COMPSCI 732 Software Tools
Course introduction

- **Aims of the Course**
  - This paper is concerned with advanced topics in tools that assist in the production of software, with a strong emphasis on practical aspects
- This is very much a research oriented paper – we will be talking but our active research interests and their application
- Lecturers:
  - John Hosking room 303.487
  - Karen Li room 303.495
  - Christof Lutteroth (supervisor) room 303.494

Course outline

- John and Karen's part (6 weeks):
  - Visual language design, application and evaluation (John) 1 week
  - Meta tools and visual language implementation(Karen) 2 weeks
  - Collaboration and knowledge management tools (John) 3 weeks
- Christof's part (6 weeks)
  - data access layers
  - version control
  - compiler generators
  - static analysis and type systems

Assessment

- There are two assignments and a final exam. The mark breakdown is
  - Two assignments 25% each
  - Final Exam 50%
- You must gain a pass in each of the assignment component and examination component to pass the course as a whole.

Assignment topics

- Visual language implementation
  - This assignment will involve the construction of a small visual language environment (or part of an environment) using our Marama meta tool which is Java based or Microsoft's DSL Tools meta tool.
- Tool project
  - This assignment will involve the development of a small software engineering tool .

Lectures and labs/tutorials

- Monday 1pm and Tuesday 1pm slots will mostly be used for lectures
  - Held in 303.279
- Thursday 1pm slot will mostly be used for lab demonstrations or tutorials/group exercises
  - Lab demos/tuts will be in GCL lab (303S-G91)
  - Some tuts may use 279
  - We'll advise locn each week

Readings

- We will be assigning regular readings
  - Typically research papers
- You will be expected to read these BEFORE the lectures
- You will be expected to contribute to the lectures using insights gained from the readings
  - Class discussions
  - Group activities
- Building understanding of how to read and critique research papers is an important skill for a PG student

Some context: software tools

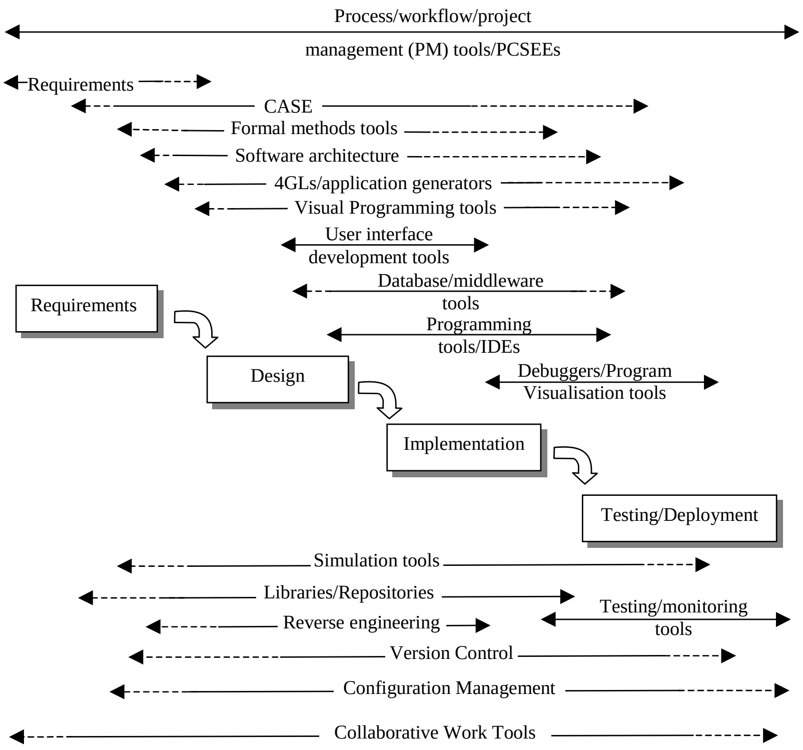
- Tools to support the development of software
  - Covers all aspects of the software development lifecycle
  - Covers support for a wide variety of methodologies and technologies
    - Both general purpose and domain specific
- Much research and commercial activity in this area
- Strong research focus in the CS Department at Auckland

- Resource: Software Tools, Grundy and Hosking (Chapter in Wiley Encyclopaedia of Software Engineering)

Context: software tools

- Rapid change in software development practice in recent times:
  - Newer development methodologies, eg RAD, XP/Agile development, Open Source development, that focus on iterative & collaborative development
    - Need for round trip engineering support
    - Need for collaboration support
  - New technologies to support, partic wrt distributed systems (eg middleware, component based approaches, web services, aspects)
    - Need new modelling and support tools

