Visual Modelling of Complex Business Processes with Trees, Overlays and Distortion-based Displays

Lei Li[1], John Hosking[1] and John Grundy[1,2]
[1]*Department of Computer Science and* [2]*Department of Electrical and Computer Engineering*
University of Auckland, New Zealand
{L.Li, john, john-g}@cs.auckland.ac.nz

Abstract

Current approaches to modelling complex business processes fail to scale to large organizations. Key issues are cobweb and labyrinth problems exhibited by conventional box and line metaphors and large numbers of hidden dependencies introduced by compartment-based modularity. We have been developing a new approach, Enterprise Modelling Language, based on trees, overlays and fish-eye viewers to overcome these shortcomings of existing workflow notations. EML utilizes several visual metaphors to enhance the representation, navigation and management of large organizational hierarchies and process flows. We describe a prototype support tool, MaramaEML, that provides support for multiple visual notations including the Business Process Modelling Notation (BPMN), the EML tree-based, multi-layer hierarchical representation, fisheye zooming capabilities, automatic BPEL code generation, and inter-notation mapping. We describe our experiences using EML to model large business processes and initial evaluation results.

1. Introduction

Business process modelling integrates typical business practices, processes and information flows, data stores and system functions. A vast number of visual technologies have been applied in the business process modelling domain to capture graphical representations of the major processes, flows and data stores [11]. Examples include Entity-relationship models [19], Data Flow Diagrams [4], Aspect-oriented Modelling [2], Flowchart Models [17], Form Chart Approaches [8], Scenarios[10], Use Cases [6], Constraint Based Languages [11] and Integration Definition for Functional and workflow Modelling [21]. Despite their different visual approaches, most of these modelling technologies and their notations rely on the use of process flow or "workflow" structure to describe the business processes. Using workflow approaches, business processes are modelled as stages, tasks and links to represent the operational aspects. They focus on how systems are structured, who and how perform business tasks, what the process ordering is, how to manage information transformation, how to track the tasks, etc.

While providing a generally accepted metaphor for process modelling, these workflow-based visual modelling methodologies present their own set of problems. These include lack of an efficient way to reduce the complexity and enhance the scalability of large business diagrams, "cobweb" and "labyrinth" layouts in large processes, requiring long term memory use for multi-view support; introducing many hidden dependencies; lack of multiple levels of abstraction support; and most of them only emphasize process modelling, missing the ability to model system functional architecture [2][11][14][17].

Our early research into dialogue notations for user interfaces [15][16] showed that a tree structure is a very effective visual method to represent hierarchical relationships and existence of dependencies in complex flow-based systems. Thus we have developed a novel tree based visual notation called Enterprise Modelling Language (EML) to address and mitigate the shortcomings of current workflow-based notations. A support tool, MaramaEML, has been developed to provide facilities such as fisheye zooming, multilayer navigation, automatic layout, code generation and notation integration. Our goals are to offer a clear, concise and compact tree overlay notation to mitigate the cobweb problem, to provide a software tool improving business process modelling with EML, and to support integration of EML with other process modelling notations.

2. Motivation

We were asked to model a large university enrolment system as part of a process improvement exercise. This is a complex enterprise system that involves dynamic collaborations among five distinguished parties: Student, Enrolment Office, Department, Finance Office and StudyLink (the New Zealand government's student loan agency).

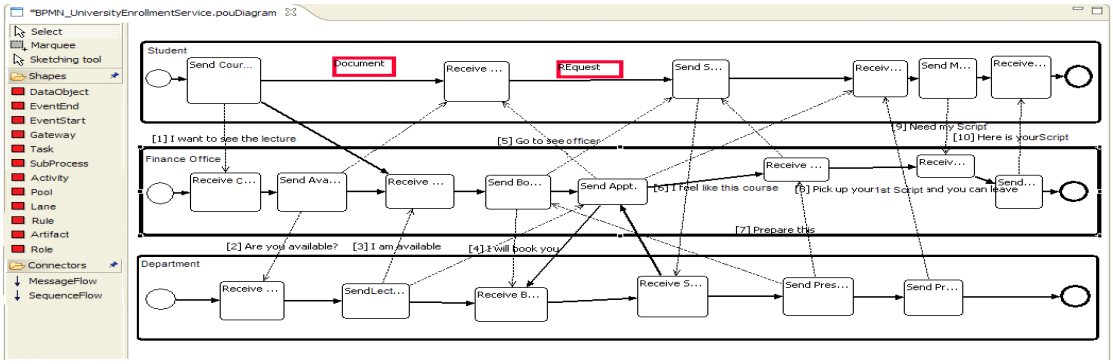

Figure 1. Part of a BPMN specification of the Enrolment System.

