Journal of Visual Languages and Computing, © Elsevier 2014

A visual language and environment for enterprise system modelling and automation

Lei Li
Beef & Lamb New Zealand
PO Box 121
Wellington 6140, New Zealand
Richard.Li@beeflambnz.com
Ph: +64-4-471-6035

John Grundy
Centre for Computing & Engineering Software Systems
Swinburne University of Technology
PO Box 218, Melbourne, Australia
jgrundy@swin.edu.au
Ph: +61-3-9214-8731

John Hosking
College of Engineering and Computer Science
Australian National University
Canberra, Australia
john.hosking@anu.edu.au
Ph: + 61-2-6125 8807

Abstract

Objective: We want to support enterprise service modelling and generation using a more end user-friendly metaphor than current approaches, which fail to scale to large organizations with key issues of "cobweb" and "labyrinth" problems and large numbers of hidden dependencies. Method: we present and evaluate an integrated visual approach for business process modelling using a novel tree-based overlay structure that effectively mitigate complexity problems. A tree-overlay based visual notation (EML) and its integrated support environment (MaramaEML) supplement and integrate with existing solutions. Complex business architectures are represented as service trees and business processes are modelled as process overlay sequences on the service trees. Results: MaramaEML integrates EML and BPMN to provide complementary, high-level business service modelling and supports automatic BPEL code generation from the graphical representations to realise web services implementing the specified processes. It facilitates generated service validation using an integrated LTSA checker and provides a distortion-based fisheye and zooming function to enhance complex diagram navigation. Evaluations of EML show its effectiveness. Conclusions: we have successfully developed and evaluated a novel tree-based metaphor for business process modelling and enterprise service generation. Practice implications: a more user-friendly modelling approach and support tool for business end users.

Keywords: business process modelling, web service generation, process enactment, zoomable user interfaces, domain-specific visual languages, business process modelling notation, business process execution language

1. Introduction

Business processes play a crucial role in running an organisation. In order to achieve process excellence, people carry out various process improvement initiatives, such as business process reengineering (BPR) and business process management (BPM) (Hill and Brinck et al 1994; Aguilar-Savén 2004), along with business process-based services (Li et al, 2010) and service composition (Gillain et al, 2013). BPM has been defined as "a structured, coherent and consistent way of understanding, documenting, modelling, analyzing, simulating, executing and continuously changing end-to-end business processes and all involved resources in light of their contribution to business success" (Recker 2010). BPM covers the overall management of organizations by looking at the lifecycle of their business processes (Box and Cabrera et al 2006; Kramer and Herrmann 2007). However, no matter which process improvement initiative people wish to conduct, they need to understand business processes, perform analyses to design or redesign them, and build or reuse appropriate services or service orchestrations to realise them (Jung and Cho 2005; Grundy et al 2006).

BPM and business process-based service composition have thus emerged as popular approaches in practice and research. However, while organizations appear to be well aware of the need for BPM and the advantages of using service-oriented architectures, implementation remains a challenging task. Indeed, many organizations still struggle with an efficient modelling approach to discover, visualize and document their business processes (Recker and Rosemann 2009). Examples of BPM approaches include Entity-Relationship models (Chen. 2002), Data Flow Diagrams, Flow Charts (Urbas and Nekarsova et al 2005), Scenarios, Use Cases, Integration Definition for Functional and workflow Modelling (Eriksson and Penker 2000), and Business Process Modelling Notation (BPMN) (BPMI 2010). Many types of workflow management and service modelling and

composition systems have been developed to model and implement enterprise business processes (Recker et al 2009; Paussto 2005; Leymann 2001; Xiong et al, 2010). Their goal is to specify, compose, co-ordinate, enact and evolve business processes using a high-level visual modelling approach. Using workflow approaches, business processes are typically modelled as stages, tasks and links. These models are then used to control the execution of software components that comprise an enterprise system. Process technology can also be used to model processes executed within systems e.g. in Enterprise Resource Planning (ERP) systems.

Despite this ongoing proliferation of process modelling languages, only a few have been widely accepted by practitioner communities. Research shows that visual process modelling methods differ significantly in their features and characteristics, such as, for instance, their representational capabilities, their support for expressing workflow patterns or their support for formal analysis of correctness criteria (Engels and Erwig 2005; Gamma et al 1995). Actual practice, on the other hand, informs us that certain BPM languages have achieved higher levels of adoption and dissemination in visual modelling practice than others while many visual modelling languages exist as objects of interest only to academic scholars (Gottfried and Burnett 1997; Grundy et al 2006). Most existing workflow based Enterprise visual modelling languages adopt box-and-line style diagrams, which work well for small to medium diagrams. A common difficulty with such approaches is a lack of scalability. Most existing modelling technologies are effective in only limited problem domains or have major weaknesses when applied to large system models resulting in "cobweb" and "labyrinth" problems, where users have to deal with many cross diagram flows. Most modelling tools use multi-view and multi-level approaches to mitigate this problem (Grundy et al 2000). These approaches have achieved some success but do not fully solve the problem, as using the same notation and flow method in a multi-view environment just reduces individual diagram complexity, but increases hidden dependencies between diagrams. This requires use of the long-term memory of the users, as they have to build and retain the mappings between views mentally. In addition, most existing flow based business modelling notations lack multiple levels of abstraction support.

We have been developing a new approach, Enterprise Modelling Language (EML) and a support environment, MaramaEML, to overcome some of these issues in a novel way (Li et al 2007; Li et al 2008). Our principal goal that directed the design of EML was to provide a simple, intuitive and executable visual notation to support rapid, user-friendly development of business process-based services. Our key target end-users are business process domain experts and service composition experts. EML adopts several visual metaphors to enhance the representation, navigation and management of large organizational hierarchies and process flows. It attempts to address many of the above limitations by modelling processes primarily by a novel tree overlay structure. In this new approach, complex business systems 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. However, our approach does not exclude existing modelling notations such as BPMN. We incorporate them in our EML support tool while providing additional richer, integrative views for enterprise process modelling. The objective of EML is to support business process management by both technical users and business users by providing a novel tree overlay based notation that is intuitive to business users yet able to represent complex process semantics. MaramaEML is an Eclipse-based integrated design environment for creating EML specifications. This IDE provides a platform for efficient visual EML model creation, inspection, editing, storage, model driven code generation of web services, and integration with other diagram types. Distortion-based fisheye and zooming functions have also been implemented to enhance MaramaEML's navigability for complex diagrams. MaramaEML supports BPEL code that is automatically generated from graphical EML representations and mapped to a Labelled Transition System Analyser (LTSA) (Foster and Magee et al 2003) for validation.

The remainder of this paper is organized as follows: Section 2 describes the motivation for our research and Section 3 provides an overview of the approach taken and its development. Section 4 introduces the detailed design of the Enterprise Modelling Language (EML), describing the visual representations of service tree structure, process overlay and exception handlers; as well as some more advanced constructs such as dependency / trigger, iteration and conditions. This section uses a complex business process example (a University enrolment system) to demonstrate the capabilities of EML and MaramaEML. The architecture and implementation of MaramaEML are discussed in section 5. Section 6 described a formal user evaluation of EML and MaramaEML with analysis of the evaluation feedback and possible improvements. We conclude the paper with a discussion and future work directions.

2. Motivation and Related Work

Our primary motivation for this research came from an attempt to model a large university enrolment system as part of a process improvement exercise, and to derive executable service orchestrations from these models to be enacted by a workflow system. This is a complex service-oriented 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). The main functional requirements of the system 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 academic conditions with academic Department staff and then informs Students of the results;
3. Department staff check course enrolment 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 supplied by the university and then approves (or declines) the loan application.

A Business Process Modelling Notation (BPMN) diagram capturing some of this enrolment process is shown in Figure 1. This illustrates the use of process stages, "pools", 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 (Baker 2002; Recker and Rosemann et al 2009; Adams 2011). While BPMN, like many approaches, provides mechanisms to manage complexity via sub-processes, these introduce potentially severe hidden dependency and lack of juxtaposition problems (Genon et al, 2011).

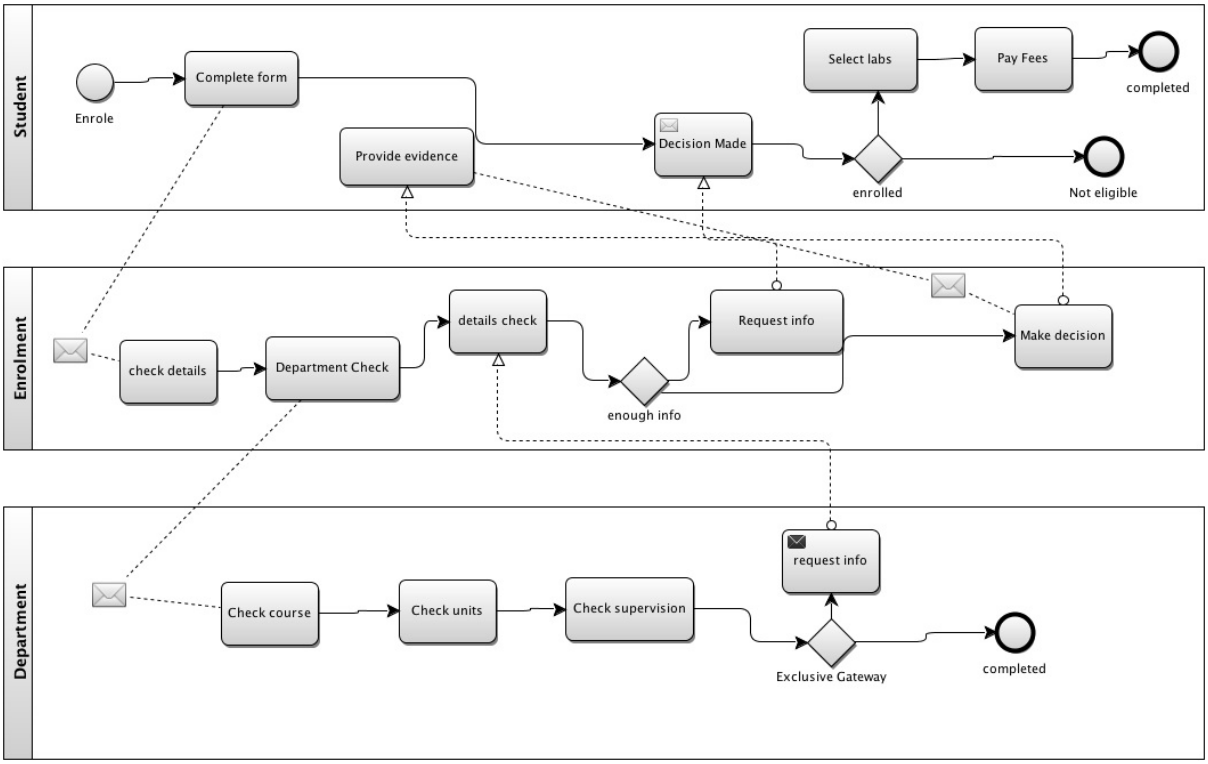

Figure 1: Part of a BPMN specification of the Enrolment System.

General-purpose modelling languages like UML (Schnieders and Puhlmann 2005) and Petri Nets (Marshall 2004) 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. Domain specific languages like Web Transition Diagrams (WTD) and T-Web systems (Kornkamol and Tetsuya et al 2003) 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. Business oriented frameworks like ARIS (Goel 2006) and TOVE (Buschmann and Rohnert 1996) are based on a generic and reusable enterprise data model technology and also provide a holistic view of process design. However they focus too much on technical processes and efficient software implementations and hence they can result in ambiguity of the models as extra programming knowledge is required.

Some efficient modelling languages like BPMN (BPMI 2010), BioOpera (Pautasso and Alonso 2005), YAWL (Adams et al 2011), Form Charts (Draheim and Weber 2005) and ZenFlow (Martinez and Patino 2005)

use visual notations to represent processes. Many also provide support tools to automatically generate industry standard code like BPML and BPEL4WS (BPMI 2006) meaning the notations are, in some sense, executable. 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 (Recker and Rosemann et al 2007). Nordbotten and Crosby believe that graphical complexity plays a key role in the usability of any visual language (Nordbotten and Crosby 1999). The main principle of Graphical Complexity states that the complexity of a visual notation should be cognitively manageable. Novices are much more affected by graphic complexity than experts, as they must consciously maintain meanings of symbols in memory. The more symbols there are to remember, the greater the cognitive load and potential errors. Many of these languages, including BPMN, have a very large number of symbols.

A related feature of modelling languages is the use of colour. Winn (Winn 1993) showed colour is one of the most cognitively effective of all visual variables in visual languages. Differences in colour are detected three times faster than shape and are more easily remembered. However, we also understand that colour is one of the most difficult variables to use effectively. If not used carefully, it can undermine communication (Moody 2009). BPMN does not prohibit the use of colour, but just not actively encourage the user to use colours. In BPMN every user can freely choose their preferred colour, so normally they vary from user to user, potentially leading to misinterpretations.

Another interesting characteristic of modelling languages is how readily the permit users to hand draw a model without using a software tool support. In the business process modelling area, this can be important as process models are typically developed in an interactive manner by sketching on whiteboards or paper. In fact, the original BPMN charter (BPMI 2010) stated that "If small BPMN processes cannot be easily jotted down by a business analyst on a blank piece of paper, then BPMN will not be successful."

