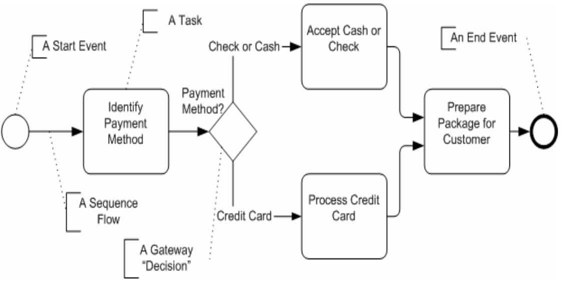


# Spreadsheet success

- Strong and consistent metaphor providing high closeness of mapping to typical balance sheet etc problems
- At one level notation is quite terse (sheet and cell metaphor), at another it is quite verbose (extensive range of functions that stretch the bounds of the matephor)
- Progressive evaluation well supported: values calculated immediately a formula entered
- Hidden dependencies a real issue – a strong cause of errors, ie leading to error proneness

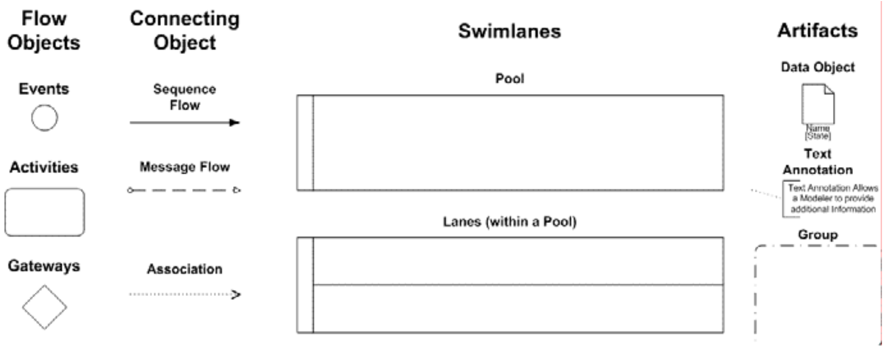
# BPNM

- Business Process Modelling Notation
- Provides notation for specifying business processes
- Understandable by:
  - Business users
  - Business Analysts
  - Technical Developers
- Can generate BPEL4WS executable from BPMN specifications
- Developed by BPMI consortium http://www.bpmn.org/