The main functional requirements are: (1) Students will use this system to search the course database and apply for enrolment in target courses; if their application is approved, they need to apply for a loan from StudyLink; (2) After receiving student applications, the Enrolment Office checks the academic conditions with academic Department staff and then informs Students of the results; (3) Department staff check the course enrollment conditions and make the final decision (approve or reject); (4) For an approved enrolment application, the Finance Office tracks fee payment and informs the Enrolment Office and Department of any changes. If a Student applies for a loan, the Finance Office also needs to confirm the student information with StudyLink. (5) StudyLink investigates the student information with the university and then approves (or declines) the loan application.

A conventional Business Process Modelling Notation (BPMN) diagram capturing some of this enrolment process is shown in figure 1. This illustrates the use of process stages, "swim lanes", process flow, etc. when modelling a process. Unfortunately as the process definition grows, the user must create either massively complex and unwieldy diagrams or "drill down" into sub-stages, introducing hidden dependencies and complex navigation [10][11][21].

What we require is an enterprise modelling tool that includes a visual language that:

a) can efficiently model distributed complex systems and related collaborations
b) can present multi-level abstraction to assist different process specifications
c) is easy to understand by both business and technology participators
d) addresses the problem of modelling over-complex diagrams among distributed parties
e) can be integrated effectively with other modelling technologies
f) supports automatic generation from visual models to industry standard code e.g. BPEL scripts

We have evaluated various visual modelling languages and support tools to model such a system. We found that most existing modelling languages and tools only solve limited design issues. General purpose modelling languages like UML and Petri Nets [4][6] have a well-established set of modelling notations and constructs. Though they are sufficiently expressive to model business scenarios, they are difficult for a business user to learn and use (fails items c and e above). Domain specific languages like Web Transition Diagrams (WTD) and T-Web systems [13] are very easy to understand but are limited to the scope of service level composition and modelling. They are not efficient in presenting multi-level abstractions of business processes (fail item b). Business oriented frameworks like ARIS and TOVE [1][10] are based on generic and reusable enterprise data model technology. They also provide a holistic view of process design, but focus too much on technical processes and efficient software implementation. Hence, they can result in ambiguity of the models as extra programming knowledge is required (fails items a, c, d). Some efficient modelling languages like BPMN, BioOpera, Formchart and ZenFlow [3][5][7][8] use simple notations to represent processes and also provide support tools to automatically generate industry standard code like BPML and BPEL4WS [5]. They all use workflow-based box and line methods to describe the system. Severe cobweb and labyrinth problems appear quickly using this type of notation to model the enrolment system [14]. Multi-view tool support has been applied in many such systems to mitigate this problem but this increases hidden dependencies and requires long term memory to retain the mental mappings between views (fails item d).

Our earlier work [15][16] on modelling complex user interfaces and their behaviour with visual dialogue notations demonstrated a tree-based overlay structure can effectively mitigate these complexity problems.