Diagram complexity management is often measured by the ability of notations to present large amounts of information without overloading the human mind. Citrin has described diagram complexity management as one of the most intractable issues in visual notations: "a well-known problem is that they do not scale well" (Citrin 1996). The limitation of diagram related elements that can be effectively processed by the human mind is limited by working memory capacity. When the limitation is exceeded, the cognitive overload issue appears rapidly. Moody has pointed out that BPMN lacks effective mechanisms for managing diagrammatic complexity (Moody 2006). The issue of complexity management is not addressed in the BPMN specification, suggesting that it was not considered in the design of the notation. Almost all the examples in the BPMN specification and the accompanying "BPMN by Example" document exceed cognitively manageable limits (Moody 2009).

Conceptual integration refers to the overview modelling capability for a visual language. Kim described it as the one of the most important mechanisms for any visual modelling languages, which provide a "big picture" view of the domain being modelled (Kim etc. 2000). It acts as an overall cognitive map into which information from individual diagrams can be assembled. Many existing process and service modelling languages, including BPMN, lack such overview presentations of services and service organisation.

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. There are many commercial tools available for business process modelling and simulation. However, despite their increasingly sophisticated functionalities, there are still obstacles to their widespread use (Ali 2009; Baeyens 2007). A common issue is a conflict between usability and flexibility. Typically, the more flexible functionality a tool intends to provide, the more difficult the tool will be to use (Anderson and Apperley 1990; Baker 2002).

Earlier work (Anderson and Apperley 1990; Phillips 1995; Li and Phillips et al 2004) on modelling complex user interfaces and their behaviour, based on the Lean Cuisine+ approach, demonstrated that a tree-based overlay structure can effectively mitigate complexity problems such as those noted above. Lean Cuisine+ (Phillips 1994a; Li and Phillips et al 2004) is a graphical notation based on the use of tree diagrams to describe systems of menus. A menu is viewed as a set of selectable representations (called menemes) of objects, actions, states, parameters and so on, which may be logically related or constrained. It uses an overlay structure for specifying the underlying behaviour of event-based direct manipulation interfaces (Phillips 1994b). The success of the Lean Cuisine+ approach motivated us to adopt a tree and overlay approach to mitigate the common complexity issue and cobweb/ labyrinth problems in current business modelling notations and to fully address the modelling requirements summarised above.

From our motivating example, informal discussion with some target end users, and associated literature review, we identified the following key research questions we wanted to answer in this research:

- RQ1: can a tree-and-overlay based service modeling approach provide advantages over existing box-and-line based visual business process modeling languages?
- RQ2: can an effective and efficient environment be implemented for modeling, checking and generating service orchestrations from these tree/overlay-based models?

3. Our Approach

To answer these research questions we initially informally interviewed several target end-users from our University's service provision team, several industry contacts working in the BPM and automated ERP system generation spaces, and several experienced developers of web service-based systems who were users and composers of services, vs service developers. In addition, we identified the following set of requirements for a tree-and-overlay based service modeling and support tool:

a) Non-programmer end users, such as business process domain experts, need to be able to efficiently model distributed complex systems and related collaborations from a set of pre-existing services;
b) Users need to be provided with multi-level abstractions to assist in defining different service-based process specifications;
c) Automated checking of service composition specifications within the tool is needed, using appropriate formalisms and feedback;
d) The notation must be integrated effectively with other modelling approaches e.g. BPMN;
e) The tool must provide automatic generation from visual models to industry standard code e.g. BPEL scripts;
f) The tool must provide large scale diagram support and interactive visualisation and debugging of generated service compositions

We then iteratively designed, developed and evaluated the EML and its supporting environment MaramaEML. We identified our target end users as business process modelling domain experts and service composition experts that have some IT knowledge, but not necessary very detailed programming and service implementation knowledge. However, as with other process modelling languages such as BPMN, very technical end users, such as service implementers, are envisaged to also be able to use the approach. EML uses the concept of a hierarchy of services and service categories to organise a large number of organisational services (e.g. see Figure 2). It uses the concept of multiple overlays over this tree to define business processes that compose and orchestrate these constituent services (e.g. see Figure 8).

The iterative design/evaluation approach we employed comprised two methods: 1) use of the Cognitive Dimensions framework (Green and Petre 1996) to inform our design and to iteratively improve it and 2) use of a formative end user evaluation once a preliminary toolset had been developed to identify weaknesses that required improvement. In both cases the results were useful in refining the language and toolset design. We elaborate on these design methods below

3.1 Initial EML and Tool Design using the Cognitive Dimensions Framework

The CD approach (Green and Petre 1996) is a popular, psychologically based, heuristic framework designed for quickly and easily evaluating a visual language environment. It sets out a small vocabulary of terms designed to capture the cognitively relevant aspects of structures, and shows how they can be traded off against each other. We used the CD approach to help design EML and to decide on features needed in its support tool. The three authors were the experts who carried out this evaluation.

We initially defined a set of prototype EML constructs and associated tool designs from the set of initial requirements we identified as above. We then used various dimensions from the CD framework to assess their characteristics. We took each key EML construct and supporting tool feature and assessed them in turn. We made a number of trade-offs between different dimensions as we refined the notation and tool support features. Two examples of the way in which we used specific CDs for refinement are:

Abstraction Gradient (types and availability of abstraction mechanisms). To reduce the abstraction gradient, we decided to use a tree layout as the basis for all service modelling situations (other than the integrated BPMN modeller). We reasoned that a solid tree layout would be easy to understand and thus provide minimal abstraction gradient for our target EML end-users.

Diffuseness (verbosity of language). Our original EML design was quite verbose, including a variety of notations and formalisms. After our initial CD analysis we reduced the number of notational elements significantly, and eliminated a form based metaphor included in the original design. The tree overlay became the dominant metaphor. It uses a relatively terse set of language elements and hence is easy to learn. Further details of this preliminary CD-based design work can be found in (Li, 2010).

3.2 Formative User Feedback

After we completed an initial version of MaramaEML, we invited several Computer Science and Software Engineering students to carry out a task-based formative evaluation of the refined EML and the MaramaEML prototype. Our objective was to assess how easy it is to learn to use EML and its support tool and how efficiently it could solve the diagram complexity problem. This was a qualitative evaluation and the results were

used to refine the tool's design. Although this version of MaramaEML was functionally near fully featured, the icon definitions were quite simple and it lacked some user interface "niceties". However, the results of the evaluation were promising.

Method

The main functions of the version of MaramaEML evaluated were:

- EML service tree and process overlay modelling ability.
- Basic BPMN modelling ability (to both compare it to EML but also ensure support for BPMN itself)
- BPEL code generation from process overlays

EML and MaramaEML were briefly introduced to the participants who were then asked to perform several predefined modelling tasks. The tasks were divided into three difficulty levels: simple, medium and complex. Participants 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). In the pen and paper tasks, users were asked to use a black pen to draw the basic tree structure, a green pen to add the process overlay, and blue and red pens to represent task and trigger overlays on top of the tree (details of these are in the following section). The process was observed and each participant was interviewed at the end of the evaluation. The qualitative feedback was then analysed and key limitations and possible solutions identified.

Results

Six Computer Science and Software Engineering students used the tool and carried out these tasks. All six users completed all the tasks. For the simple modelling task, three out of six users found the pen and paper based modelling approach was more efficient than using the software modelling tool. However, for medium and high complexity tasks, all the users found the software tool was better than pen and paper.

One strong finding was that EML and MaramaEML were very straightforward to use and understand. Users found the tree overlay method reduced the complexity of business processes compared to using only conventional BPMN views. They also found the multi-view support to be a useful approach to enhance the modelling strength.

Several limitations of both EML and MaramaEML and several potential improvements were identified. These included a need:

- for a more detailed mapping traceability between EML and BPMN views
- to provide an integrated environment for the two modelling languages (EML and BPMN)
- to improve visual quality of both EML and BPMN views
- to verify generated BPEL code to guarantee its execution quality
- to enhance diagram scalability (coping with complex and large modelling diagrams)

3.3 Refinement

Based on the formative evaluation feedback, we developed a revised version of MaramaEML, which addressed the identified limitations. This including, for example, addition of the following features:

- A text based log file was created to enhance system traceability
- All modelling views (EML and BPMN) were integrated into the same development environment
- A Labelled Transition System Analyser (LTSA) engine was integrated to verify generated BPEL code. If the code passes the validation, an extra LTSA graphical view of the system is constructed
- New icons and layouts were developed to improve the visualisation quality of EML and BPMN
- Zooming and fisheye view functions were added to enhance visualisation scalability

These and other features of both EML and MaramaEML are described in detail in the following two sections.

4. EML

EML models primarily use a novel tree overlay structure: complex business systems are represented as service trees and business processes are modelled as process overlay sequences on the service trees. In this section we describe the core service tree representation, the overlay mechanisms, including how to represent exceptions, dependencies and other enhancements to the basic overlay approach.

3.1 Service Tree Structure

EML uses a tree layout to represent the basic structure of a service. We chose to use trees as they are familiar abstractions for managing complex hierarchical data for business modellers and business people; they can be easily collapsed and expanded to provide scalability; they can be rapidly navigated; and they can be over-laid by cross-cutting flows and representations of concerns. All the Services, sub-Services and Operations are organized in a hierarchical tree structure to model the system. Connections between the three types of component reflect the functional relationships between them. This provides a service taxonomy that we have found very effective for large numbers of services. Depending on the technology used to realise the services, this can also be used to organise both sub-groups of operations and services into various different groups. Typically the same end users develop the service tree, or sub-trees, as those who develop process flows. However, different end users may be responsible for each.

The basic modelling rules are:

- An Enterprise system must have at least one Service Tree.
- Every service tree must have only one Service node. It may (or may not) include an arbitrary number of sub-Service nodes.
- A Service node is always at the top of the single service tree structure. It must include at least one Operation node (directly or indirectly). It may include an arbitrary (possibly zero) number of sub-Services.
- A sub-Service is contained inside a Service or sub-Service node. It must include at least one Operation (directly or indirectly) and may have an arbitrary (possibly zero) number of sub-Services.
- An Operation is the leaf node of the service tree that is used to carry out a discrete task. It cannot include any Service, sub-Service or other Operation.

A Service is a configuration of technology designed for organizational networks to deliver which satisfies the needs, wants, or aspirations of customers (Feng and Lee 2010; Hanna 2002; Hudak 1989). In EML, a *Service* is a compound operation group that is defined by a list of other activities (sub-services or operations). A *Sub-service* is a graphical object within a service tree, but it also can be "opened up" to show another sub-service grouping. Services and sub-Services share the same shape notation in EML, a small circle. The user can pre-define different colours to distinguish different groups of services / sub-services. All the services and sub-services also have an open centre so that pre-defined EML enhancement function icons can be included within the shape to help identify extra functions (e.g. Elision, Reuse etc.). The name of the service / sub-service is placed outside the circle boundary and positioned arbitrarily around the notation (normally at the bottom or right side of the service / sub-service node). In EML a service has five different execution states, they are:

- *Un-executed / Skipped* (the default state, when the service is not invoked or is skipped)
- *Finished* (the service has completed successfully)
- *Failed* (errors were detected when the service was last executed)
- *Aborted* (the service is killed in the middle of execution)
- *Other* (other unexpected states)

An Operation is an atomic activity that is included within a Service. An Operation is used when the function in the Service is broken down to a finer level of Process Model detail. Operations are the leaf nodes of the Service tree. A square shape (with orthogonal corners) represents an atomic operation inside a service (the operation and service are connected by a tree branch). The user can use different fill colours in operations to distinguish different operation groups; a light grey is used by default. The Operations also have an open centre so that EML enhancement function icons can be included within the shape to integrate other functions (e.g. Exception Handler). The name of the Operation is placed outside the rectangle boundary positioned arbitrarily around the notation (normally at the bottom or right side of the node). A normal Operation square is drawn with a single thin black line. But in certain circumstances (e.g. Single Loop Operation, in Process overlay, in Dependency Trigger etc.), EML changes the boundary style to represent additional information.

Figure 2 shows a complex, fully expanded overview of an EML tree modelling a Travel Planner service. A Travel Planner is a web service-based enterprise system to help users to organize trips. Customers use the client application to submit itinerary enquiries to a travel agent. The agent service receives the requests and communicates to travel providers (Airline, Hotel etc.) to find out a suitable booking.

- A *Travel Planner* is a Service node in the tree. It has three sub-Services (*Customer Service*, *Agent Service* and *Provider Service*).
- There are six Operations inside the *Customer* sub-Service *(Send Book Request, Consider Itineraries, Send Confirm Information, Make Payment, Cancel Booking and Receive Invoice)*.
- The *Agent* sub-Service includes another three sub-Services (*Prepare Itineraries*, *Payment Control* and *Product Booking*) and two Operations (*E-mail Out* and *Print*). There are also lists of different Operations or

sub-Services in the above three sub-Services.