Some context: software tools



Process/workflow/project management (PM) tools/PCSEEs

Requirements

CASE

Formal methods tools

Software architecture

4GLs/application generators

Visual Programming tools

User interface development tools

Database/middleware tools

Programming tools/IDEs

Debuggers/Program Visualisation tools

Requirements

Design

Implementation

Testing/Deployment

Simulation tools

Libraries/Repositories

Testing/monitoring tools

Reverse engineering

Version Control

Configuration Management

Collaborative Work Tools

Five issues for software tool design

- In pairs come up with five general issues for software tool design 5 mins
  - Things you need to be aware of as a tool designer
  - Things that strongly influence the way in which you would approach a tool design task
- In pairs of pairs exchange and discuss your lists and come up with 1-2 top issues 2-3 mins

Visual languages

- “some visual representations (in addition to or in place of words and numbers) to accomplish what would otherwise have to be written in a traditional one-dimensional programming language”
  - Shu, N *Visual Programming*, Van Nostrand Reinhold, NY, 1988
- Visual programming is programming in which more than one dimension is used to convey semantics. Eg:
  - multi-dimensional objects
  - use of spatial relationships
  - use of the time dimension to specify “before-after” semantic relationships.
- A Visual Programming Environment allows visual specification and generation of code
- NB some use of 2-D in conventional PLs
  - use of indentation/layout to convey semantic info

Why visual languages?

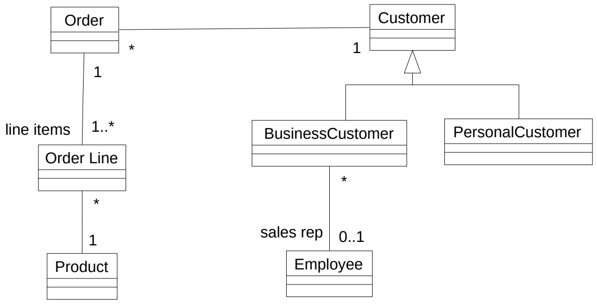
- Make good use of human cognitive capabilities
  - A picture is worth a thousand words
- Arise naturally in many design situations
- Often allow a “higher level” approach to design
- Often useful for complex configuration tasks
  - Evolving frameworks pattern language
- Much research on VLs at UoA ☺

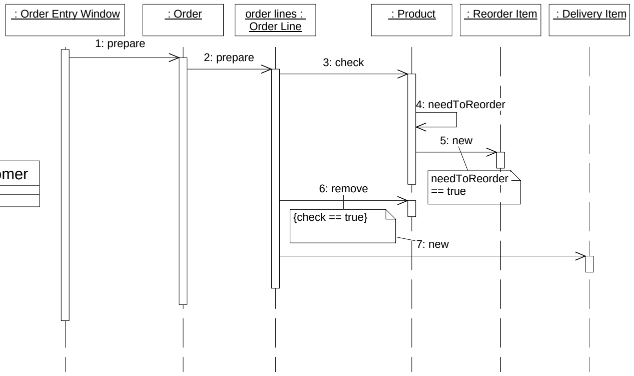
History

- Early work didn't scale
  - – Executable flowcharts
  - – Programming by demonstration
- Followed by work in
  - – Programming environments that replaced some textual programming by visual (eg VisualWorks, Visual Basic)
    - Won't consider here
  - – CASE tools – programming in the large
  - – General purpose VLs – the original nirvana
  - – Domain Specific VLs – constraining the task

Example Visual Languages: UML

- UML is a collection of visual notations used for programming in the large
- Originally purely a design language but MDA/MDE approaches ar changing that