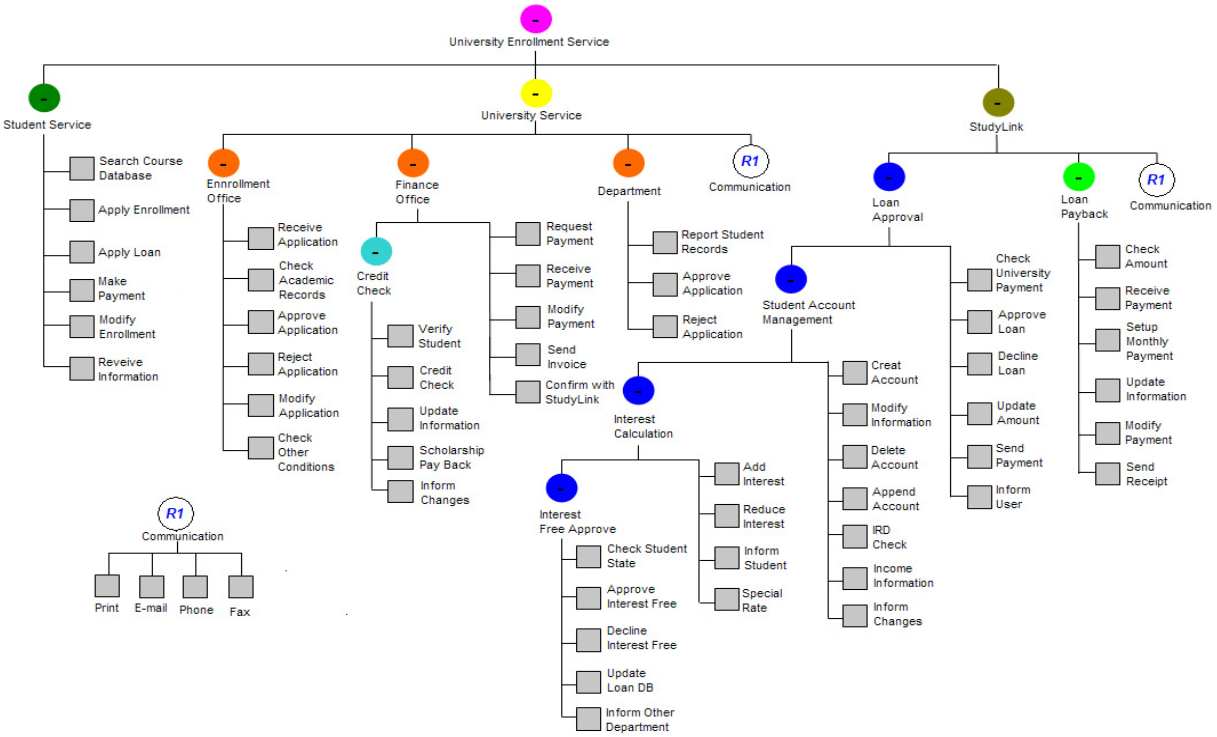

Figure 2. University Enrollment Service Structure.

They are familiar abstractions to manage complex hierarchical data for business modellers and business people; can be easily collapsed and expanded for scalability; can be rapidly navigated; and can be over-laid by cross-cutting flows and concern representations.

3. Enterprise Modelling Language

To meet the requirements of section 2, we have developed EML and a support tool, MaramaEML. EML's main features include a tree-based modelling representation, service collapse, elide and expand functions; three layers of flow integration (Process Flow, Exception Flow and Trigger Flow); iteration and component reuse modelling; data and condition encapsulation; exclusive and concurrent choice; and synchronizing merge/split.

3.1 Tree Layout

EML uses a tree layout to represent the basic structure of a service. Figure 2 shows a complex, fully-expanded overview of an EML tree modelling the university enrolment service. The student service, university service, and StudyLink are sub-services (represented as ovals) of the university enrolment service. The university service includes five embedded services (enrolment office, finance office, credit check, department and communication). The rectangle shapes represent atomic operations inside the service. The StudyLink service also includes a detailed four layer sub-service structure.

Even in this complex model the EML diagram still provides a clear structural view. In an EML-modelled enterprise system, major services are represented as separate trees. In order to mitigate the complexity of the diagram, we use symbols inside each service to identify the elision level of the service visualisation. A minus (-) symbol indicates all activities in the service have been expanded (e.g. all the services in figure 1). A plus (+) symbol indicates that part or all of the sub-tasks (services) are elided (e.g. *Loan Payback service, Student Account Management service* and *Credit Check Service* in figure 3). Every notation in the diagram can be elided and expanded to give users freedom to control the diagram size. Each tree element has a set of detailed properties e.g. service type, status, input, output, loop, condition, rule etc.

3.2 Process Flow (Overlay)

A fundamental part of business process modelling is the representation of flow between stages. In EML each business process is represented as an overlay on the basic tree structure or an orchestration between different service trees. In a process layer, users have the choice to display a single process or collaboration of multiple processes.