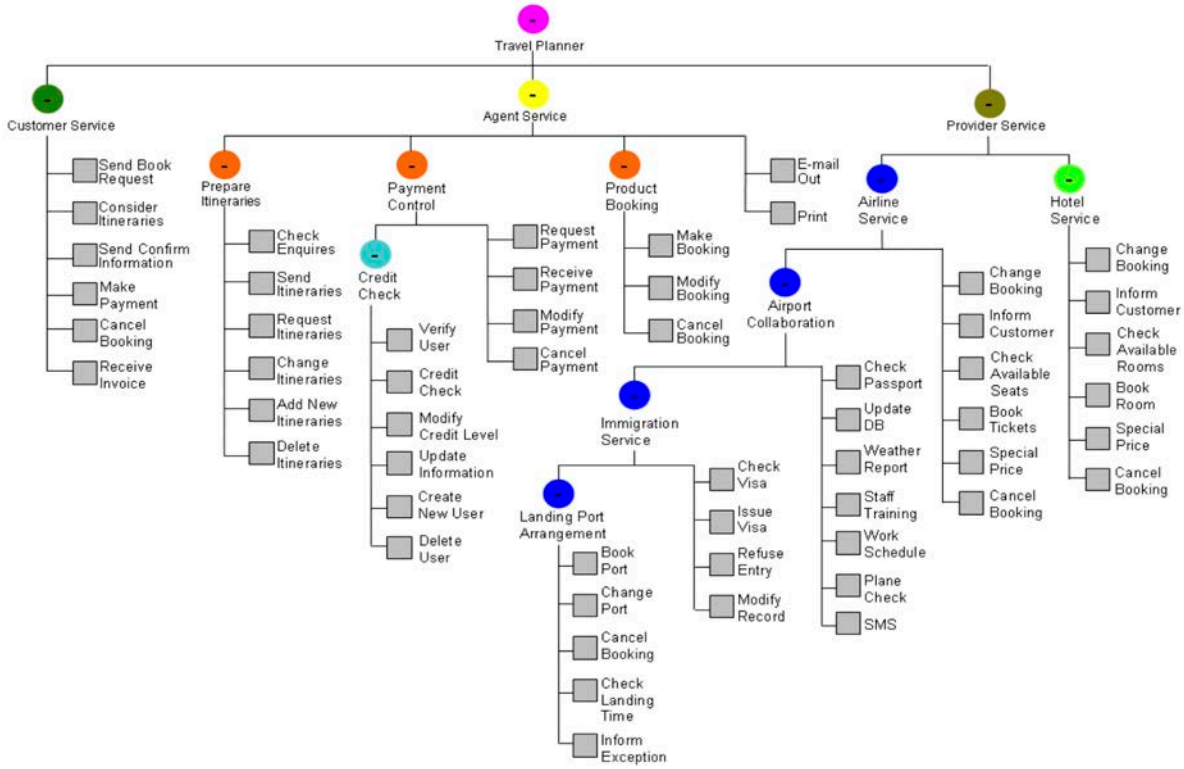

Figure 2: Example EML Tree Structure

The *Provider* sub-Service has a multi-level hierarchical sub-Service tree structure. It has *Airline* and *Hotel* sub-Services at the first level, and the *Airport Collaboration* sub-Service is structured under the *Airline* sub-Service (level two). The *Immigration* sub-Service is inside the *Airport Collaboration* sub-Service at level three. It also includes another bottom level sub-Service *Landing Port Arrangement*. There are twenty-eight Operations involved in this multi-level sub-Service tree.

3.2 Elision

In order to manage diagram complexity, symbols inside each service identify the elision level of the service visualisation. With a tree-based visualisation, a collapse/expand elision mechanism is a natural way to provide complexity management. Most users are familiar with such an approach due to its commonality in graphical user interfaces and desktop user interfaces. In EML a Service or sub-Service can be in a collapsed (elided) mode that hides its details or in an expanded mode that shows its details within the view of the service in which it is contained. The collapsed and expanded forms of the Service / sub-Service objects are distinguished by two markers. A minus (-) symbol indicates all activities in the service have been expanded (Product Booking service node in Figure 4). A plus (+) symbol indicates that part or all of the sub-tasks (services and operations) are elided (Payment Control service node in Figure 4). The elision function only applies to the Services or sub-Services and cannot be used on Operations.

3.3 Service Reuse

EML supports service reuse to reduce structural complexity and to increase modelling efficiency. In EML we chose to represent reusable components using a separate tree. This preserves the overall approach of tree-based decompositions adopted for EML and allows one service tree to reuse elements in another, reusable service tree. The user pre-defines its structure and saves it in a library. Reusable components have a unique ID for future usage. The user can easily attach a reusable component to any branch of an EML tree. The reusable services share the same attributes of Service / sub-Service. But their "*Reuse Status*" is "True" and "*Reuse ID*" is a unique number beginning with "R" (e.g. R1). If a reusable service is attached to the EML tree branch, the "*Elision Type*" attribute will be automatically set as "*Collapse*". However, the user can change it to "*Expand*".

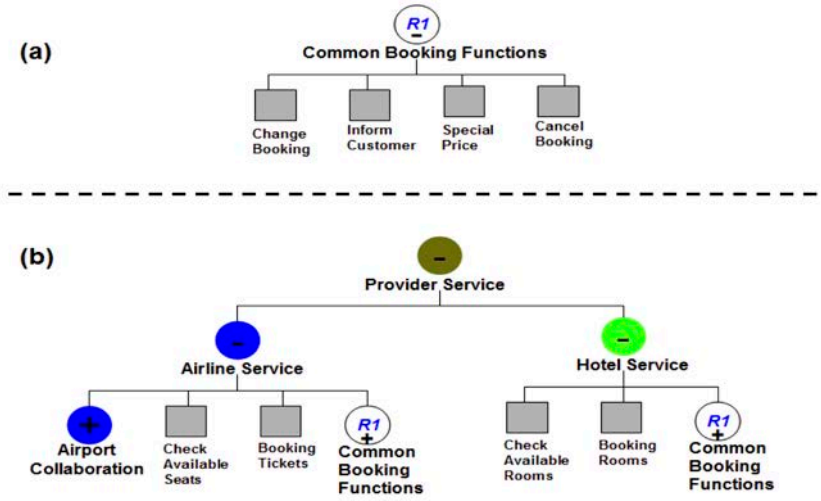

Figure 3(a): Define a Reusable Service; (b): Use Reusable sub-Service in Hotel Service

Figure 3 shows a Reusable Service's definition and usage. In the *Travel Planner* Service Tree of Figure 2, we see that the *Airline* and *Hotel* sub-Services have several identical operations (*Change Booking, Inform Customer, Special Price* and *Cancel Booking*). To reduce redundancy, we combine these as a Reusable sub-Service *Common Booking Functions* shown in Figure 3(a). The *Reuse ID* at the centre of the circle is *R1*. Figure 3(b) demonstrates usage of the Reusable Service in the *Hotel* and *Airline* services. The user directly attaches *R1* service node to both tree branches. The elision status of this "*Common Booking Functions*" service has been changed to "*Collapse*" automatically.

3.4 Process Overlay

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. By modelling a business process as an overlay on the service tree, the designer is given a clear overview of both the services structure and the business process simultaneously. Processes can be elided mitigating the cobweb problem commonly existing in flow-based visual notations. In EML, a Business Process may contain more than one separate sub-Process. Each Process may have its own Sub-Processes or share (Reuse) sub-Processes with other Processes. The individual sub-Processes are independent, but could have data connection with others. In order to enhance readability and reduce diagram complexity, EML does not use "Data Flow". All the data communicated between services, operations and processes are encapsulated in the flows, service nodes and operation nodes. By using this mechanism, the user can focus on the business process itself while complex data details are hidden behind it. However, the user can always obtain (and show) this information from the notation's data attributes. It provides a more flexible and clearer view to the business processes and service structures.

Figure 4 shows a *Travel Booking Process* in an EML process overlay. Only process related services and operations are shown; other, unrelated services have been elided (e.g. *Payment Control Service, Airport Collaboration Service*). The process starts (blue rectangle) with a client side application (not shown) passing a request message to the *Send Book Request* operation of the *Customer Service*. The *Agent Service* receives the request through the *Check Enquires* function, and uses its *Request Itineraries* operation to check availability information with the *Airline* and *Hotel services*. The agent requests flights and rooms with a list of parameters. There are iterations (dashed double arrowheads links) between *Request Itineraries, Check Available Seat*s and *Check Available Rooms*. When the agent finds that both the air ticket and the hotel room are available on the requested date (end condition C1 & C2), it terminates the loop and sends the client a report generated by the Send Itineraries operation. The customer *Considers Itineraries* and *Sends Confirm Information* to the *Agent Service*. The agent receives this information and then *Makes Booking*. After both *Book Tickets* and *Book Room* operations are successfully completed, the agent calls *Make Payment Process* (Figure 6) to ask for the payment and end the existing process (Rounded Rectangle).

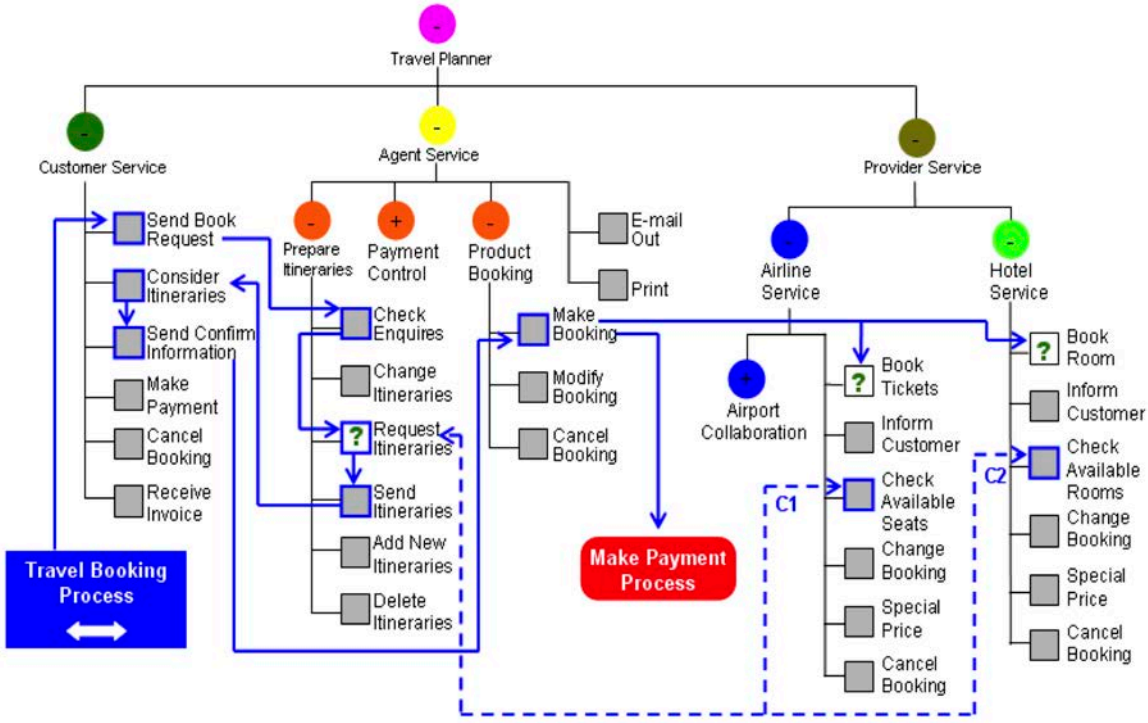

Figure 4: Travel Booking Process Overlay

The double-sided arrow in the Process Start indicates that the process starts with a condition. In this example, it is the arrival of a *Booking Request Message*. This message condition is hidden inside the Process Start notation. The process name inside the Process End notation means that the successful end of this process will lead to the start of a new process (*Make Payment Process*). All the process sequence IDs have been hidden since there is only one process in the diagram. However, the user can make them visible especially when more than one processes appear simultaneously. There are two types of execution state included in this example:

(1) The finish status (by default). In this state, the boundaries of the operations have been changed to a blue colour. It means that this process step is completed successfully.

(2) The failed status. In this state, a green question mark appears in the centre of the operation shape (e.g. in *Request Itineraries, Book Tickets and Book Room* operations). It means that the process step with this operation has not completed normally.

The concept of service "failure" in EML indicates that additional flow(s) are present to handle such situations. The concept of "fail" is used to represent a number of possible situations that require handline, such as an explicit failure return payload e.g. "no rooms available for these days" or empty list/document, invalid or unexpected response from a service, failure to return from service invocation (timeout), etc.

3.5 Exception Handler Overlay

Another type of overlay, an exception handler, specifies what happens if an exception or fail condition changes the normal process flow. A green question mark annotation in the middle of an operation or service indicates an exception handler. Figure 5 shows a hotel room booking exception handler layer. Instead of simply calling *Cancels Booking* and rolling back, we use the EML exception overlay to model a more complete exception solution. If the *Hotel* finds that all standard rooms on the required date have been booked out, it sends a negotiation message (*Change to Luxury Room*) back to the travel agent. The *Agent Service Modifies Booking*, *Changes Itineraries* (previous travel plan), makes a new itinerary and sends to the *Customer Service*. The customer receives the latest updated travel plan and *Considers Itineraries*. When the customer makes a final decision, the *Customer Service* then *sends Confirm Information* to the *Agent Service* again. If the user *Accepts* the hotel's suggestion, the process will lead to *Make Booking* again. Otherwise (*Refuse*), the customer informs the agent to *Cancel Booking* and the *Agent Service* asks the *Hotel Service* to *Cancel Booking*.

The green diamond icon after *Send Confirm Information* is used to represent the conditions. The *Accept* decision is a default option. A dot shape attached at the start of the Accept exception flow is used to represent the default attribute. Since the *Refuse* decision is an alternative path, the result (*Cancel Booking*) of this

exception flow remains in the Un-executed status (no border). Meanwhile, the default decision result (*Make Booking*) is in the "Finished" status (green border).