Example Visual Languages: Circuit diagrams

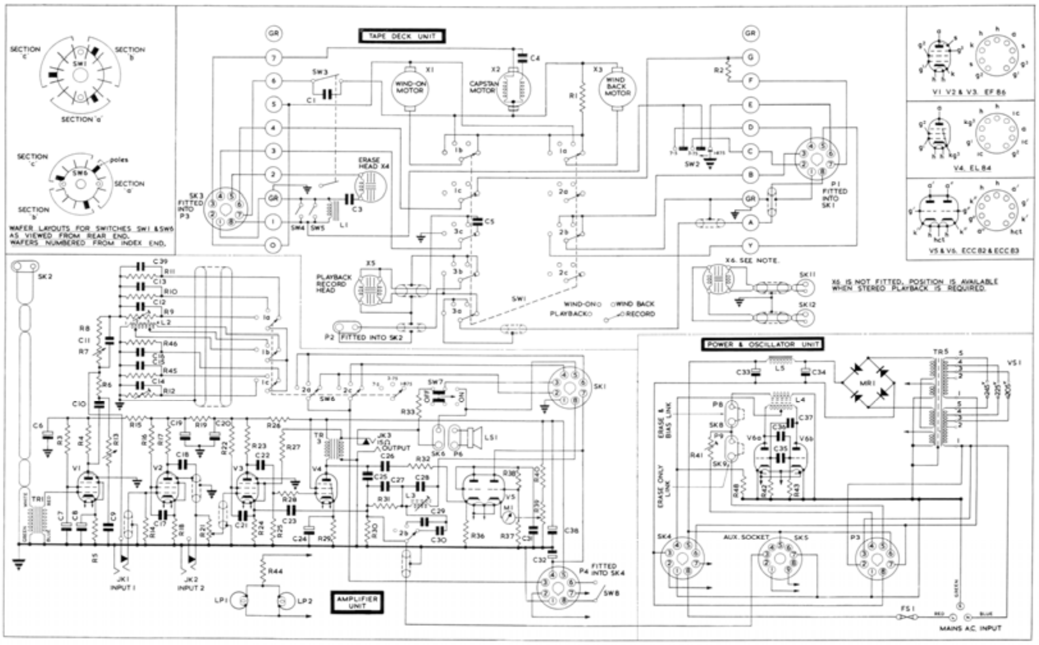
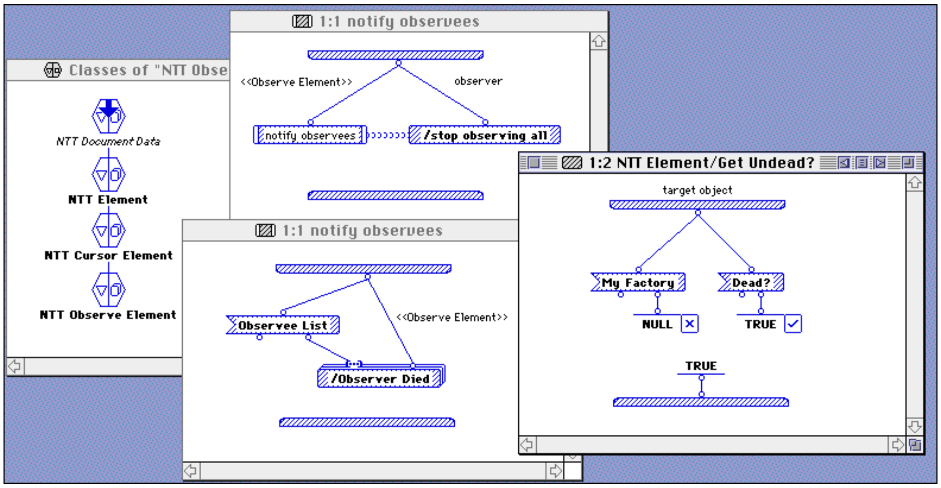


FIG. 20 CIRCUIT DIAGRAM.

Example VLs: Prograph

- Prograph (Cox et al 1989) uses a visual dataflow metaphor
    - – dataflow metaphor very popular in VL – nodes for processing elements, arcs for dataflows



Prograph

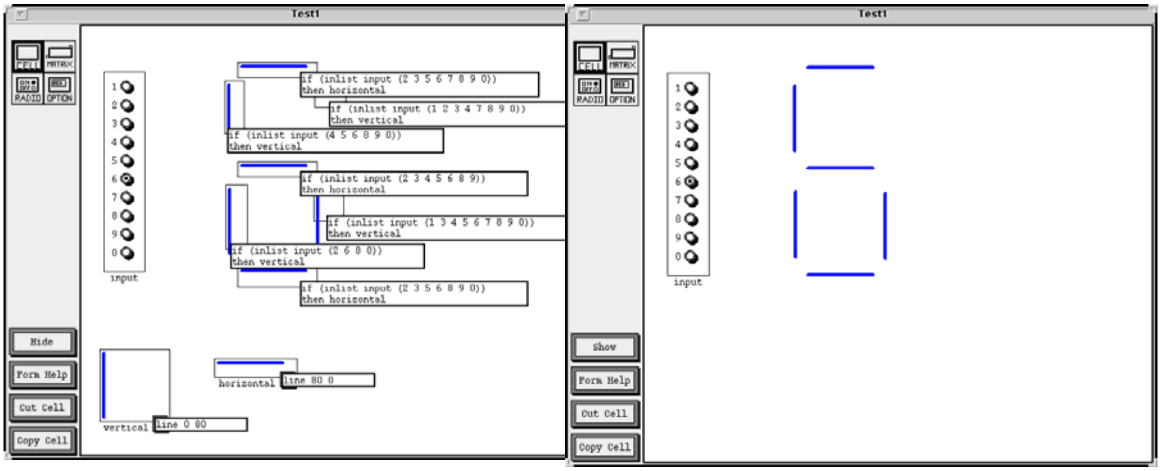
- Has a well developed OO framework
  - Dataflow “methods” for classes
  - GUI library framework allows rapid prototyping of applcns
- Has extensive debugging support
  - Reuses dataflow diagrams during execution with values instantiated to visualise execution behaviour
- Probably the only “successful” commercial general purpose visual programming language