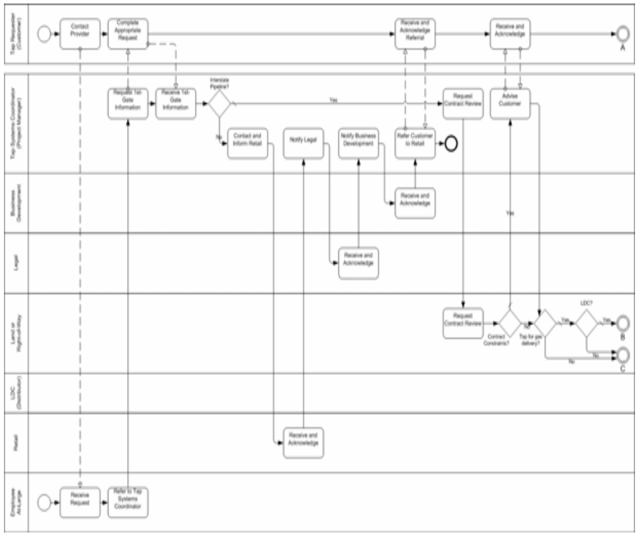


# BPNM

- Simple notation



# BPNM complex example



# BPNM successes

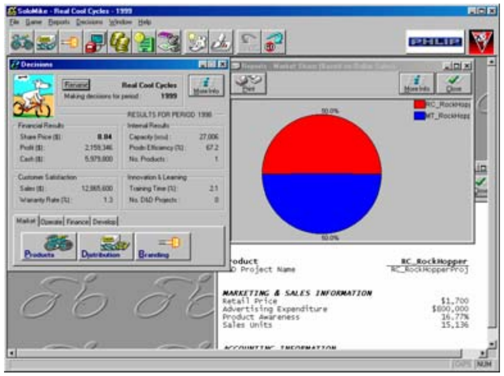
- Small number of concepts leads to a terse notation with small number abstractions. Good use of containers 9eg pools, lanes) to provide additional semantics without overloading user.
- Closeness of mapping high as core notation familiar to end users and business analysts (specific design choice).
- Good visibility and few hidden dependencies. Subprocesses can be "in lined" if desired.
- Low progressive evaluation as a design notation, but potential for runtime visualization reusing design diagrams

# Local Work

- SmartSims Builder (business modeller)
  - Model business interactions for business simulation game
- Orion Mapper/Rhapsody (message mapping)
  - Mapping between different semi-structured data schema
- Form based mapper (business form mapping)
  - Mapping between business forms, eg invoices
- SDL (statistical survey specfn)
  - Specifying various stages in the design of a statistical survey

# Mikes Bikes (SmartSims)

- Simulates bicycle business
- Factors in R&D, marketing, production costs, etc
- Play against robot or other teams
- Not generic: business model hard coded
- Need for matching business modelling tool to design new business situations
- MSc Thesis by See Wong



Real Cool Cycles
Making decisions for period 1999

RESULTS FOR PERIOD 1998

Financial Results
Share Price ($): 8.84
Profit ($): 2,159,346
Cash ($): 5,979,800

Internal Results
Capacity (units): 27,006
Prodn Efficiency (%): 67.2
No. Products: 1

Customer Satisfaction
Sales ($): 12,865,600
Warranty Rate (%): 1.3

Innovation & Learning
Training Time (%): 2.1
No. D&D Projects: 0

roduct RC_RockHopper
D Project Name RC_RockHopperProj

MARKETING & SALES INFORMATION
Retail Price $1,700
Advertising Expenditure $800,000
Product Awareness 16.77%
Sales Units 15,136

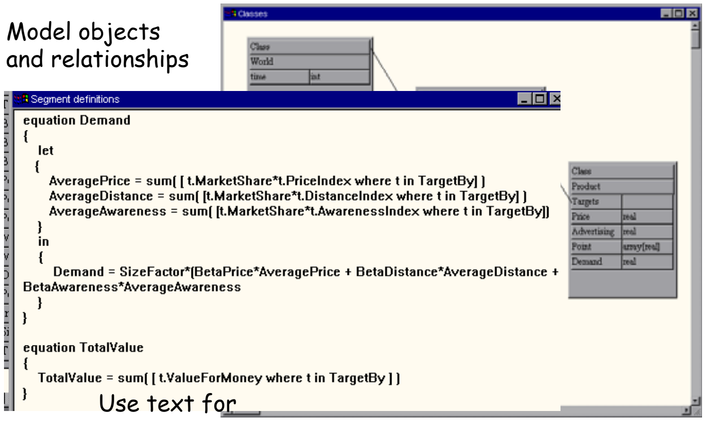
ACCOUNTING INFORMATION

# SmartSims

- Ideally would like business modellers to be able to construct new models
- Thus need a notation that has closeness of mapping for them
  - Familiar with entity relationship type modelling
  - Familiar with expression relationships using equations
- Thus combine the two using a UML like formalism
  - Something like a class diagram to represent business object types and their gross relationships (low abstraction gradient)
  - Reuse these with instantiated values to visualise enacted model (progressive evaluation/concreteness support)
  - Equational formulae to represent detailed relationships (combination of terse and verbose similar to spreadsheets)
  - GUI builder to construct UI of final simulation (cf Labview)