There are another three exception handling icons in *Special Price* and *Request Itineraries* operations and *Airline Service*. The user has chosen to define these in different layers, but they could be combined with *Change to Luxury Room* in the same layer if desired. For a particular tree node, the user can define several different exception layers distinguished by their start conditions. So in EML, exception handling is much more than a simple roll back mechanism. It is an individual process or even a complicated integration of alternative processes. The exception handling overlay is an individual overlay based on the same service tree structure as conventional processes. EML has the freedom to allow the user to define them in a single layer or combine them with the process and trigger overlays described next to generate an integrated overview of the system.

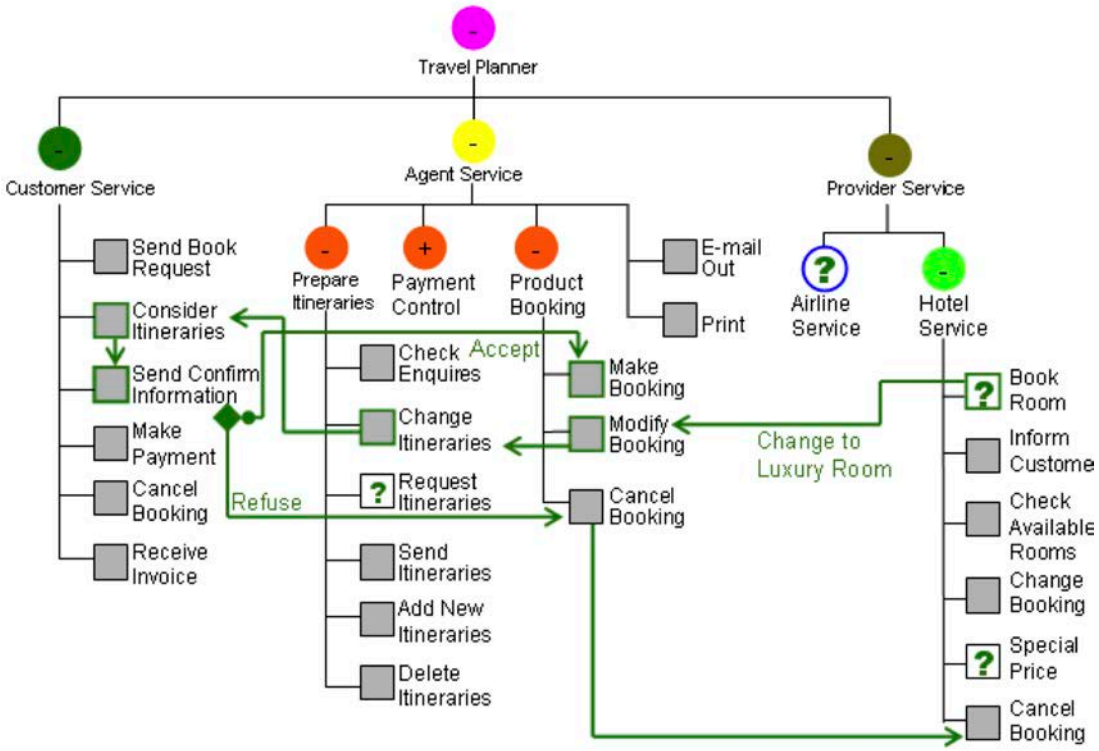

Figure 5: Hotel Room Booking Exception Handler Overlay

3.6 Dependency / Trigger Overlay

An EML trigger overlay is used to model internal system dependencies. Since EML uses a multi-layer structure, users can choose to combine trigger layers with process layers or separate them by using different views to reduce diagram complexity. The major difference between a process and a trigger is the actor involved. A process is performed by a user; the sequence and result of a process are normally variable. A trigger is enacted automatically by the system itself. As it is an internal system dependency, the trigger execution order and outcomes (for the same trigger condition) are usually unalterable. Thus another benefit of the trigger overlay is to support internal dependency process reuse. Once a trigger is defined, it can be reused in different processes and exception handlers.

Figure 6 illustrates the *Make Payment Process* example with trigger flows. It follows the *Travel Booking Process* example from Figure 4. When the user successfully books air tickets and hotel rooms from *Travel Booking Process*, the *Agent Service* starts to *Request Payment* using *Payment Control* service. The agent sends the payment request to *Customer Service*, and the customer then *Makes Payment*. If the agent *Receives Payment* within three days (default option), then it sends the invoice by *E-mail* to the customer. The customer *Receives the Invoice* and the whole process ends. However, if the agent *Payment Control* service doesn't receive the payment in three days, it *Cancels the Booking* using *Product Booking* service.

This operation triggers two extra operations automatically: *Cancel Booking* in *Airline Service* (T1.1) and *Hotel Service* (T1.2). A "triggered" operation is used to indicate additional, cascading operations, often run in parallel, augmenting the process much like an "extends" relationship in use case modelling. Triggered operations can become large additional flows and we use this construct to help manage flow complexity in EML overlays. In this example, we integrate the trigger with the process overlay. However, the user can hide the

trigger and define it in a different layer to reduce diagram complexity. The *Cancel Booking* trigger itself is reusable. For any other processes an exception handler or trigger can be directly reused by linking the flow to the operation and filling in a trigger start condition.

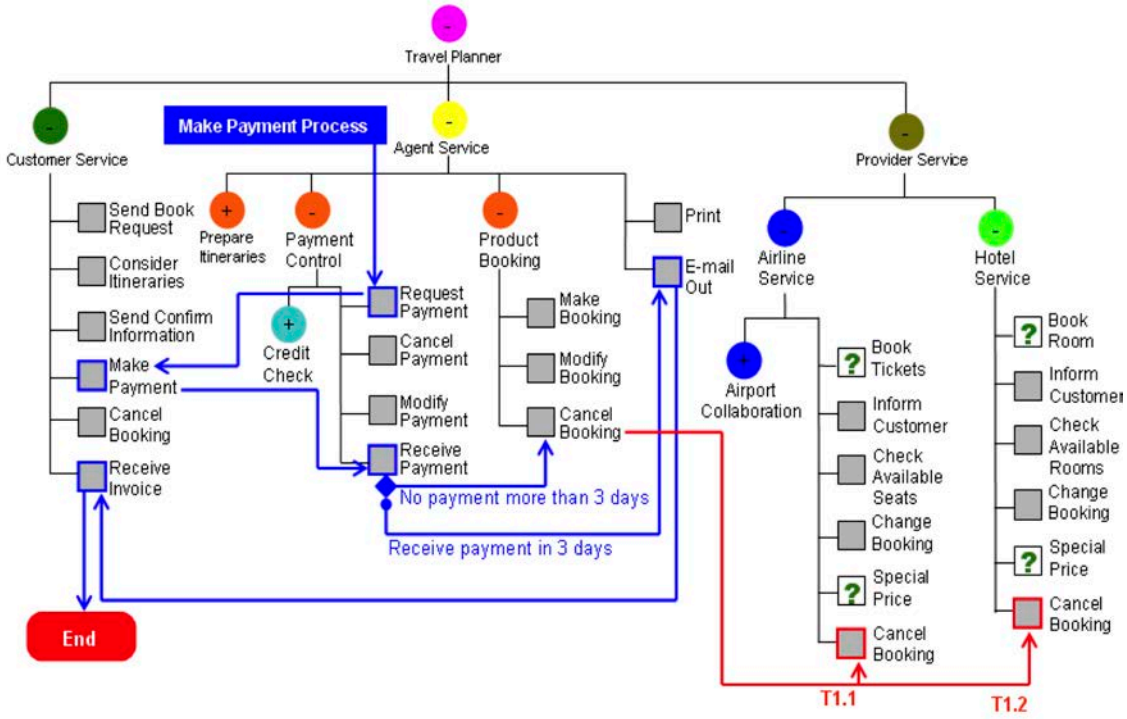

Figure 6: Make Payment Process with Triggers

3.7 Iteration

We represent iterations occurring in different overlays by using the same visual method (styled lines), but use process specific colours to distinguish them. This increases the consistency of the notation and reduces the modelling complexity of the diagram. There are three types of loops:

1. A single activity loop is represented as a single arrowhead dashed line whose source and target are same operation (or service/sub-service). Attributes in the dashed flow control the iteration (e.g. loop times, start and complete conditions, input/output data etc.).
2. Loops of two operations (or services / sub-services), use a dashed line with two arrowheads. Figure 4 includes the iteration between *Request Itineraries, Check Available Seats* and *Check Available Rooms* operations in the trigger overlay. When the *Agent Service* receives the room and flight ticket booking application from the *Customer Service*, they need to check whether the suitable hotel room and flight ticket are available on a customer required date. The customer normally sends a list of preferred date, room and ticket types for the room booking. The *Agent Service* starts from the highest preference date, room and ticket type to the lowest preference types. The process loops until the termination conditions *C1* (finds the suitable flight on required date or there is no flight available on all the preferred dates) and *C2* are met (finds the suitable room on required date or there is no suitable room available on all the preferred dates).
3. If a loop involves more than two operations (or services / sub-services), a single arrowhead dashed line guides direction, linking different operations or services in a closed circuit.

3.8 Conditions

A condition shape is a diamond. The fill colour of the condition is based on the overlays. If a shape is used in a process layer, it appears in blue (in Figure 6). However, if it is in an exception handler or trigger overlay, then the colour becomes green (in Figure 5) or red. All conditions icons are filled to indicate conventional exclusive if/then/else type semantics, or may have an open centre indicating other conditional semantics e.g. OR, AND, XOR or OTHERS.

4. MaramaEML

The MaramaEML environment allows designers to model with EML and generate Business Process Execution Language (BPEL) orchestrations of web services from EML models (Li et al 2008). MaramaEML is implemented using our Marama meta-tool (Grundy et al 2008; Grundy et al 2013) as a set of Eclipse plug-ins, providing a robust, scalable design tool. It integrates EML and BPMN modelling with BPEL code generation and LTSA checking. Performance simulation (Grundy et al 2006) is also incorporated in the integrated EML support environment, facilitating cost-effective tests of the integrated specifications using random data and visualisation of test results using the same design-level specification views. We use the university enrolment system example of Section 2 to illustrate MaramaEML's major functionalities and also the application of EML to another substantial example.

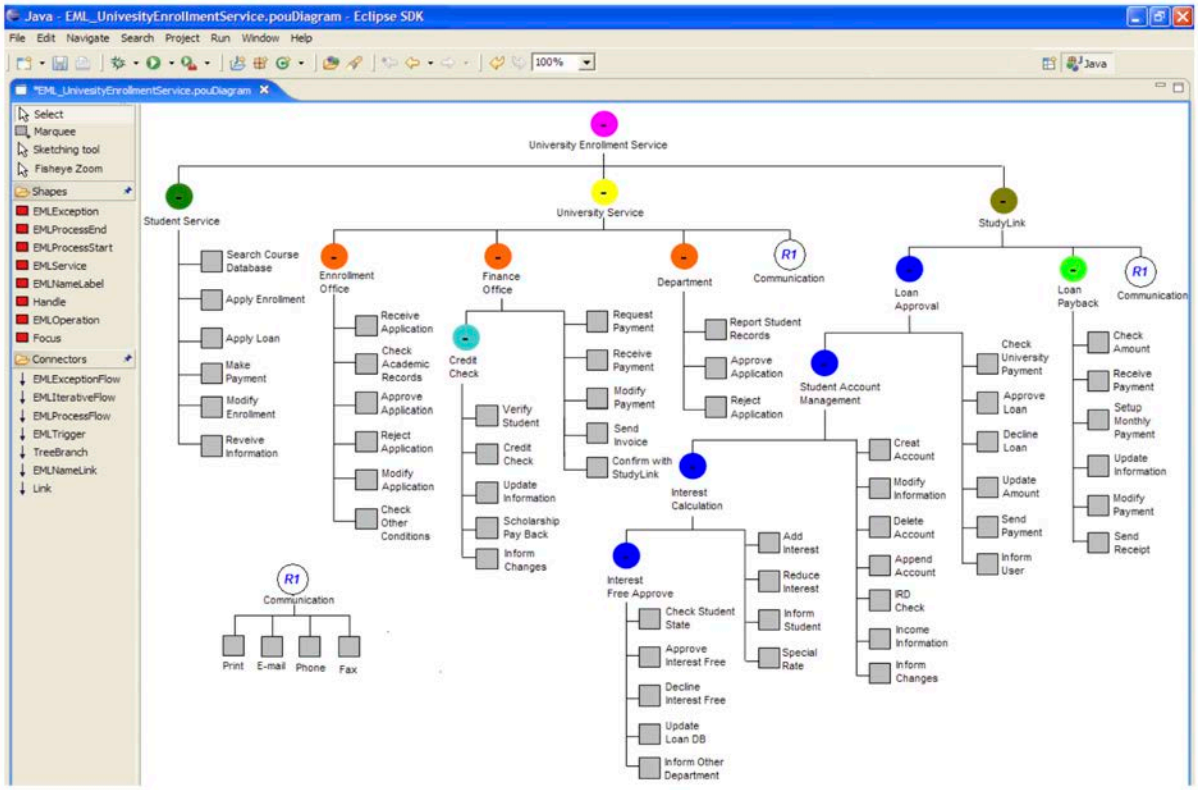

Figure 7: University Enrolment System Overall Structure

4.1 Service Tree Modelling

Figure 7 shows a complex, fully-expanded overview of an EML tree modelling the university enrolment service using MaramaEML. The student service, university service, and StudyLink are sub-services (represented as ovals) of the university enrolment service. The university service includes five service groupings (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. MaramaEML supports EML's service reuse notation. In Figure 7, the *Communication Service*, defined as a reusable component at bottom left), is reused by the *University Service* and *StudyLink Service*.