Example VLs: Forms/3

- Forms/3 (Burnett 1995,98) uses a spreadsheet metaphor
- Programmer constructs forms with free format cells (not fixed to a grid) using direct manipulation
- Each cell has a formula which may refer to contents of other cells, possibly in other forms
- Linked formulae create a one-way constraint network – consistency is maintained
- Can construct types and instantiate them (prototype approach to OO) – cells can reference instances
- Can sketch shapes

Forms/3

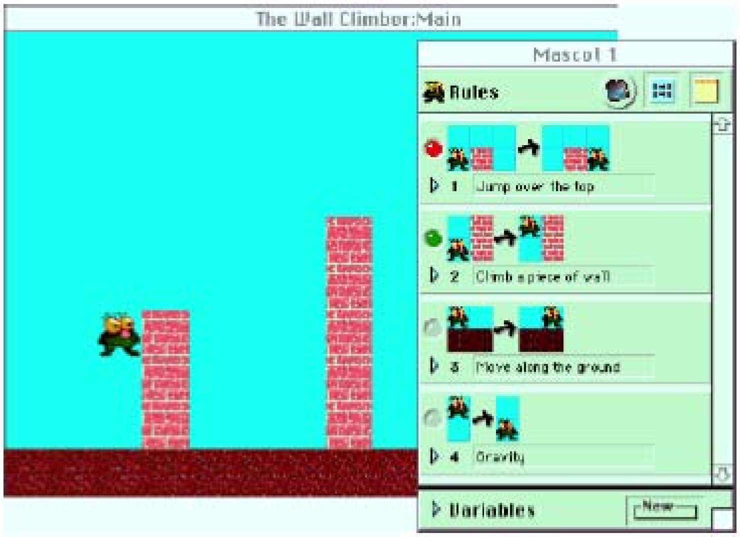
- Aimed at non-programmers
- Much recent work on adding test tools (see EUSES project)



```
if (inlist input (2 3 5 6 7 8 9 0))
then horizontal
if (inlist input (1 2 3 4 7 8 9 0))
then vertical
if (inlist input (4 5 6 8 9 0))
then vertical
if (inlist input (2 3 4 5 6 8 9))
then horizontal
if (inlist input (1 3 4 5 6 7 8 9 0))
then vertical
if (inlist input (2 6 8 0))
then vertical
if (inlist input (2 3 5 6 8 9 0))
then horizontal
horizontal line 80 0
vertical line 0 80
```

Example VLs: KidSim/Cocoa

- Cocoa (Smith et al 1994) uses a rule based metaphor combined with a 2-D cellular grid
  - Rules are specified using programming by demonstration
  - Aim is to make programming accessible to kids



KidSim/Cocoa

- Characters are defined
- Rule preconditions specify character proximities/ orientations
- Rule actions may remove or relocate characters, introduce new characters, etc
- Order-based disambiguation of rules if multiple rules for a character can fire
- Developed into commercial product: Stagecast Creator
- Several other similar languages, most notable of which is AgentSheets (Repenning). Alice has similarities.

Goals and strategies of visual programming

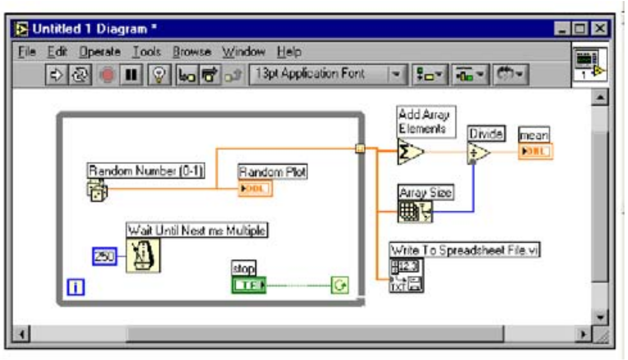
- Goals
  - make programming more accessible to some audience (often end users)
  - improve correctness of performing programming tasks
  - improve speed of performing programming tasks.
    - NB what's a "programming task" – see attention investment later
- Strategies
  - Concreteness: express program using specific instances
  - Directness: feeling of directly manipulating object
  - Explicitness: making relationships explicit
  - Immediate visual feedback or liveness: automatic display of effects of manipulations, even to the extent of editing "code" of running programs (cf spreadsheets)
  - Small number of concepts
- Metaphor is important