# Builder environment

Model objects and relationships



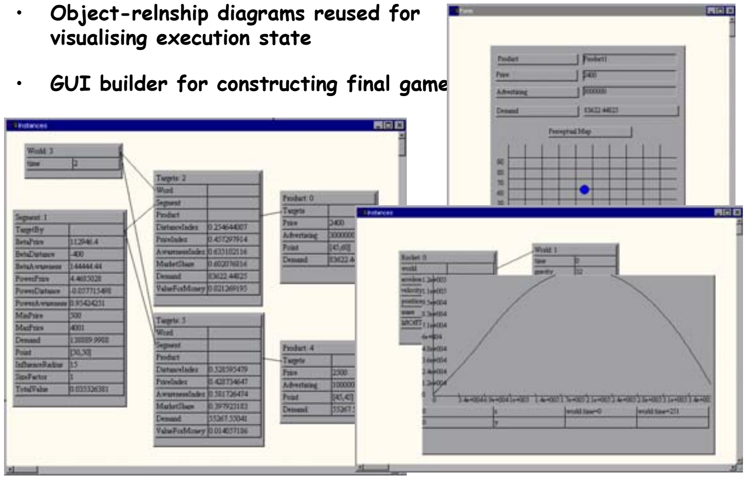
```
equation Demand
{
   let
  {
     AveragePrice = sum( [ t.MarketShare*t.PriceIndex where t in TargetBy] )
     AverageDistance = sum( [t.MarketShare*t.DistanceIndex where t in TargetBy] )
     AverageAwareness = sum( [t.MarketShare*t.AwarenessIndex where t in TargetBy])
   }
   in
   {
      Demand = SizeFactor*(BetaPrice*AveragePrice + BetaDistance*AverageDistance +
BetaAwareness*AverageAwareness
   }
}

equation TotalValue
{
   TotalValue = sum( [ t.ValueForMoney where t in TargetBy ] )
}
```

Use text for expressing constraint equations

# Run time visualisation

- Object-relnship diagrams reused for visualising execution state
- GUI builder for constructing final game



# Encore - Message Mapping (Orion Systems Ltd)

- Basic problem: want to map semi-structured data (health messages) from one schema to another.
  - Translating health information messages from one standard format to another involves laborious, time intensive, error-prone programming. Much of the code required is repetitive and hence lends itself to high level tool support with code generation
  - Many message standards plus XML-based variants
- Typical current approach
  - C++ program, 10's of pages of code
  - Boring and tedious work - very error prone

# Encore - Message Mapping

**Goals:**

**Design notations and implement a proof of concept tool to support complex hierarchical message translation (target is health sector, but wider application eg XML – XML translation is obvious)**

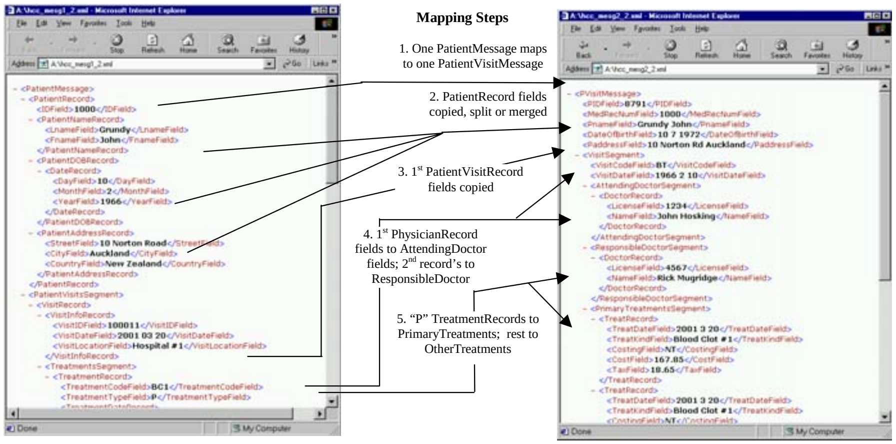
**User is experienced data modeller**
**One component of a more complete approach to this problem (includes schema specn, message field formatting, DOM support, etc)**