4.2 Overlay for Processes, Exceptions and Triggers

In EML each business process is represented as an overlay on the basic tree structure or an orchestration between different service trees. For example P1.1 to P1.17 in Figure 8 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. Involved operations or services have bold outline borders to help 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 a payment from StudyLink, the finance officer confirms the enrolment and *Sends the Invoice* to the student.

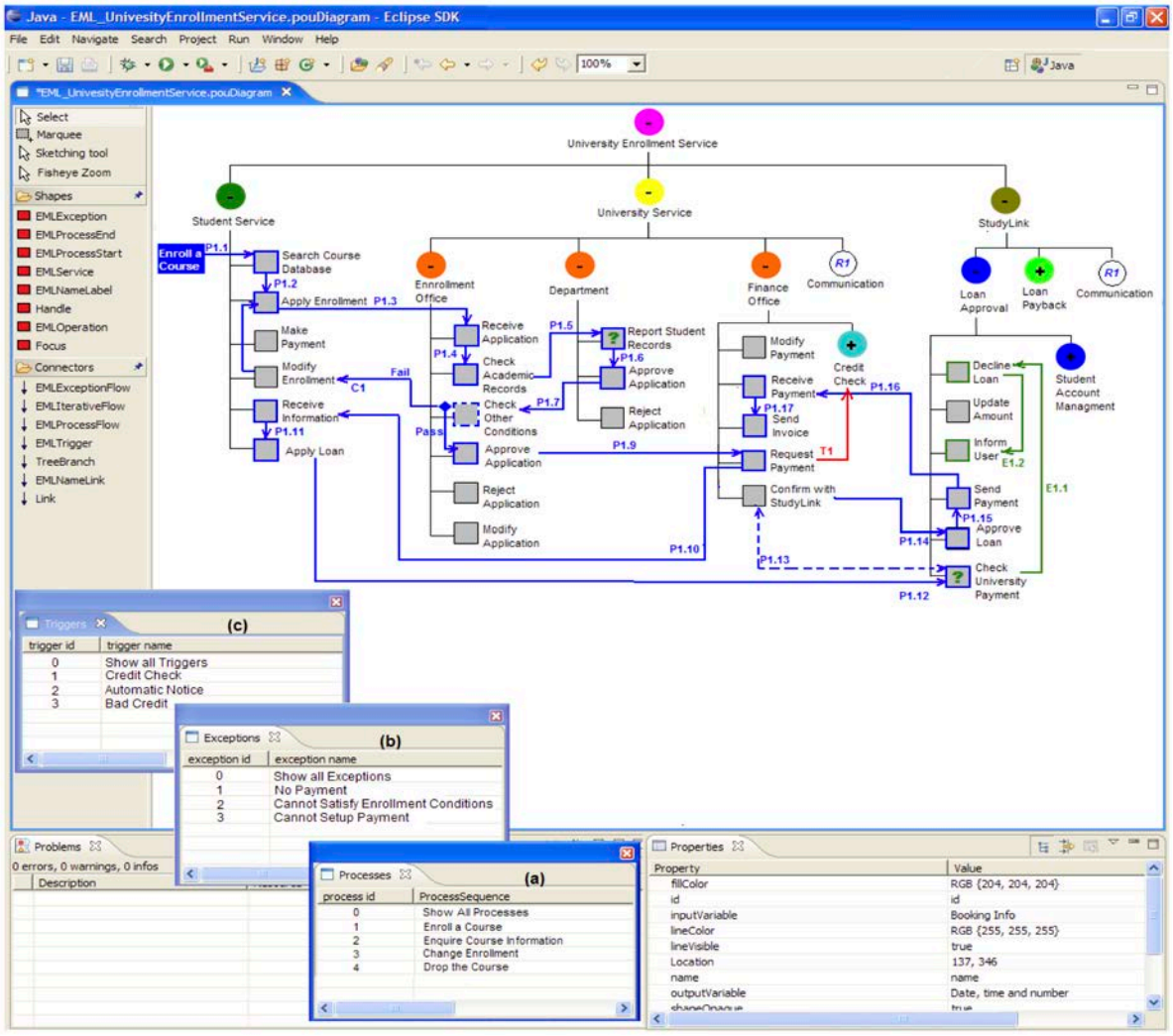

Figure 8: Using EML Overlays to Model the Enrol in a Course Process

In a process layer, users have the choice to display a single process or collaboration of multiple processes. In MaramaEML, a user can selectively show/hide EML Processes (Figure 8 (a)), Exceptions (Figure 8 (b)), or Triggers (Figure 8 (c)). Note also the user has elided parts of the tree to allow focus on branches that are relevant to the process displayed. T1 in Figure 8 shows a trigger condition. Here, when the Finance Office *Requests Payment* from the student, they also need to do a *Credit Check*. MaramaEML users can define trigger conditions as attributes, via property sheets, at each end of the connecter to control the dependency. Users can choose to combine triggers with the process layers (as in this example) or separate them, using different views to reduce complexity. Figure 8 also includes two overlaid exception handlers for handling course prerequisite failure (in *Report Student Records*) and failure to confirm loan application in *Check University Payment*. Conditions for these are also specified in property sheets, as are the conditions for conditional flows, such as the one attached to the boundary of *Check Other Condition*.

4.3 BPMN Integration

MaramaEML allows other business process modelling notations to be integrated to collaborate with EML and to facilitate modelling of different structural and behavioural aspects. For instance, in EML, data are bound to process flows via textual properties so as to reduce diagram complexity. However, sometimes a user may require this kind of information to be presented directly in the diagram. BPMN diagrams can represent the

internal flow sequence of data well, but this kind of flow-based approach can easily cause diagram cobweb problems. An ideal solution is to provide the user with access to both diagram types. MaramaEML includes linked BPMN and EML views with consistency management between then i.e. when an item is changed in the EML view, the corresponding BPMN item(s) are changed and vice-versa.

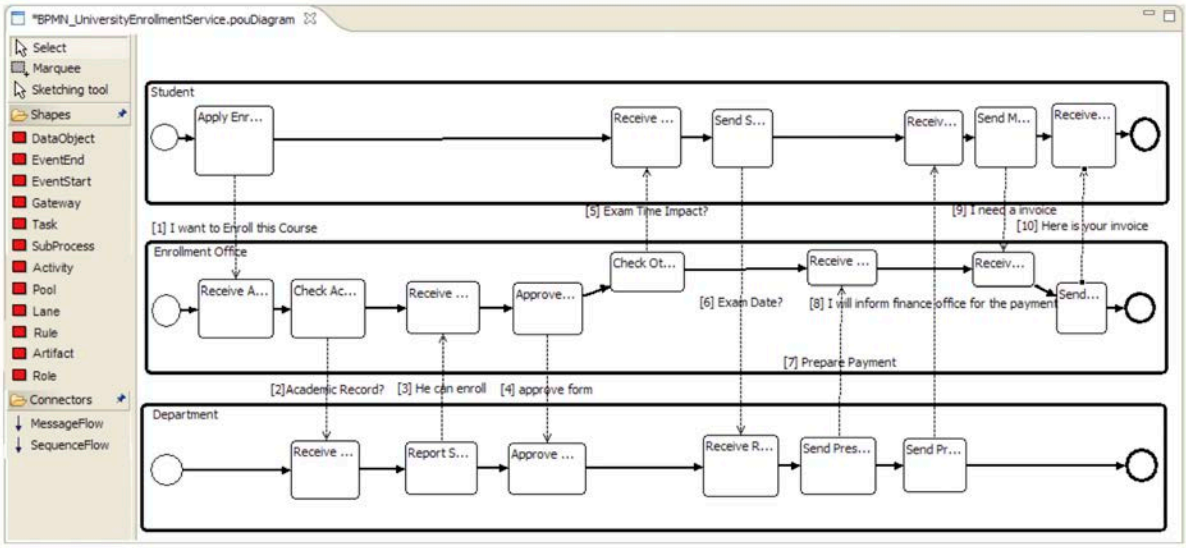

Figure 9: BPMN View --- Enrol in a Course

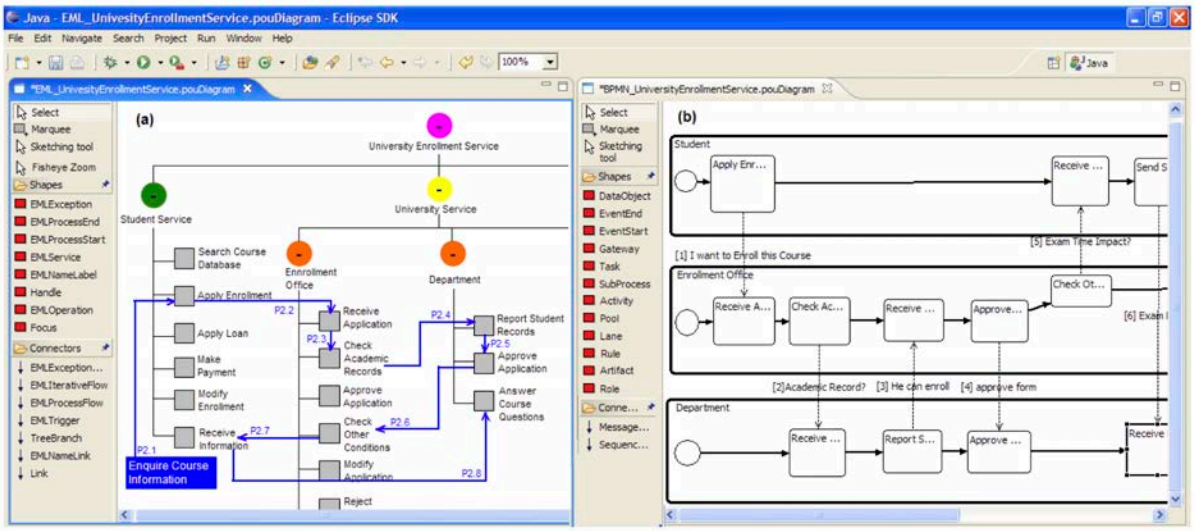

Figure 10: Using EML and BPMN views to model the same process

Figure 9 shows a BPMN view for the "Enrol a Course" process. The *Student*, *Enrolment Office* and *Department* are described in three pools. Figure 10 shows the juxtaposition of the EML view (a) and the BPMN view (b) to model this process. From the EML view the user can obtain a clearer service structural view and an understanding of the process sequence, while from the BPMN view, the user can also see the data transformation. These different domain-specific visual language process notations share a canonical merged model in the MaramaEML tool. Modifications made by the modeller to either notation are reflected in the other.

4.4 BPEL Generation and LTSA Validation

MaramaEML can generate Business Process Execution Language (BPEL) code and coordinate processes in a BPEL workflow engine, allowing users to export and integrate EML process specifications with other BPEL compatible environments. We have also integrated an LTSA-based model checker (Foster and Magee et al 2003) into MaramaEML to verify the correctness of the EML models. As shown in Figure 11, the EML process layer (a) has been automatically compiled to executable BPEL code (b). Our code generator performs model

dependency analysis and maps EML model constructs to structured BPEL activity constructs. The LTSA engine then verifies the correctness of the generated BPEL code. To generate the BPEL code, the user needs to move to the EML tree structure view (a) and use a popup menu (e) to call the "Generate BPEL4WS from EML" function. BPEL code will be automatically compiled and displayed in area (b). To verify the BPEL code, the user needs to change to the "LTSA Perspective" from (h), open the target BPEL code in (b), select the process name (f) and choose an LTS compile function from popup (g). The output from validation appears in (d) and the final LTS view appears in (g). If there is no compilation error, a LTS diagram is presented (c). Errors checked for include incorrect sequencing of web service orchestrations, incorrect parameters, orphaned process stages, non-terminating processes and unreachable processes.

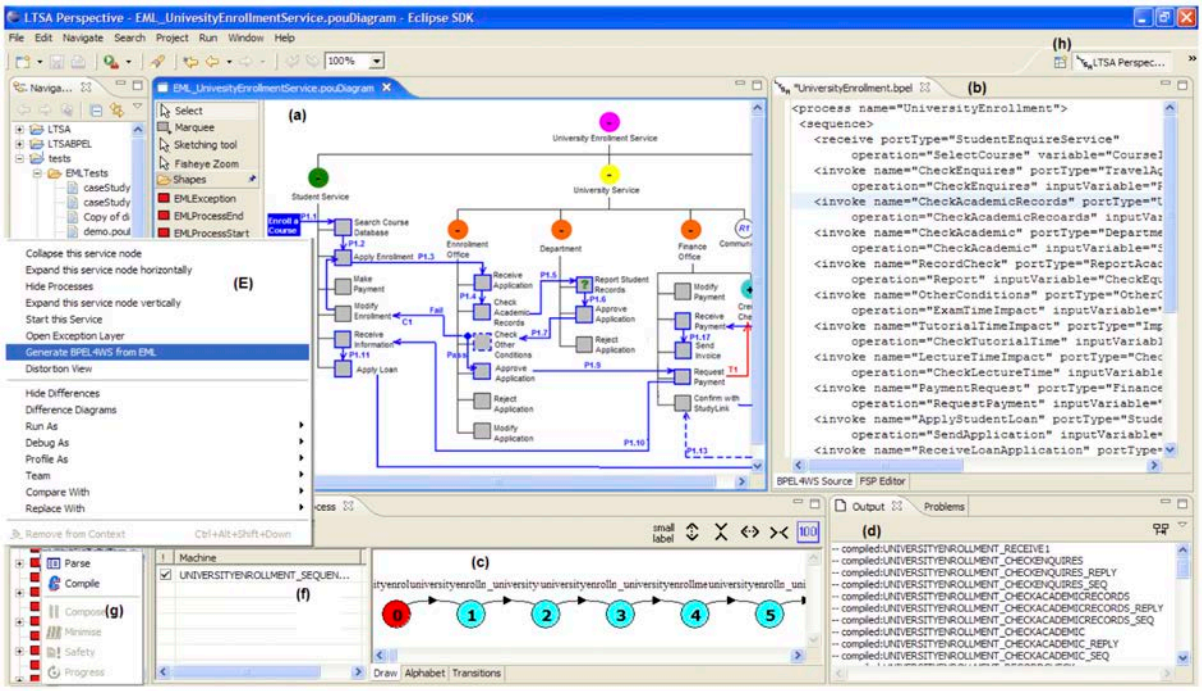

Figure 11: BPEL Generation and LTSA Code Validation

4.5 Zoomable and Fisheye Views

EML's novel tree overlay structure has reduced the modelling complexity at a visual methodological level. However, due to the nature of enterprise complexity, sometimes views can still be very large. At a technical level, to enhance EML's diagram navigability and understandability a zooming (radar view function) and a distortion-based fisheye zooming function have been developed in MaramaEML.