Domain Specific VLs

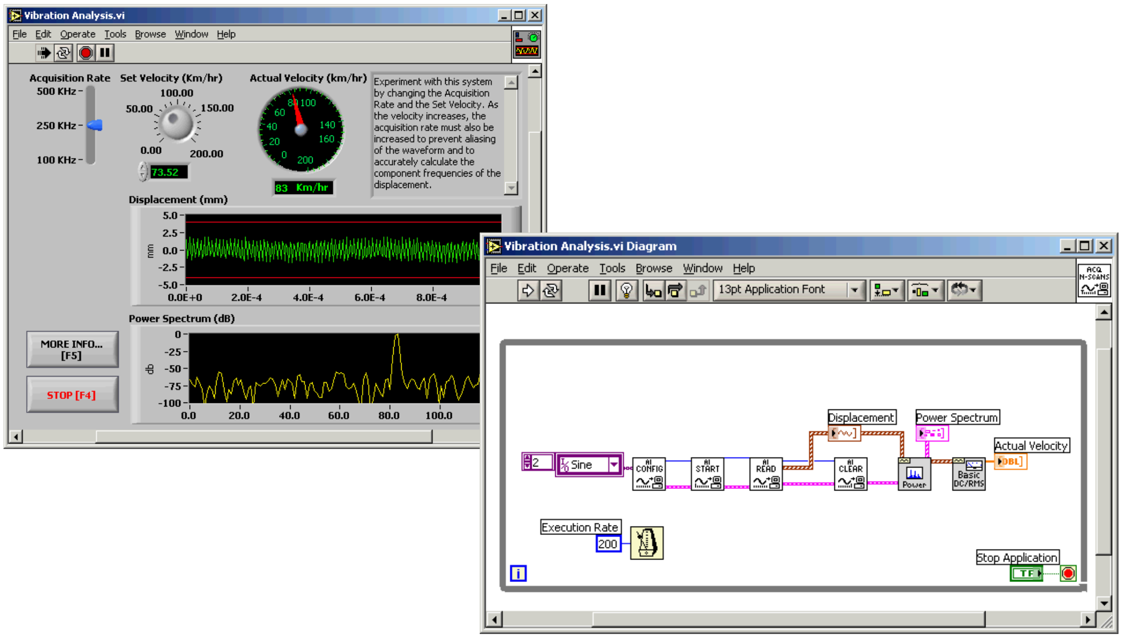
- A domain specific visual language is one where the notation is customised for a particular problem domain
- Have trade off between generality of language (ie range of problems able to be solved) and terseness of notation and closeness of mapping (cognitive dimensions concepts – see later)
- Look at:
  - A few widely used DSVLs
  - Some 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



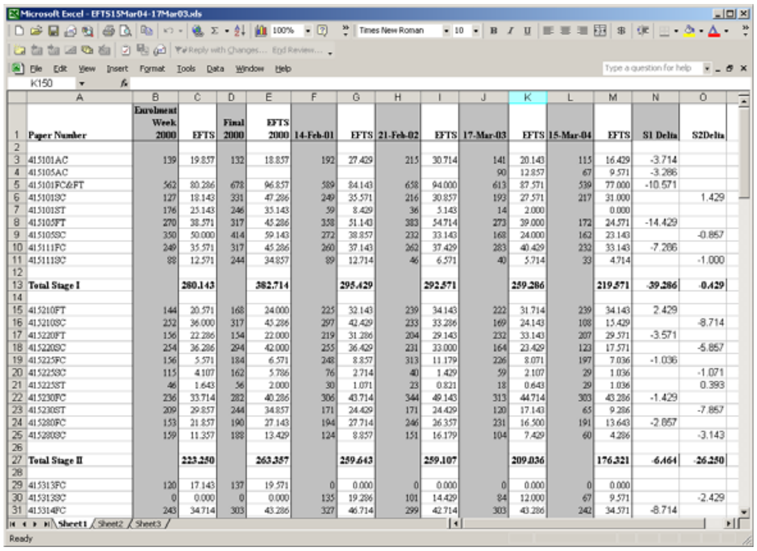
Experiment with this system by changing the Acquisition Rate and the Set Velocity. As the velocity increases, the acquisition rate must also be increased to prevent aliasing of the waveform and to accurately calculate the component frequencies of the displacement.