**Solution**

> **Use a visual language to specify mappings between elements in hierarchical schemas (familiar notation for data modellers -> Good closeness of mapping)**
> **Supplement with a textual equational language for specifying individual element mappings (reuse terse + verbose of spreadsheets)**
> **Mappings can be uni- or bi-directional**
> **Compile to a threaded interpreter to execute mappings**

# Example of messages that need mapping

Fields, records need splitting, combining
Hierarchies added, flattened etc



```
A:\hcc_mesg1_2.xml - Microsoft Internet Explorer
- <PatientMessage>
  - <PatientRecord>
      <IDField>1000</IDField>
    - <PatientNameRecord>
        <LnameField>Grundy</LnameField>
        <FnameField>John</FnameField>
      </PatientNameRecord>
    - <PatientDOBRecord>
      - <DateRecord>
          <DayField>10</DayField>
          <MonthField>2</MonthField>
          <YearField>1966</YearField>
        </DateRecord>
      </PatientDOBRecord>
    - <PatientAddressRecord>
        <StreetField>10 Norton Road</StreetField>
        <CityField>Auckland</CityField>
        <CountryField>New Zealand</CountryField>
      </PatientAddressRecord>
    </PatientRecord>
  - <PatientVisitsSegment>
    - <VisitRecord>
      - <VisitInfoRecord>
          <VisitIDField>100011</VisitIDField>
          <VisitDateField>2001 03 20</VisitDateField>
          <VisitLocationField>Hospital #1</VisitLocationField>
        </VisitInfoRecord>
      - <TreatmentsSegment>
        - <TreatmentRecord>
            <TreatmentCodeField>BC1</TreatmentCodeField>
            <TreatmentTypeField>P</TreatmentTypeField>
```

Mapping Steps

1. One PatientMessage maps to one PatientVisitMessage

2. PatientRecord fields copied, split or merged

3. 1st PatientVisitRecord fields copied

4. 1st PhysicianRecord fields to AttendingDoctor fields; 2nd record's to ResponsibleDoctor

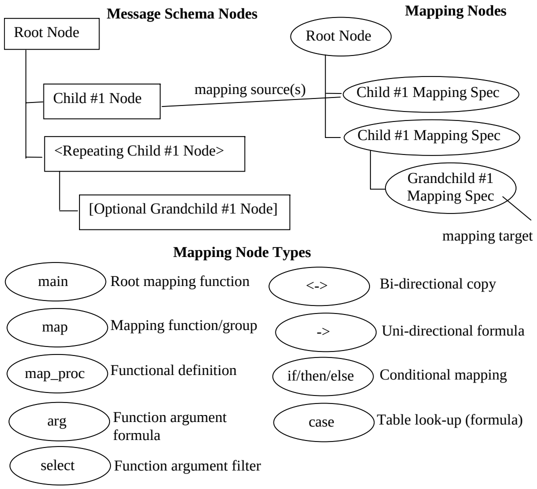
5. "P" TreatmentRecords to PrimaryTreatments; rest to OtherTreatments

```
A:\hcc_mesg2_2.xml - Microsoft Internet Explorer
- <PVisitMessage>
    <PIDField>8791</PIDField>
    <MedRecNumField>1000</MedRecNumField>
    <PnameField>Grundy John</PnameField>
    <DateOfBirthField>10 7 1972</DateOfBirthField>
    <PaddressField>10 Norton Rd Auckland</PaddressField>
  - <VisitSegment>
      <VisitCodeField>BT</VisitCodeField>
      <VisitDateField>1966 2 10</VisitDateField>
    - <AttendingDoctorSegment>
      - <DoctorRecord>
          <LicenseField>1234</LicenseField>
          <NameField>John Hosking</NameField>
        </DoctorRecord>
      </AttendingDoctorSegment>
    - <ResponsibleDoctorSegment>
      - <DoctorRecord>
          <LicenseField>4567</LicenseField>
          <NameField>Rick Mugridge</NameField>
        </DoctorRecord>
      </ResponsibleDoctorSegment>
    - <PrimaryTreatmentsSegment>
      - <TreatRecord>
          <TreatDateField>2001 3 20</TreatDateField>
          <TreatKindField>Blood Clot #1</TreatKindField>
          <CostingField>NT</CostingField>
          <CostField>167.85</CostField>
          <TaxField>18.65</TaxField>
        </TreatRecord>
      - <TreatRecord>
          <TreatDateField>2001 3 20</TreatDateField>
          <TreatKindField>Blood Clot #1</TreatKindField>
          <CostingField>NT</CostingField>
```