Figure 12 shows a MaramaEML zooming view (a). The user draws a "Radar square" (blue square) in the tree overview area (b). As the user moves the blue radar square, the components in area (a) are panned to focus accordingly. By using this function, the user can have an overview of the whole tree structure, and also be able to navigate to detailed parts. Figure 13 shows a MaramaEML fisheye view (a). The user draws a "fisheye area" (blue square) in area (b). Components in the blue square are represented in area (a) at enhanced size (*Department* Sub-tree), while the rest is distorted with the degree of shrinkage increasing with the distance from the fisheye area. As the user moves the blue radar square, the components in area (a) are moved accordingly into focus. In the example, the starting shrinkage degree is 2, which is much larger than desirable, but serves to illustrate the mechanism. At any stage, the user can change this value by selecting from a pull down menu.

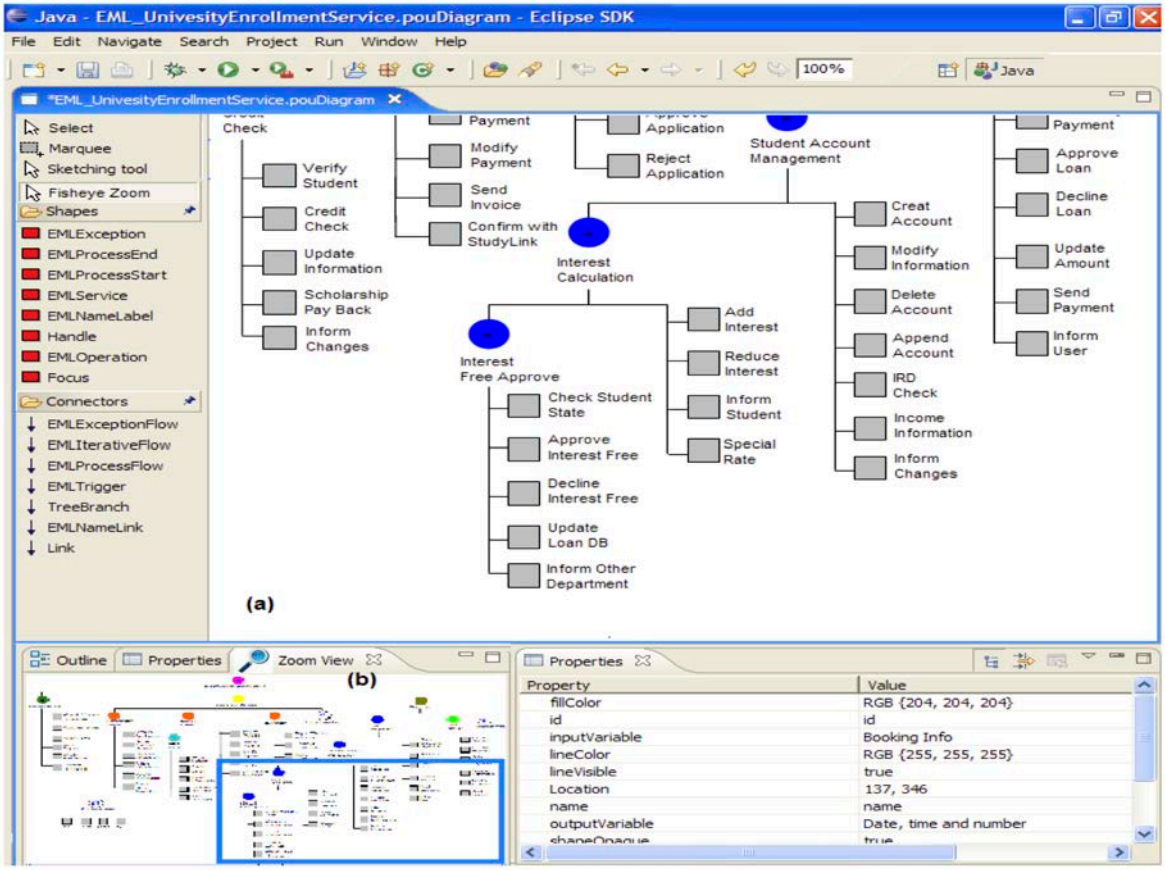

Figure 12: Zoomable View in MaramaEML

5. Implementation

MaramaEML was implemented using the Marama meta-tools (Grundy et al 2008, Grundy et al 2013). Structural aspects of MaramaEML were specified visually using Marama's visual meta-editors. Behavioural aspects were constructed programmatically as Java-based Marama event handlers.

5.1 Service Tree Structure

An event handler for the MaramaEML tree layout was defined for the root/leaf and parent/child shapes to respond to. When a shape is added to a MaramaEML modelling view, the location of the shape is analysed and then corresponding reacting behaviours are executed to automatically layout the shape based on whether the shape is created as a child of a parent shape or is standalone. This event handler catches a *new shape added* event and responds by running a tree layout algorithm. We have developed a algorithm to calculate the vertical and horizontal space between all the nodes in the tree structure. When the user adds a new service or operation node under a service node, the positions of all previous nodes are recalculated and updated to maintain the tree layout; tree branches are rebuilt too. A sub-tree can be moved to a standalone location to become an independent service tree or to a parent shape location to become a migrated child. This is implemented as an event handler to react to a *shape move* event. When a shape is moved, all its subsequent descendants are retrieved and moved together as a whole in reaction to the event.

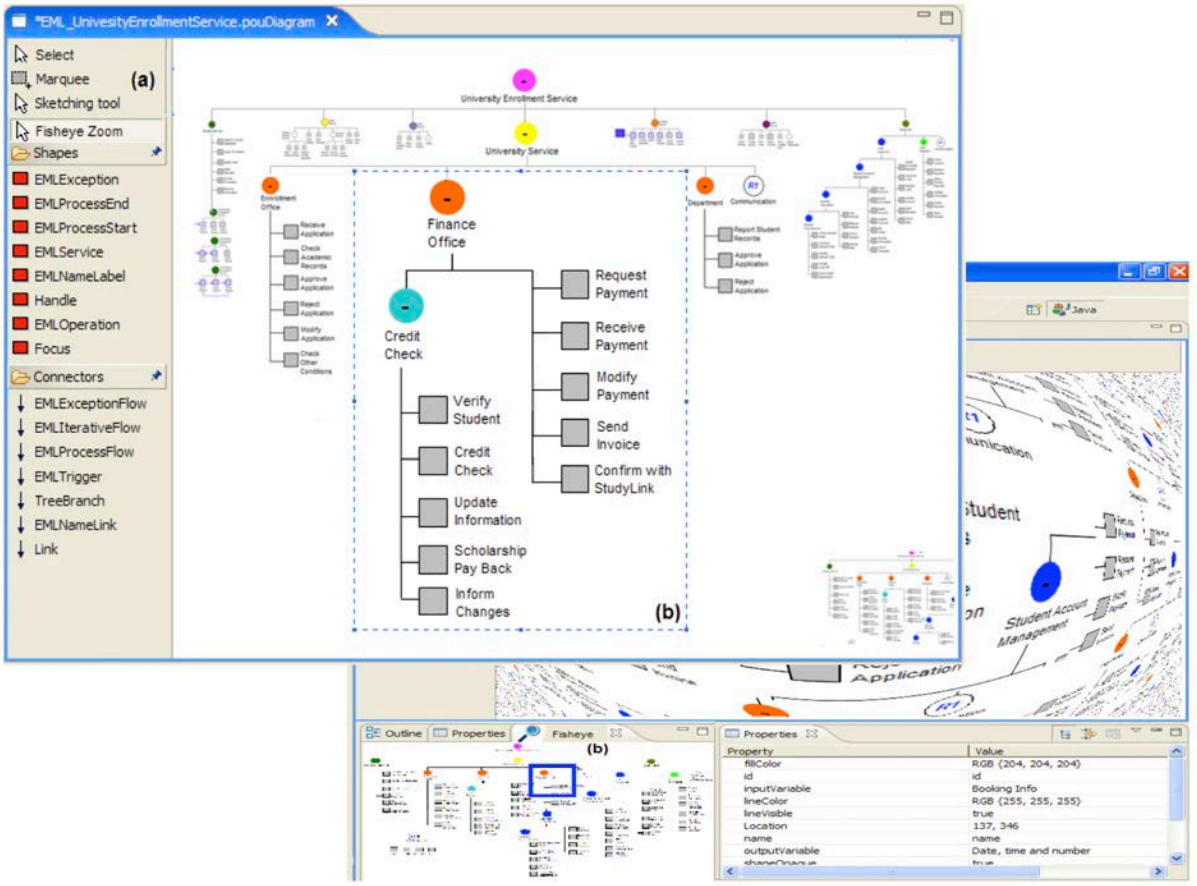

Figure 13: Fisheye Views in MaramaEML

5.2 Overlay

The overlays of process models on a tree structure were firstly defined as normal elements, including the Process Starts, Process Flows and Process Ends as the basis, with Trigger and Exception Flows as complements. Iterations can be defined for each Process/Trigger/Exception Flow via iterative property settings. Multiple process overlays can be modelled in the same diagram, and they are distinguished from one another using unique process identifiers (each process-related element has property identifying which process it is belonging to). Process overlays can be shown or hidden, selected and deleted, to react to the user's interaction. These features were implemented as user triggered event handlers (reacting to right click actions on context menus). In addition, an Eclipse ViewPart implementation called "EML Processes View" provides the user with a more straightforward way (juxtaposed display with the diagram) to view all the processes overlaid in a diagram, while also allowing selection of a particular or a subset of processes to be displayed. In the implementation of process overlays, all the elements of an EML diagram are walked through to identify their notation type and properties. Elements related to process overlays are collected and distinguished using a Hashtable data structure, which is traversed through and analysed to supply on-demand interactive display of multiple process overlays.

5.3 Detailed Properties

In order both to validate trees and overlays, and to generate detailed technical elements of BPEL4WS scripts, many EML constructs have associated properties. These include:

- Service interface names, deployment location and ports, operations, payload types, and some aspects of the Service Level Agreement associated with the service (timeout, alternative location(s), etc).
- Overlay flow sequencing (auto-generated), to and from service location/port, payload details
- Conditional execution and triggered execution properties

Details of these properties and their usage can be found in (Li, 2010).

5.4 BPEL Generation

The BPEL code generation facility is implemented as both embedded runtime behaviour in Marama (by adding to the Marama API) and as a user-triggered event handler. It uses the Marama API calls to query user-defined modelling elements and perform mapping code generation to the file system. The code generation algorithm is straightforward, traversing EML nodes using a Hashtable data structure and analysing the types and properties of the EML elements to permit mapping to the corresponding BPEL code structure. As multiple processes can be defined in one EML diagram via process overlays, multiple BPEL process files can be generated. One-to-one mapping of EML elements to BPEL constructs is typical, for instance, an EML Service maps to a BPEL PortType; an EML Operation maps to a BPEL Operation; an EML ProcessFlow maps to a BPEL Link; an EML ExceptionFlow maps to a BPEL Compensation Handler; an EML TriggerFlow maps to a BPEL Event Handler. The code generator buffers the diagram analysis results, i.e. the XML code snippets contributing to the final BPEL processes definition, and then outputs the completed XML files. Generating complete and executable BPEL code requires additional diagram properties (e.g. input and output data, conditions, error message etc.) to be set via property sheets over and above those needed for basic EML modelling. The generated BPEL is given to the third-party LSTA-based model checker to verify correctness of the BPEL (no orphan states, correct sequencing of web service orchestrations etc) and results of this model checking shown using the LSTA-based model checker's visualisation facilities.

5.5 Zoomable View

Standard zooming functions, including zoom in, zoom out, zoom fit and selection zoom are implemented as toolbar commands on the Eclipse Workbench. To these, we have added an Eclipse PageBookView, which listens to user's diagram selection events, and renders a 'Radar' zoom view for the whole EML diagram. The "zoom" functions zoom in/out the entire EML diagram by a predefined scaling factor. The "zoom fit" function fits all diagram elements in the available screen space with an automatically adjusted scaling factor. The "selection zoom" function allows user to select an area of the diagram and zoom to fit the selected part. The "Radar" zoom view accompanies the EML diagram, providing a thumbnail as well as an indication of visible items inside the screen boundary and those outside of the boundary of the EML diagram. This implementation is integrated with the Marama API. We have provided an additional package inside the MaramaEditor plug-in to manage the zooming functions while still exploiting the existing Marama code base. The package includes zooming interfaces, various zooming actions, Marama diagram mouse trackers, and viewing areas. MaramaEditor is then configured to enable these zooming features using its zoom manager.