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


Microsoft Excel - EFTS15Mar04-17Mar03.xls
100%
Times New Roman
10
Reply with Changes... End Review...
File Edit View Insert Format Tools Data Window Help
Type a question for help
K150
Sheet1 Sheet2 Sheet3
Ready

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	**Paper Number**	**Enrolment Week 2000**	**EFTS**	**Final 2000**	**EFTS 2000**	**14-Feb-01**	**EFTS**	**21-Feb-02**	**EFTS**	**17-Mar-03**	**EFTS**	**15-Mar-04**	**EFTS**	**S1 Delta**	**S2Delta**
2															
3	415101AC	139	19.857	132	18.857	192	27.429	215	30.714	141	20.143	115	16.429	-3.714	
4	415105AC									90	12.857	67	9.571	-3.286	
5	415101FC&FT	562	80.286	678	96.857	589	84.143	658	94.000	613	87.571	539	77.000	-10.571	
6	415101SC	127	18.143	331	47.286	249	35.571	216	30.857	193	27.571	217	31.000		1.429
7	415101ST	176	25.143	246	35.143	59	8.429	36	5.143	14	2.000		0.000		
8	415105FT	270	38.571	317	45.286	358	51.143	383	54.714	273	39.000	172	24.571	-14.429	
9	415105SC	350	50.000	414	59.143	272	38.857	232	33.143	168	24.000	162	23.143		-0.857
10	415111FC	249	35.571	317	45.286	260	37.143	262	37.429	283	40.429	232	33.143	-7.286	
11	415111SC	88	12.571	244	34.857	89	12.714	46	6.571	40	5.714	33	4.714		-1.000
12															
13	**Total Stage I**		**280.143**		**382.714**		**295.429**		**292.571**		**259.286**		**219.571**	**-39.286**	**-0.429**
14															
15	415210FT	144	20.571	168	24.000	225	32.143	239	34.143	222	31.714	239	34.143	2.429	
16	415210SC	252	36.000	317	45.286	297	42.429	233	33.286	169	24.143	108	15.429		-8.714
17	415220FT	156	22.286	154	22.000	219	31.286	204	29.143	232	33.143	207	29.571	-3.571	
18	415220SC	254	36.286	294	42.000	255	36.429	231	33.000	164	23.429	123	17.571		-5.857
19	415225FC	156	5.571	184	6.571	248	8.857	313	11.179	226	8.071	197	7.036	-1.036	
20	415225SC	115	4.107	162	5.786	76	2.714	40	1.429	59	2.107	29	1.036		-1.071
21	415225ST	46	1.643	56	2.000	30	1.071	23	0.821	18	0.643	29	1.036		0.393
22	415230FC	236	33.714	282	40.286	306	43.714	344	49.143	313	44.714	303	43.286	-1.429	
23	415230ST	209	29.857	244	34.857	171	24.429	171	24.429	120	17.143	65	9.286		-7.857
24	415280FC	153	21.857	190	27.143	194	27.714	246	26.357	231	16.500	191	13.643	-2.857	
25	415280SC	159	11.357	188	13.429	124	8.857	151	16.179	104	7.429	60	4.286		-3.143
26															
27	**Total Stage II**		**223.250**		**263.357**		**259.643**		**259.107**		**209.036**		**176.321**	**-6.464**	**-26.250**
28															
29	415313FC	120	17.143	137	19.571	0	0.000	0	0.000	0	0.000	0	0.000		
30	415313SC	0	0.000	0	0.000	135	19.286	101	14.429	84	12.000	67	9.571		-2.429
31	415314FC	243	34.714	303	43.286	327	46.714	299	42.714	303	43.286	242	34.571	-8.714	