## Basic visual notation for Mapper

Represent each schema vertically, with lines and indentation depicting hierarchy

Mappings are functions between schema nodes
- simple copy
- translation formula
- conditional
- separate function with args
- table lookup
- etc



Message Schema Nodes

Mapping Nodes

Root Node

Child #1 Node

<Repeating Child #1 Node>

[Optional Grandchild #1 Node]

mapping source(s)

Root Node

Child #1 Mapping Spec

Child #1 Mapping Spec

Grandchild #1 Mapping Spec

mapping target

Mapping Node Types

main – Root mapping function

map – Mapping function/group

map_proc – Functional definition

arg – Function argument formula

select – Function argument filter

<-> – Bi-directional copy

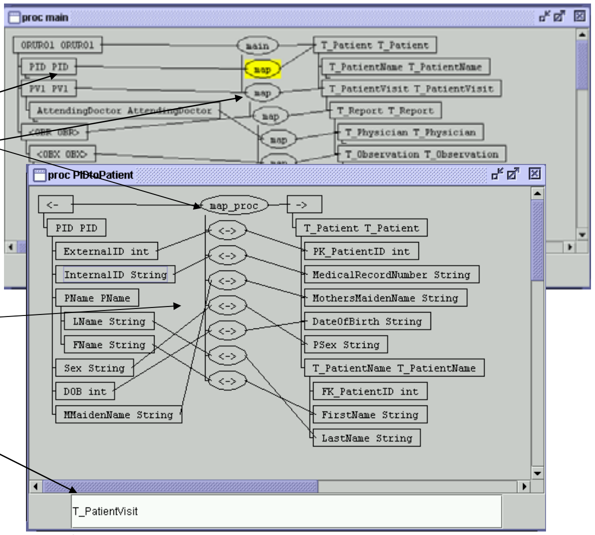
-> – Uni-directional formula

if/then/else – Conditional mapping

case – Table look-up (formula)

## Proof of concept tool

- **Mapping between hierarchies**
- **Proc abstraction map calls map procedure to map between patient name and PID records**
- **Equivalence mapping**
- **Textual formula box for more complex formula based mappings**



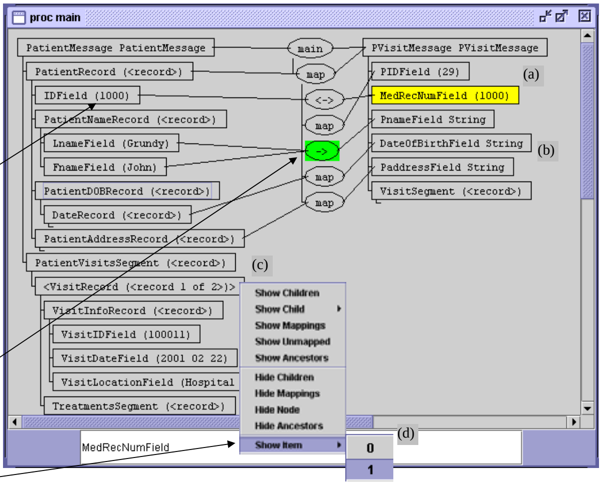
T_PatientVisit

## Mapping visualisation

Mapping diagrams reused to visualise execution (concreteness)