5.6 Fisheye View

We implemented a fisheye view function by providing a local context against a global context. This is a focus and context visual technique often referred to as a "distortion based display". Three major attributes control the fisheye function: Focal point, Distance from focus and Degree of Interest (DOI). A point of interest has to be defined which sets the focal point. The "distance from focus" determines the distance from my point of interest (focus) to some point x. Longer distances lead to smaller sizes of the shapes in the distorted display. The implementation of DOI is composed of a static and dynamic component. The static component is either the a priori importance or the global importance of the element relative to every other object in the system. For the user, the global importance is how a tree node is used more than another tree node in the EML diagram. The dynamic component creates a relationship between the user's interest and the importance of an item depending on the latest interactions on the tree. The DOI is assigned to every element in the EML diagram, and a node is selected as the central focus point. It is important to note that if the point of interest changes, then the DOI must be recalculated for every node.

6. Evaluation

As outlined in the Approach section, continuous evaluation has played an important role in the entire EML and MaramaEML design and implementation process using a combination of Cognitive Dimensions based assessments and formative end user evaluations. Recall the research questions we identified from Section 2:

- RQ1: can a tree-and-overlay based service modeling approach provide advantages over existing box-and-line based visual business process modeling languages?
- RQ2: can an effective and efficient environment be implemented for modeling, checking and generating service orchestrations from these tree/overlay-based models?

We have provided preliminary support for an affirmative answer to RQ1 and RQ2 through our formative evaluation. To provide a more summative evaluation addressing these questions, we undertook a formal end user evaluation of MaramaEML in the form of a comparative evaluation. As an example of an existing box and line approach, we chose BPMN and so focused the end user study around a comparative evaluation of EML and BPMN. Combining the comparative aspects of RQ1 and RQ2 leads to two hypotheses, which we tested in our evaluation:

H1: The task performance of users is better using EML than BPMN
H2: Users find EML more usable than BPMN

6.1 Method

Our study had a within subjects design to compare the usability and effectiveness of EML and BPMN followed by additional tasks to evaluate the usability of several EML specific features. To avoid a learning effect, in the comparative part of the study all subjects repeated the same tasks for each modelling language, but with half the subjects using EML first, followed by BPMN, while the other half used BPMN followed by EML. A briefing session prior to undertaking the tasks introduced subjects to the two modelling languages and a target example to be modelled. Following completion of all tasks subjects completed a usability survey.

The procedure and tasks undertaken are as follows:

Step 1: Evaluation Schedule Introduction

Step 2: ***EML & BPMN Introduction ---*** Participants were briefly introduced to EML and BPMN, and some working examples explained (including a Travel Booking system)

Step 3: ***nDeva++ Introduction ---*** The target modelling example was introduced (nDeva++, a web-based University Enrolment System)

Step 4: Participants download EML Tool, User Guide and EML introduction slides from the website

Step 5: ***Modelling Task 1 ---*** Participants were divided into two groups. Group 1 was asked to model nDeva++ overall structure using EML. Group 2 was asked to model nDeva++ system use BPMN

Step 6: ***Modelling Task 2 ---*** Group 1 Participants were asked to add an "Enrol in a Paper" task process using EML. Group 2 Participants were asked to add an "Enrol in a Paper" task process using BPMN

Step 7: Modelling Task 3 --- All participants were asked to save their work and use the other language to repeat Step 5 ~ 6 (i.e. Group 1 participants then used BPMN, and Group 2 EML).

Step 8: ***Modelling Task 4*** --- All participants explored use of the show / hide tasks and trigger functions, show / hide tree component functions.

Step 9: ***Modelling Task 5*** --- Participants explored use of the fisheye view and zooming functions, code generation and validation functions and other support functions

Step 10: ***Answer Questionnaire*** --- Participants were required to complete General Usability Questionnaire

6.2 Participants

Thirty-two subjects participated in this evaluation. They were recruited by email invitations sent to industry, academic and graduate student mailing lists. No inducements were offered. Subjects were required to have BPM, computer science or software engineering literacy, resembling the intended range of various end user targets for the toolset. Participants were selected on a "first in" basis from responses to the email invitation. Table 1 shows the range of experience in IT and BPM as self identified by participants. Formal IT indicates tertiary education in Computer Science and/or Software Engineering. Informal or No IT, indicates those informally self trained or untrained in IT. BPM indicates who have had BPM experience – all had been self-taught. Not specified indicates participants failed to self identify in one or other of the categories. From this, we can observe that the participant set is biaks

sed towards those with BPM knowledge and also has very limited number of participants without a formal IT training. However, this was somewhat representative of the types of end user we were interested in serving with MaramaEML: business process modelling domain experts and service composition experts that have some IT knowledge, but not necessarily detailed service implementation expertise. We would have liked to have a larger number of non-formally trained IT end users in our sample e.g. with a business management background.

Table 1: Participant background experience in IT and BPM, numbers of participants

	Formal IT	Informal or No IT	Not Specified	Total
BPM	13	5	2	20
No BPM	10	0	0	10
Not Specified	2	0	0	2
Total	25	5	2	32

6.3 Results and Discussion

Each subject took around two hours to complete the briefing, evaluation tasks and survey.

1. Usability of EML

Before addressing the two hypotheses specifically, we first present survey feedback looking at usability characteristics of MaramaEML in isolation to obtain a feel for its areas of strength and weakness. Figure 14 shows averaged results from the 5 point likert scale responses. The "EML Tree Structure Usefulness" received the highest average score (4.8 out of 5). Almost all participants believe that using a tree overlay structure as a modeling approach will strongly enhance their modeling ability. The "overlay usefulness" and "EML General Modeling Ability" received averages of 4.5 and 4 out of 5 respectively. The two least well received features related to MaramaEML's perormance with respect to Scalability and Efficiency. Particpants commented on the unresponsive speed of the tool when zooming a large, complex diagram, and the need for better control of information hiding. Looking at differences between the different populations, those with no BPM background on average rated each of the usability characteristics between 0.3 to 0.5 lower than those with BPM experience. These differences were significantly different at a 0.1 level using a t-test, except for efficiency. This would indicate that those with BPM experience saw more value in the abstractions employed in EML than did those without such experience. The small number of participants without formal IT training meant no significant differences could be drawn between those with and without IT training (in this and other characteristics reported below).

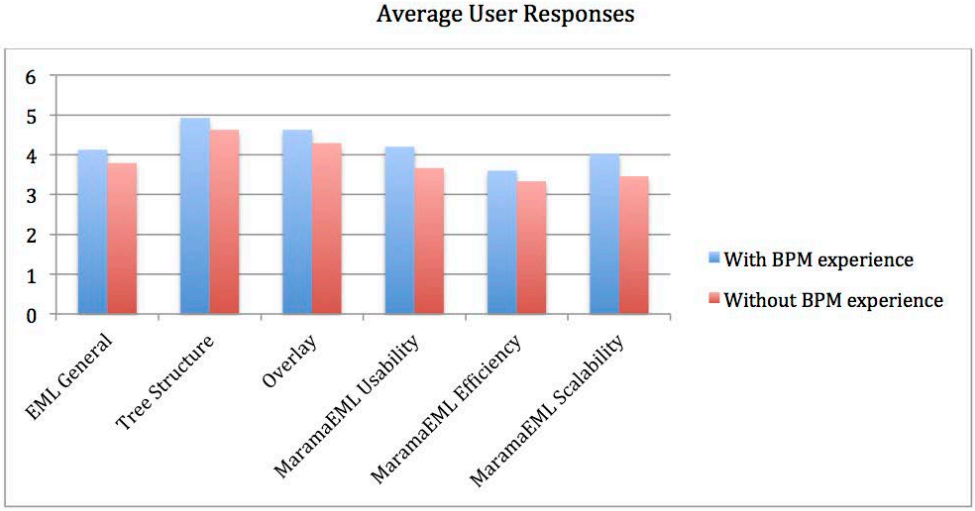

Figure 14: EML and MaramaEML Usability Rating Summary

2. Comparative User Performance

Figure 15 compares the variation in model quality achieved by users when undertaking tasks using EML and BPMN. Only 1 subject could not complete the assigned tasks. To estimate model quality, we collected all answers from the participants and rated them against model answers we had developed. Ratings ranging from

Excellent to Poor were given based on a predefined assessment rubric for each task. These ratings are, by their nature, subjective, but the consistent application of the scoring rubric minimised this subjectivity. Overall, the quality of user EML and BPMN models was very similar. 83% of EML models were rated as good or better, while 75% of BPMN models were rated good or better. However, there was no statistical significance in the differences obtained, when measured using a t-test, at the 0.05 significance level. There was, however, a very significant difference between the performance of those with BPM experience and those without. For both BPMN and EML modelling, those without BPM experience performed very significantly (at the 0.01 level) worse than those with BPM experience, with between 1 to 1.5 scale points difference in the averages. This is unsurprising; one would expect those without BPM experience to produce lower quality models. For those with no BPM experience, their performance with EML was marginally better than with BPMN (at a 0.1 significance level).

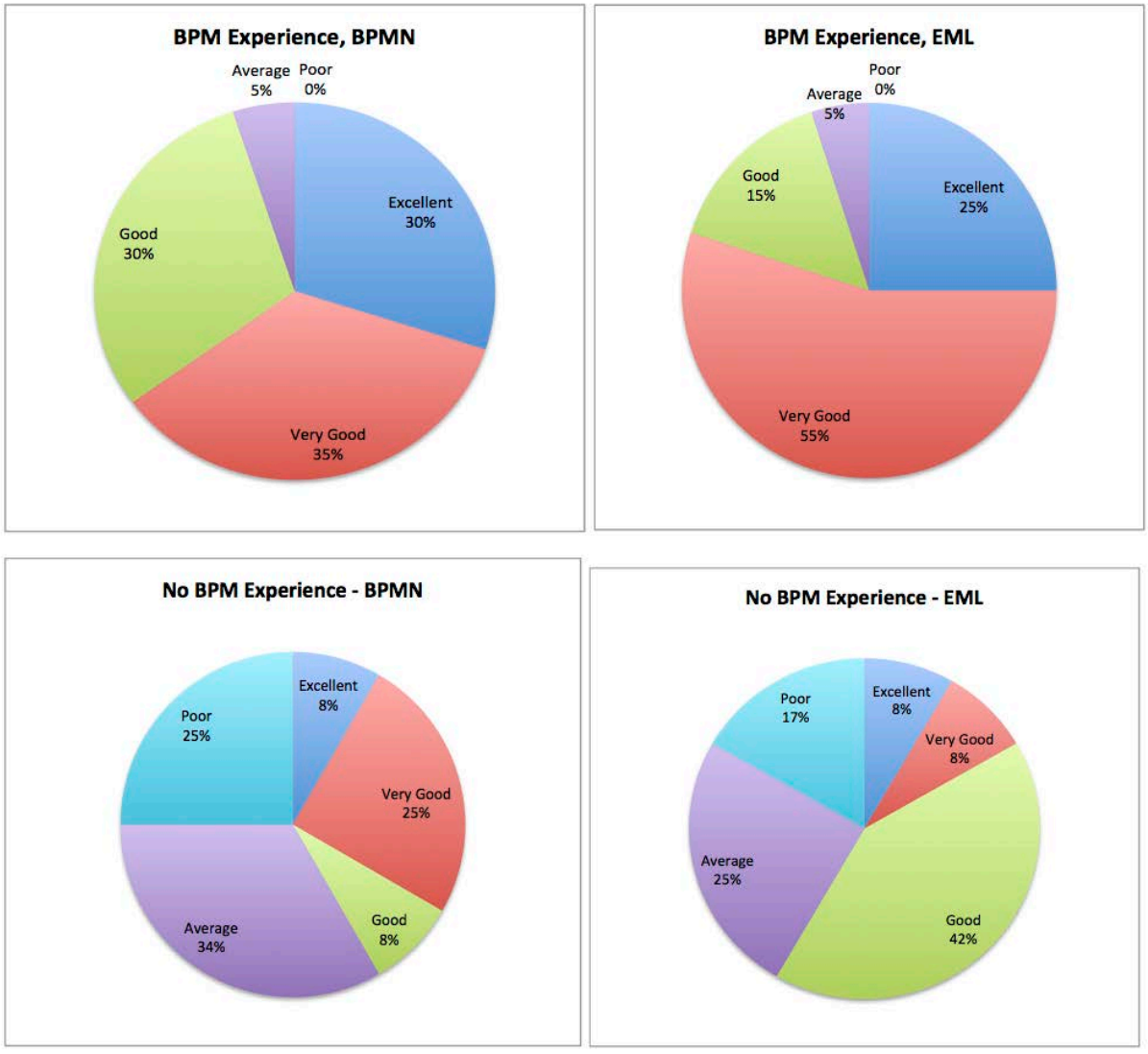

Figure 15: User Performance EML Vs. BPMN

Combined these results support the null hypothesis for H1: i.e. there is no difference in user performance when modelling in EML or BPMN. It is worth noting, however that this result comes from a population where nearly 65% of the total participants already had BPMN experience before performing this evaluation, while EML was a totally new visual language for all of them and they had just 30 minutes training in it at the beginning of the evaluation.