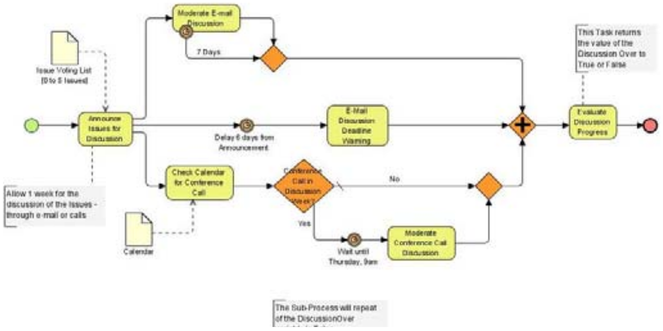
Spreadsheet success

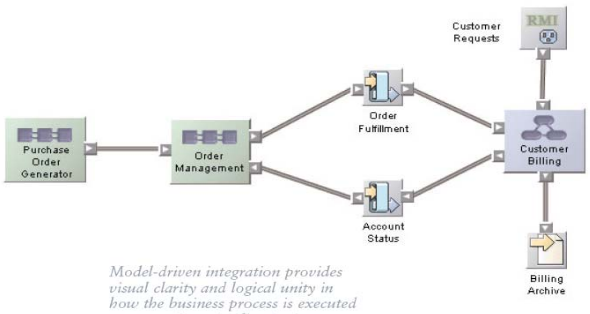
- Strong and consistent metaphor providing high closeness of mapping to typical balance sheet etc problems
- At one level notation is quite terse (sheet and cell metaphor), at another it is quite verbose (extensive range of functions that stretch the bounds of the matephor)
- Progressive evaluation well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

Business process modelling

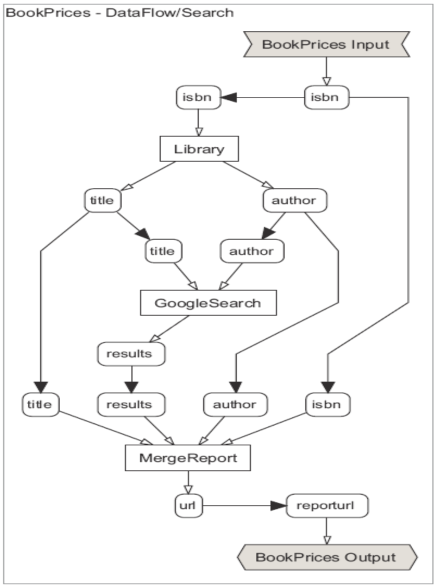
- Since the early 1970s many languages, standards, methodologies and tools for business modelling have been created
- Methodologies: ER Models, DFDs, Flow Charts, Scenarios, Use Cases, IDEF, etc.
- Notations: UML, BPMN, BioOpera, WTD, AOM etc.
- Tools: JOpera, T-Web, ZenFlow, ARIS, WebSphere, Visio etc.

Box-and-line Style Diagrams


Moderate E-mail Discussion
7 Days
Issue Voting List [0 to 5 Issues]
This Task returns the value of the Discussion Over to True or False
Announce Issues for Discussion
Delay 6 days from Announcement
E-Mail Discussion Deadline Warning
Evaluate Discussion Progress
Allow 1 week for the discussion of the issues - through e-mail or calls
Check Calendar for Conference Call
Conference Call in Discussion Week?
No
Yes
Calendar
Wait until Thursday, 9am
Moderate Conference Call Discussion
The Sub-Process will repeat of the DiscussionOver



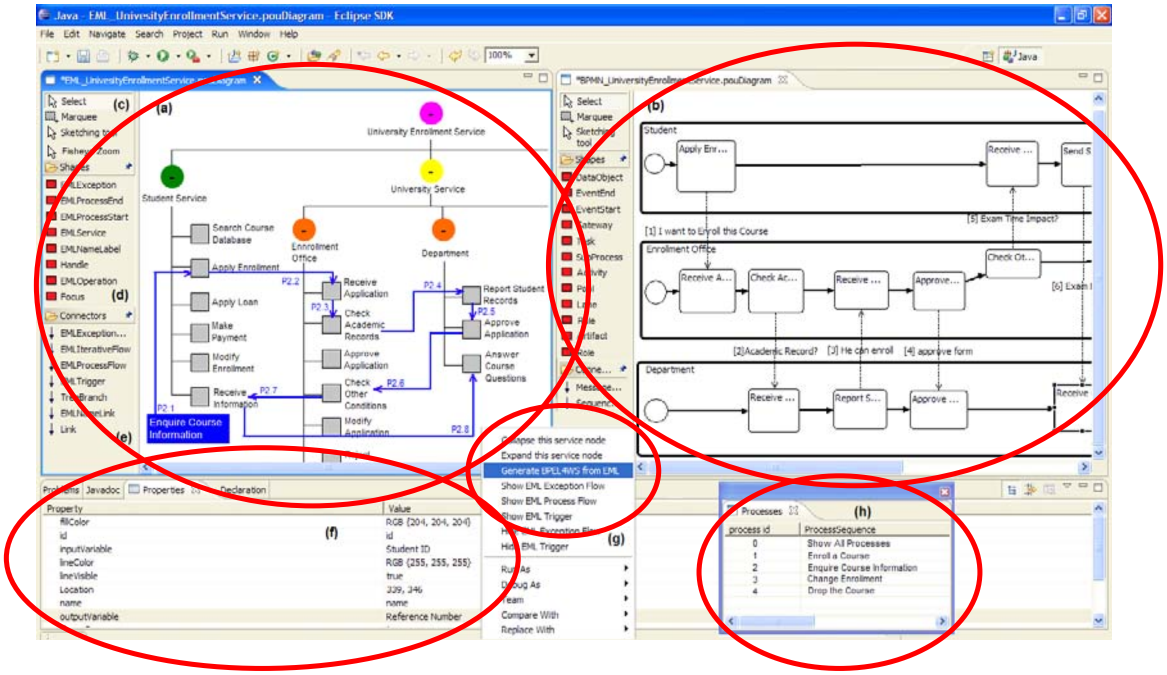
Model-driven integration provides visual clarity and logical unity in how the business process is executed across systems, trading partners, and workflow.