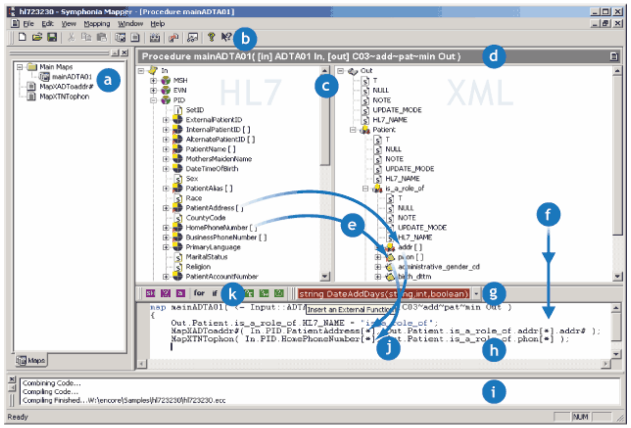
Hierarchies populated with values

Step through mapping execution and through collection values



Implemented using JViews framework

## Commercial product: Rhapsody



```
map mainADTA01( <- Input::ADTA                C03~add~pat~min Out )
{
    Out.Patient.is_a_role_of.HL7_NAME = "is_a_role_of";
    MapXADToaddr#( In.PID.PatientAddress[*], Out.Patient.is_a_role_of.addr[*].addr# );
    MapXTNTophon( In.PID.HomePhoneNumber[*], Out.Patient.is_a_role_of.phon[*] );
```

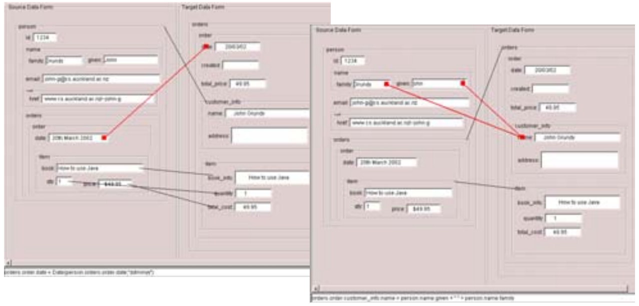
```
Combining Code...
Compiling Code...
Compiling Finished...W:\encore\Samples\hl723230\hl723230.ecc
```

ORION SYSTEMS

http://www.orion.co.nz/

## Form-based Mapper

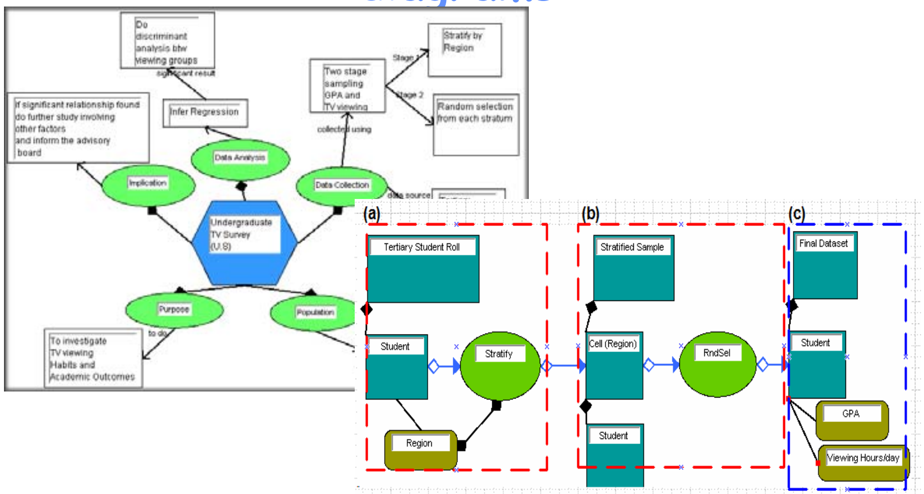
- An alternative approach to specifying mapping
  - Project by Yongqiang Li
- Aim to be used by a business modeller rather than a DBA
- Uses form metaphor rather than hierarchical tree