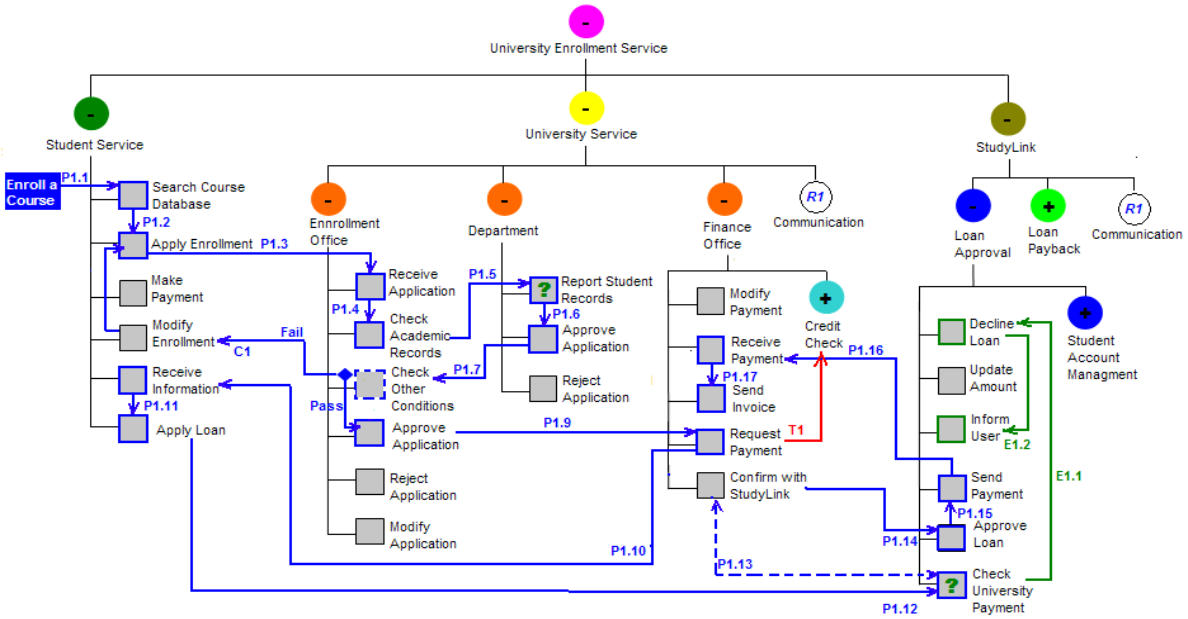

Figure 3. Using EML overlays to model the *Enrol in a Course* process.

By modelling a business process as an overlay on the service tree, the designer is given a clear overview of both system architecture and process concurrently. Processes can be elided mitigating the cobweb problem common in existing flow-based visual notations.

For example P1.1 to P1.17 in Figure 3 shows the *Enrol in a Course* process on the *University Enrolment Service* tree. The process starts with a process name followed by a process flow (blue arrow) representing the sequence. Each flow has a sequence number; for a complex process, users can use this to model concurrency / synchronization. Outline borders of operations or services used are bold to identify the track. Data is bound to a process flow to flow in or out of operations. In this process, the student uses *Search Course DB* to select the suitable course and *Applies Enrolment.* The enrolment officer *Receives Application* and *checks this student's Academic Records* with the *Department*. As soon as the *Department Reports the student's record* and *Approves the course Application*, the enrolment officer will *Check other Related Conditions* and ask the finance officer to *Request the Payment.* The student then *Applies Loan* and StudyLink *Checks University Payment* information with the Finance Office and decides if it *Approves or Declines the Loan*. If the university receives payment from StudyLink, the finance officer confirms the enrolment and *Sends the Invoice* to the student.

3.3 Service Reuse

EML supports service reuse to reduce structure complexity and increase modelling efficiency. A reusable component is represented in a separate tree. The user pre-defines its structure and saves it in a library. Reusable components have a unique name for future usage. The user can easily attach a reusable component to any branch of an EML tree. In Figure 2 we define a *Communication Service* as a reusable component (at the left bottom), reused by the *University Service* and *StudyLink Service*.

3.4 Dependency / Internal Exception (Overlay)

It is important to know if a specific event occurs or condition met. Events and conditions are referred to as dependency relationships. In some cases, we can also treat internal (system) exceptions as triggers. An EML trigger layer can be used to solve dependency problems. T1 in Figure 3 shows how dependency information can be passed from one part of a process to another if a normal process flow is insufficient. The red (lighter grey) single arrowhead trigger connector (T1) represents the dependency. In above example, when the Finance Office *Requests the Payment* from the student, they also need to *Check student's Credit.* If the student has a scholarship, the request payment amount may be changed. The user can define the trigger conditions as attributes at each end of the connecter to control the dependency. The start and end point of a trigger can be a service, operation or process. Since EML uses a multi-layer structure, users can choose to combine the trigger and process layers (as in Figure 3) or separate them, using different views to reduce complexity.

3.5 Iteration

EML supports specification of process iteration at

different levels. (1) A single activity loop is represented as a dashed outline border. Attributes control the iteration (e.g. loop times, start and complete conditions, input/output data etc.). *Check Other Conditions* in Figure 3 is a single activity loop example. After the department *approves the course enrolment application* based on academic record, the enrolment office use this function to repeat all the other related conditions (e.g. available seats in class, test time impact, tutorial group assign etc). (2) Loops of two operations, use a dashed line with two arrowheads. Process P1.13 in Figure 3 shows iteration of the *Check University Payment* and *Confirm with StudyLink* operations. When StudyLink received the student loan application, they need to check all course related information with the university (e.g. student status, course fee amounts, start and end date etc.). The process loops until a termination condition is met (all the information has been confirm). (3) If a loop involves more than three operations, a single arrowhead dashed line guides direction, linking different operations or services in a closed circuit.