Motivation for MaramaEML

- Most of these approaches only emphasize process modelling, missing the ability to model system functional architecture
- Common source of difficulty: appropriate visual methods to reduce the complexity of large business modelling diagrams
- Existing modelling technologies are:
  - effective in only limited problem domains or
  - have major weaknesses when attempting to scale to large systems modelling
    - e.g. "cobweb" and "labyrinth" problems

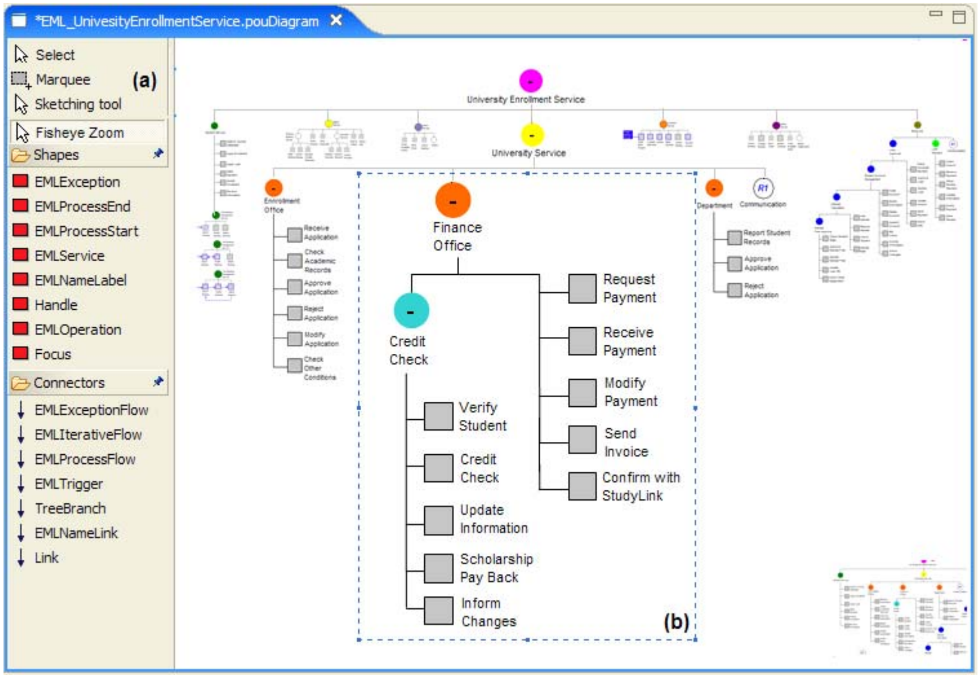
MaramaEML



Property	Value
fillColor	RGB {204, 204, 204}
id	id
inputVariable	Student ID
lineColor	RGB {255, 255, 255}
lineVisible	true
Location	339, 346
name	name
outputVariable	Reference Number

process id	ProcessSequence
0	Show All Processes
1	Enroll a Course
2	Enquire Course Information
3	Change Enrollment
4	Drop the Course

Distortion-based view for scalability



Design and Evaluation of Visual Languages

- How “good” are the languages we have just looked at?
- How can we design such languages so they meet users needs?
- Difficult:
  - Combination of psychology, user interface design, abstraction skills, expressability, narrowness of task, etc, etc
  - Typical usability studies are VERY expensive
  - Need some lightweight “tools” to help us understand the impact of design decisions

VL Design aids

- Two aids to design and evaluation of VLs:
  - Usability Analysis of Visual Programming Environments: a cognitive dimensions framework
    - Green and Petre
  - The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering
    - Moody
- Will explore both next time
- Readings: you MUST read both papers for next lecture
- Available from the Resources page of the 732 website