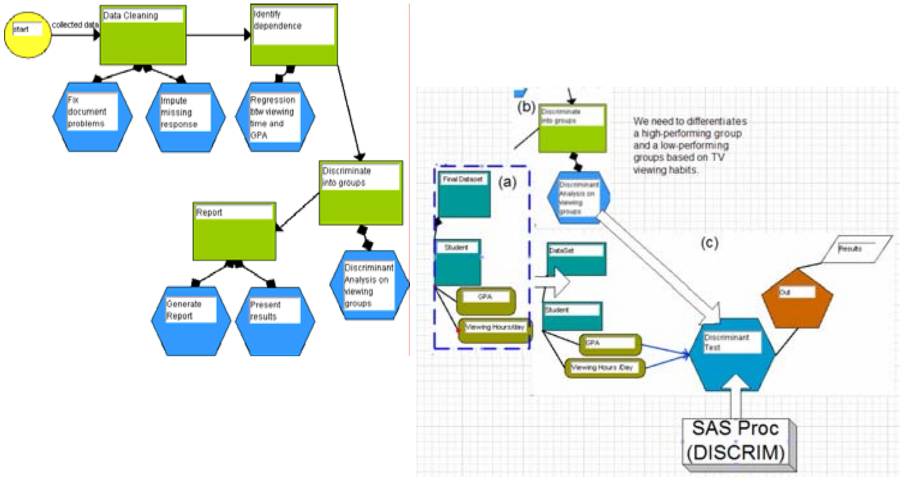
## SDL

- Environment for specifying statistical surveys
- To be used by survey designers (not nec prof statisticians)
  - Tend to have minimal programming experience
- Solution
  - Provide multiple notations with consistency maintenance
    - Each notation has small number of concepts ie terse
  - Survey diagrams for brainstorming/overview
  - Survey data diagrams for survey source data structures and their manipulation
  - Survey analysis diagrams defining statistical processes/techniques and flow of control in data analysis phase
  - Survey techniques diagrams for specifying behaviour of individual statistical techniques
- Prototyped using Pounamu (originated as a 732 assignment!)

# Survey and survey data diagrams


Do discriminant analysis btw viewing groups
significant result
If significant relationship found do further study involving other factors and inform the advisory board
Infer Regression
Two stage sampling GPA and TV viewing
Stage 1
Stratify by Region
Stage 2
Random selection from each stratum
collected using
Data Analysis
Implication
Data Collection
data source
Undergraduate TV Survey (U.S)
Purpose
Population
to do
To investigate TV viewing Habits and Academic Outcomes
(a)
(b)
(c)
Tertiary Student Roll
Student
Stratify
Region
Stratified Sample
Cell (Region)
RndSel
Student
Final Dataset
Student
GPA
Viewing Hours/day

# Survey analysis and technique diagrams



We need to differentiates a high-performing group and a low-performing groups based on TV viewing habits.

# Summary

- Have looked at several Domain Specific VLs
- Emphasis in each case was to find a notation or set of notations that is in some senses natural for the end user (ie closeness of mapping rated highly as a cognitive dimension)
  - Function blocks + wiring for Labview
  - Table + calculator for spreadsheet
  - Object relnship diagrams + equational constraints for Builder
  - Tree or Form + drag and connect + formula for Encore/Mapper
  - Variety of metaphors for SDL
- Also emphasis on reusing diagrams at execution time to visualise behaviour at the same level of abstraction used to construct the program (moving towards liveness/progressive evaln and concreteness but recognising that compile cycle inevitable in many applications)
- Common to use terse high level abstractions and more verbose lower level detail (often textual) which gives some hidden dependencies and can lead to error proneness (cf spreasdsheets)