3.6 Exception Handling (Overlay)

EML's exception overlay is used to model errors in transactions. A failure handling notation (question mark in the middle of an operation or service) specifies a transaction failure. Users can set up a start condition to discriminate different kinds of failures and activate appropriate exception handlers. An exception handling layer is constructed to model transaction error handling in detail. For example, Figure 3 shows the *Enrol a Course* process with two exception handlers overlaid. When the *Department* staff check the student's academic record, an error handler is added to the operation (question mark in *Report Student Records*). If the student's previous academic record doesn't satisfy the course prerequisite, it will decline the application and drive the exception handler to carry out an alternative process (negotiate an alternative course with the student). A second exception hander is on the *Check University Payment* operation. If the student loan application cannot be fully confirmed by the *Finance Office*, the alternative is to *Decline Loan Application* and *Inform the Student.* Two green connectors (E1.1~E1.2) represent the exception flow.

3.7 Exclusive and Multiple Choices

A diamond shape in Figure 3 (attached to the boundary of *Check Other Condition*) is used to express a conditional flow. If the other course related conditions (e.g. an exam clash with another course) cannot be fully satisfied (*Fail*), it informs the student to *Modify Enrolment*. Symbol *C1* is an annotation used to describe the flow execution condition. Here, it may be a possible non-clash exam time table for the student to reference. If the student *Pass*es the checking, the enrolment officer will then *Approve the Application*.

4. MaramaEML

We have developed an integrated design environment, MaramaEML for creating EML specifications. This IDE provides a platform for: efficient EML visual model creation, inspection, editing, and storage, model driven code generation, and integration with other diagram types. We summarise MaramaEML's major features here. Figure 4 (a) shows a screen dump of a MaramaEML model in use with a typical EML tree plus a process overlay. Diagram selection options are in (c). The user can use normal or marquee selection, tablet based diagram sketching or zooming to control EML diagrams. MaramaEML shapes toolbars in (d) and (e) provide options relating to the construction and editing of EML tree. Detailed information is entered in the *Properties* window (f). Elision and expansion are triggered via popup menus (g) or the +/- elision buttons. *Collapse this service node* and *Expand this service node* functions are available for the user to elide or expand a service node. The user can also select *Show/Hide EML Process/ Exception/Trigger Flow* functions to view or hide overlays. When a *Show/Hide Flow* function is selected, a detailed flow list is brought to the screen for further selection (h). By double clicking the process names in this list, the user can choose to view one (or more) appointed process or all of them. Similar operations apply to the *Exception and Trigger Flows.*

Due to the complexity of business processes, a single modelling notation is insufficient to satisfy all modelling needs. Using integrated visual modelling business process structures representing different aspects can be built and maintained graphically.

For instance, in EML, the data are bound to the process flows via textual properties to reduce diagram complexity. However, sometimes a user requires this kind of information to be presented in the diagram. BPMN diagrams represent the internal data through data flow sequence well, but this kind of flow-based approach easily causes diagram cobweb problems. The ideal solution is to provide the user access to both diagram types. Our MaramaEML support tool includes concurrent BPMN, EML and Form Chart views.

Figure shows a multi-view collaboration screen dump with an EML view (a) and BPMN view (b) to model same process (*Enquire Course Information*). In

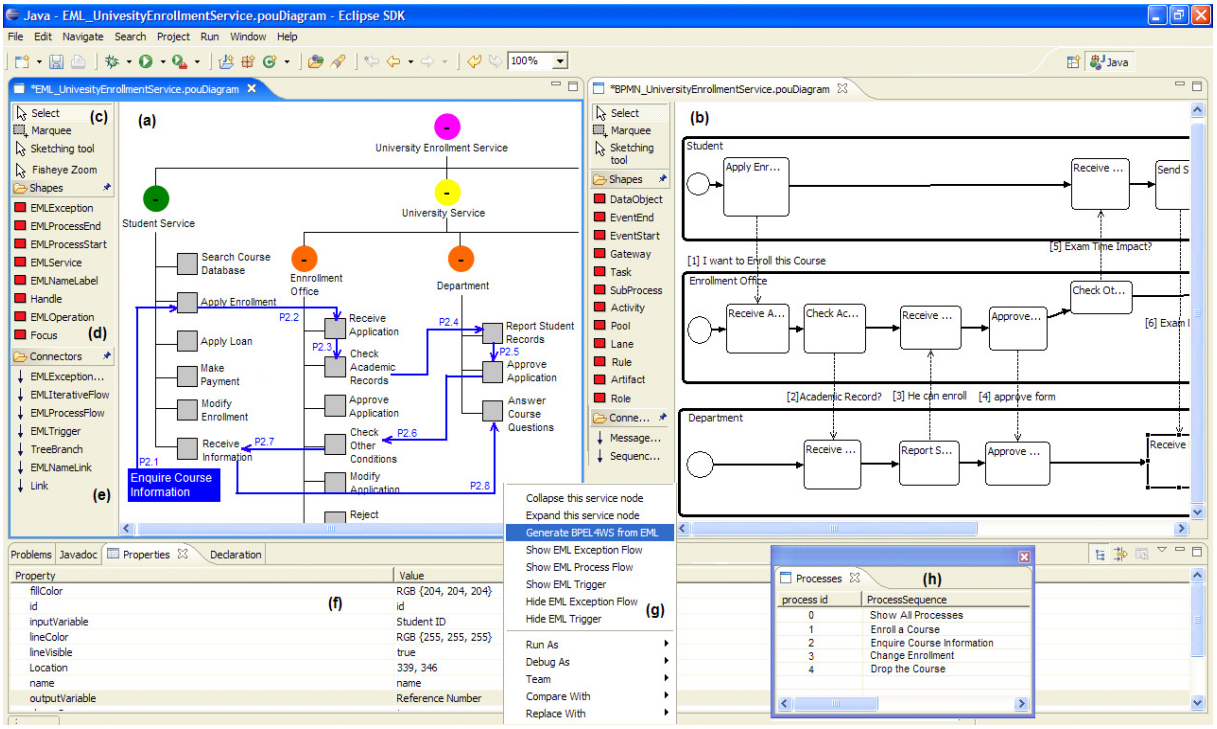

Figure 4. Using EML and BPMN Views to model the same business process.

the EML view the user can obtain a clear process sequence while they can also see the data transformation in the BPMN view.

In order to enhance EML diagram navigation a distortion-based fisheye zooming function has been developed. Figure 5 shows an EML fisheye view screen dump (a). The user draws a "fisheye area" (dashed line square). Components in this area (b) are represented at normal size (*Finance Office* Sub-tree), while the rest are distorted with the degree of shrinkage increasing with distance from the fisheye area.

To support code generation and process model validation we have integrated a BPEL code generator and LTSA engine into MaramaEML. In Figure 5 the EML process layer has been compiled to BPEL executable code automatically (c). Code is generated by model dependency analysis and translation to structured activity constructs. We have integrated an LTSA engine to verify the correctness of the generated BPEL (Business Process Execution Language) code. The LTSA engine compiles EML output (BPEL) and shows the results in (e). If there are no compilation errors, an LTS diagram (Labelled Transition System) is shown (d).

We used our Marama and Pounamu meta-tools to develop MaramaEML [12][18]. We specified the EML domain-specific visual language notation and meta-model and generated Eclipse-based editors from these to realise the basic support environment. The tree layout, overlays and distortion-based displays are all implemented as complex visual event handlers. The integration of EML with BPMN notation, code generation of BPEL, and LSTA engine integration are implemented as event-driven, model-level data updates.

5. Evaluation

We conducted two evaluations of MaramaEML: the first an extensive cognitive dimensions analysis [20]; the second a task-based end-user evaluation. We summarise these evaluations below.

5.1 Cognitive Dimensions Evaluation

Consistency (similar semantics are expressed in similar syntactic forms): The EML underlying structural tree provides a consistent framework on which similar semantic operations (standard process flow, triggers and exceptions) are overlaid using similar syntactic forms (flow, distinguished by colour).

Visibility (ability to view components easily): In MaramaEML different modelling notation views can be juxtaposed. The fisheye viewer supports a high degree of visibility within EML views, even for very large diagrams.

Premature commitment (constraint on the order of doing things): The user has considerable freedom to model a business process using any EML, BPMN and Form-Chart notation. In the EML view, the user can freely traverse through the tree structure view and the

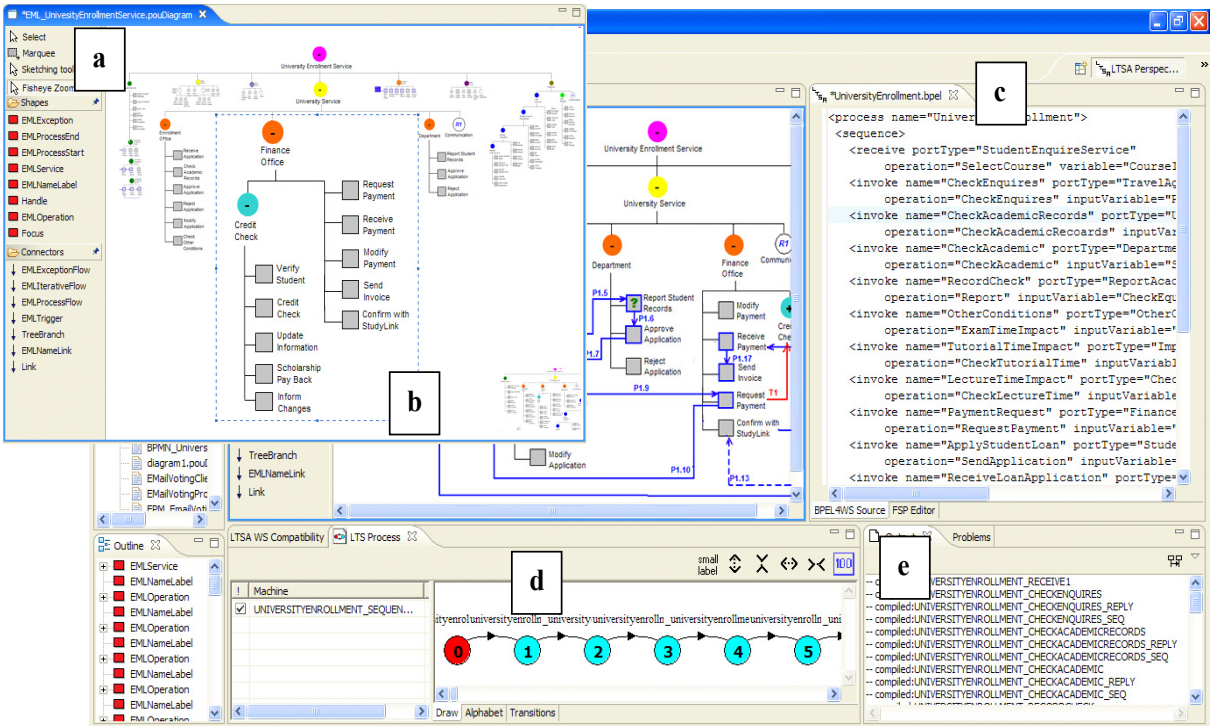

Figure 5. MaramaEML Fisheye Zooming and LTSA generation to validate process flows.

business process, triggers and transaction layer, or even integrate them together. However the user needs to define the business tree structure first and then construct process overlays.

Hidden dependencies (important links between entities are not visible): Although by default some dependencies are hidden, e.g. data bound to the process flows, and trigger and exception flows are normally not shown in a process layer, most are readily accessible via property sheets. The propriety window and multi-language modelling support can represent these kinds of abstraction with MaramaEML tool support. It is possible to show all the dependencies in the same layer if required.

Error proneness (the notation invites mistakes and the system gives little protection): The EML notation is simple, well-defined with high level business modelling graphical representations. MaramaEML enforces connectivity constraints and provides design feedback to users on correctness of notational usage.

Abstraction (types and availability of abstraction mechanisms): EML is a high level process modelling language but the metaphors it uses are very business-oriented and tailored for the enterprise domain, and the layout is very easy for the user to understand providing minimal abstraction gradient for the target end-users.

Secondary notation (extra information in means other than formal syntax): In EML, we integrate different shapes, colours and text descriptions together to convey information. Sketch annotations can also be added to diagrams to convey extra information.

Closeness of mapping (closeness of representation to domain): EML uses a tree metaphor to represent the service construction. This hierarchical structure is a natural way to model business structure and users are familiar with using it to model complex organizational hierarchies. Business processes are constructed using a flow-based overlay metaphor on top of the tree structure providing good closeness of mapping to process sequencing. The elision techniques mean that users can focus on one process at a time minimising their cognitive load.

Diffuseness (verbosity of language): EML uses quite a small set of language elements so it is quite terse and hence easy to learn.

Hard mental operations (high demand on cognitive resources): In EML, different states and components of a business process are well discriminated through the use of the tree-based hierarchy, process overlay, dependency trigger and different layers of exception handlers. The complexity of a business process has been successfully reduced in the EML multi-layer structure through its elision and overly techniques.

5.2 Task Based End-user Evaluation

We used several Computer Science research students to carry out a task-based end-user evaluation of EML and MaramaEML. The objective was to assess how

easy it is to learn to use EML and its support tool and how efficiently it can solve the diagram complexity problem. EML and MaramaEML were briefly introduced to the students and they were then asked to perform several predefined modelling tasks. The tasks were divided into simple, medium and complex levels, and they were asked to repeat the same task in two different environments (pen and paper based EML modelling and software tool-based integrated EML and BPMN modelling).

Feedback suggested EML and MaramaEML are very straightforward to use and understand. The users feel the tree overlay method is greatly favoured for reducing the complexity of business processes compared to using only conventional BPMN views. They found it very valuable to have a tree overlay based modelling language as a supplement to overcome the shortcomings of exiting business process notations. The multi-view collaboration is a useful approach to enhance the modelling strength. The fisheye zooming function is quite easy to use and increases the navigation ability evidently.

Several limitations and potential improvements have been identified in our evaluations. These include a need for more detailed mapping traceability, unresponsive computer speed in the software tool (when zooming a large, complex diagram), and a need for better control of information hiding. We are also planning a larger evaluation with business end-users.

6. Summary

We have described EML a novel business process modelling language based on tree hierarchy and overlay metaphors. Complex business architectures are represented as service trees and business processes are modelled as process overlay sequences on the service trees. By combining these two mechanisms EML gives users a clear overview of a whole enterprise system with business processes modelled by overlays on the same view. An integrated support tool for EML has been developed using the Eclipse based Marama framework. It integrates EML with existing business notations, BPMN and Form-Charts, to provide high-level business service modelling. A distortion-based fisheye zooming function enhances complex diagram navigation ability. MaramaEML can also automatically generate BPEL code from the graphical representations and map it to LTSAS for validation.

References

[1] A. Goel, *Enterprise Integration --- EAI vs. SOA vs. ESB*, Infosys Technologies White Paper, 2006

[2] A. Gokhale, and J. Gray, "An Integrated Aspect-Oriented Model-Driven Development Toolsuite for Distributed Real-Time and Embedded Systems", *Proc. 6th IWSAOM,* Chicago, 2005

[3] A. Martinez, M. Patino, etc, "ZenFlow: A Visual Web Service Composition Tool for BPEL4WS", *Proc of VL/HCC'05*, Dallas, 2005, P181~P188

[4] A. Schnieders, F. Puhlmann, and M. Weske, *Process Modelling Techniques,* PESOA Report No. 01/2004 Hasso Plattner Institute, 2004

[5] BPMI, http://www.ebpml.org/bpml.htm, 2006

[6] C. Marshall, *Enterprise Modelling with UML. Designing Successful Software Through Business Analysis,* Addison Wesley, 2000

[7] C. Pautasso, and G. Alonso, "Visual Composition of Web Services", *Proc. of VL/HCC'03*, Auckland, 2003, p92~p99

[8] D. Draheim and G. Weber, *Form-Oriented Analysis*, Springer-Verlag Berlin Heidelberg, 2005

[9] E. Guerra, P. Diaz, and J. Lara, "A Formal Approach to the Generation of Visual Language Environments Supporting Multiple Views", *Proc. VL/HCC'05*, Dallas, TX, 2005, p284 ~ p286

[10] F. Buschmann, R. Meunier, and H. Rohnert, etc, *Pattern-Oriented Software Architecture*. John Wiley and Sons, 1996

[11] H.E. Eriksson, and M. Penker, *Business modelling with UML: business patterns at work*, Wiley, 2000

[12] J.C. Grundy, J.G. Hosking, N. Zhu, and N. Liu, "Generating Domain-Specific Visual Language Editors from High-level Tool Specifications", *Proc ASE06* Japan, 2006, p24 ~ p28

[13] J. Kornkamol, S. Tetsuya, and T. Takehiro, "A Visual Approach to Development of Web Services Providers/Requestors", *Proc. of VL/HCC'03,* Auckland, 2003, p251~p253

[14] J. Recker, M. Indulska, M. Rosemann, and P. Green, "How good is BPMN really? Insights from Theory and practice." *Proc 14th ECIS*, Goeteborg, 2006

[15] L. Li, C.H.E Phillips, and C.J. Scogings, "Automatic Generation of Graphical Dialogue Model from Delphi", *Proc of APCHI2004*, Rotorua, 2004, p221~p230

[16] L. Li, The Automatic Generation and Execution of Lean Cuisine+ Specifications, *MSc thesis*, Massey University, New Zealand, 2003

[17] L. Urbas, L. Nekarsova, and S. Leuchter, "State chart visualization of the control flow within an ACT-R/PM user model", *In Proc. 4th Int. workshop on Task models and diagrams,* Poland, 2005, p43~p48.

[18] N.P. Zhu, J.C. Grundy, and J.G. Hosking, "Pounamu: a meta-tool for multi-view visual language environment construction", *Proc VL/HCC'04*, Rome, p254 ~ p256

[19] P. Chen, "Entity-relationship modelling: historical events, future trends and lessons learned", *Software Pioneers*, Springer, New York, 2002, p296 ~ p310

[20] T. Green, and M. Burnett, etc, "Cognitive Walkthrough to Improve the Design of a Visual Programming Experiment", *Proc VL2000*, 2000 P172~P179.

[21] V.O. Pinci, and R.M. Shapiro, "Work Flow Analysis", *Proc. of the 25th CWS*, LA, USA, 1993, P1122~P1130