3. Comparative Usability

Figure 16 shows results from our user study of 32 participants rating EML vs BPMN using several usability features introduced in Section 2: Graphical Complexity, Notation Executability, Complexity Management,

Conceptual Integration, Use of Colour and Suitability for Hand-drawing. Each feature was evaluated on a 0 to 5 Likert-type scale, but allowing for users to enter half point values, i.e. effectively a near continuous 11-point scale. For all except graphical complexity, a larger value implies enhanced usability. In all cases the differences in the mean values, as measured using a t-test, are significant at the 0.01 level, making them highly significant. We discuss these results in the subsections below.

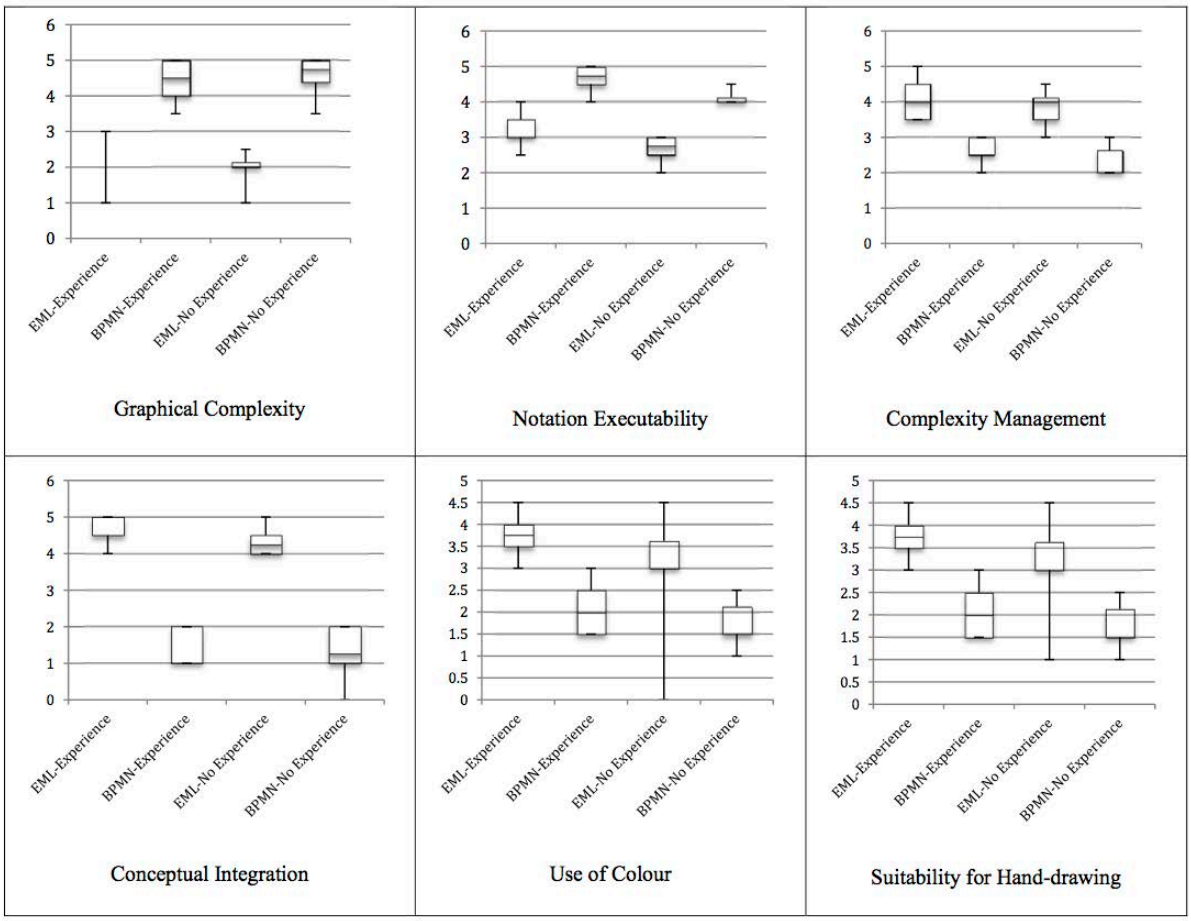

Figure 16: User Perception of EML Vs. BPMN using several dimensions.

Graphical Complexity

For graphical complexity, the responses range from 0, the notation is not graphically complex, to 5 the notation is extremely graphically complex, the one feature where smaller is better. A majority of both BPM experienced users and non-experienced BPM found the graphic complexity of BPMN (both groups mean 4.5, std. devn. 0.5) to be extremely high and thus to be a barrier to its learning and use. By contrast, with EML (experienced mean 1.9, std. devn. 0.4; no experience mean 2.0, std dvn. 0.5), they found it much easier to learn to use each symbol, use it correctly and combine it with other EML symbols. There was no significant difference in the responses of those with BPM experience and those without. The BPMN notation library has a total of 177 visual symbols. Such a level of graphic complexity is clearly problematic for users. EML only has 15 symbols, making the notation easier to learn, apply and understand. In BPMN, the number of grammatical rules increases with the number of symbols. The specification contains hundreds of rules for how symbols can be combined together. However, EML only has 10 rules in total.

Notation Executability

BPMN (BPM experienced mean 4.7, std. devn. 0.3; no experience 4.1, std devn. 0.2) rated significantly better for notation executability in comparison to EML (experienced mean 3.2, std. devn. 0.5; no experience mean 2.6, std devn. 0.4). BPMN's execution semantics means that process engines can execute BPMN visual models, a strong model-driven development capability that our subjects found appealing. Although EML can be used to generate BPEL code in MaramaEML, and MaramaEML has an integrated LTSA engine to validate the

generated BPEL quality, we have not achieved the same level of formal mapping and standardization as is the case with BPMN. However, this was never our intention. We are not trying to produce a rival notation for BPMN or repeat the things that BPMN is doing well, but to use EML as a complementary notation to improve the usability and enhance the modelling ability. Those without BPM experience rated both BPMN and EML's notation executability significantly (at the 0.05 significance level) lower (by 0.4-0.5 scale points on average) than did those with BPM experience. This perhaps indicates a lack of knowledge as to what types of execution of a business modelling language might be useful.

Diagram Complexity Management

EML (experienced mean 4.1, std. devn. 0.5; no experience mean 3.8, std devn. 0.5) scores significantly better than BPMN (experienced mean 2.6, std. devn. 0.4; no experience 2.2, std devn. 0.4) for complexity management. There was no significance in this rating between those with and without BPM experience. When using BPMN most subjects used "Page Connectors" and "Sub-elements" (e.g. Subprocesses, Subchoreographies and Subconversations etc.) to manage diagram complexity. Page connectors allow diagrams to be linked across multiple pages, and "Sub-elements" allow users to break down the diagram into smaller elements. However, even expert users claimed they were not aware of other approaches in BPMN to reduce diagrammatic complexity, in spite of the availability of BPMN sub process diagrams. One user claimed the Page Connector was not an effective solution as it requires users to move between pages, potentially involving a loss of context in the transition. As for "sub-elements", while users generally believed that there is a good advantage in using our diagram nesting approach, most of them do not think a single solution (one user calls it an "all in one" solution) will be enough to resolve the complexity issue in BPMN.

EML uses a hierarchical decomposition approach to manage diagrammatic complexity. EML provides "sub-elements" via "Collapse" or "Expand" tree brunches, and "page connectors" via the connection between different process overlays or tree branches. EML's unique overlay structure (process, trigger and exception) gives users the power to either expand all the details in the same diagram (if required), selectively expand, or expand them to separate diagrams or processes at the next level of detail. EML users have a set of hierarchically linked diagrams rather than a single diagram with smaller diagrams nested inside. It is unclear to us why subjects chose not to use BPMN sub-process diagrams in a simular manner. Most subjects reported that although EML potentially increases the number of diagrams compared to BPMN, all of them are cognitively manageable in size. More than 90% of the participants expressed that they preferred this approach instead of a "super large" diagram as very often happens with BPMN and similar graph-based notations. Other empirical studies (Moody 2002) show that decomposing big diagrams into linked hierarchical diagrams can improve end user comprehension and verification performance by more than 50%.

Conceptual Integration

Our evaluation results show that EML (experienced mean 4.6 std. devn. 0.3; no experience mean 4.3, std devn. 0.4) provides a significantly better conceptual integration capability than BPMN (experienced mean 1.6 std. devn. 0.5; no experience mean 1.3, std devn. 0.8). Subjects found that BPMN diagrams are limited in scope to a single end-to-end business process, resulting in a large number of diagrams at the same level of abstraction with no easy way of gaining an overview of the domain as a whole. When we asked subjects to provide an overview model for the example (in step 5 of the evaluation), they either left it empty or developed their own informal diagrams together with BPMN diagrams to deliver the overview. For EML, the participants reported that the integration of tree and overlay structure provides a very good way to represent the big picture, from the concept level overview to the detailed activities.

Colour-based Visual Expression

EML had a significantly higher result (experienced mean 3.75 std. devn. 0.4; no experience mean 3.1, std devn. 1.1) than BPMN (experienced mean 2.2 std. devn. 0.6; no experience mean 1.8, std devn. 0.5) for this feature. Those with BPM experience rated EML significantly (at the 0.05 level) higher for this feature than those without BPM experience, but these was no significant difference in the rating of BPMN. BPMN does not prohibit the use of colour, but just not actively encourage the user to use it. So because in BPMN every user can freely choose preferred colours, these vary from user to user, potentially leading to misinterpretations. Most of our subjects considered that the colour usage in their BPMN diagram was akin to artwork instead of being used to achieve cognitive effectiveness. More than 80% of our users expressed that EML was the first process modelling language that formally asked them to use the power of colour. Nearly 95% of subjects supported the idea of defining and using colour usage rules formally, as it helps them to enhance communication via the

modelling language. However, we also found one colour-blind user who had difficulty with the colour usage in EML.

Suitability for Hand-drawing

Our evaluation results showed EML (experienced mean 3.8 std. devn. 0.4; no experience mean 3.2, std devn. 0.9) to be significantly better than BPMN (experienced mean 2.2 std. devn. 0.6; no experience mean 1.8, std devn. 0.5) for this feature. Subjects found that the BPMN symbols set presents considerable challenges for hand drawing, as it includes a wide range of icons, composite symbols, shape fills and border styles. Subjects also found BPMN's in-situ decompositions difficult to apply, as any expanding of compound elements essentially requires redrawing the diagram. In contrast, most of the participants found that EML diagrams can be drawn quickly and easily. The multi-overlay structure allows users to add extra overlay on top of existing tree or processes instead of redrawing. However, expanding a tree branch requires some degree of redrawing overhead. For most EML diagrams, subjects only needed to remember 4 major symbols (service, task, tree structure and overlay), and use four different colour pens.

Overall, our comparative usability results provide significant support for hypothesis H2, ie. *Users find EML more usable than BPMN*, in all areas except notation executability. The results suggest that MaramaEML is straightforward to use and understand in comparison to BPMN.

6.4. Limitations

Several threats to validity of our study can be identified. Selection bias could result from the subject recruitment approach. We attempted to mitigate this by using a broad variety of mailing lists for our invitation and taking a first come first served approach to respondents. While the latter eliminates some forms of bias it could provide a bias towards those who have strong interests in BPM; this may explain the larger numbers of those with BPM experience compared to those without (20 versus 10 participants). In addition, it has created a clear bias towards those with formal IT experience over those with no formal IT knowledge (25 versus 5 participants). We have no way of knowing whether the proportions observed in our sample are reflective of the population at large using BPM routinely. Also, the number of participants with no formal IT training was too small for us to be able to differentiate in any significant way the responses of those with and without formal IT training. Repeating the experiment with a larger proportion of non IT trained participants would allow us to draw better distinctions between these two populations.

We chose to compare EML against BPMN only as an example of an existing box and connector BPM approach, suggesting an internal threat to validity over how representative BPMN is in this regard. The widespread use of BPMN suggests this threat is minimal. We also were unable to measure task durations. This was because we were trying to eliminate the external threat of experimenter effects by running the task sessions unsupervised. However, this means our measurement of performance in evaluating H1 only relates to quality of models and not task durations. For the latter we had to rely on qualitative feedback from subjects.

6.5. Research Questions

Returning to our two research questions, the evaluation suggests support for an affirmative answer to RQ1 (*can a tree-and-overlay based service modeling approach provide advantages over existing box-and-line based visual business process modeling languages?)*. However, the strength of this conclusion is limited by the bias of the sample population towards those with BPM and IT experience. The usability feature comparisons demonstrate a clear usability advantage for EML over the use of BPMN, even for this sample group. For RQ2 (*can an effective and efficient environment be implemented for modeling, checking and generating service orchestrations from these tree/overlay-based models?)*, the answer is a little less clear cut. MaramaEML does not have as strong a support for code generation as do existing BPMN tools and this was reflected in the Scalability and Efficiency scoring in the isolated usability results and also the Notation Executability results in the comparative study. Combined these indicate only a partially affirmative answer to RQ2, indicating further work needs to be undertaken to enhance efficiency and scalability of the toolset. Given the bias of the sample population to BPM experience, including experience with other BPM-supporting tools, this may have resulted in more negative answers overall than if a larger sample of non-BPM experienced users were sampled. However, we were unable to demonstrate an statistically significant difference between the groups from our evaluation results.

6.6. Future Enhancements

Based on our evaluation, several improvements and extensions could be made to the EML and MaramaEML to increase their flexibility and functionality. During our end user evaluation, we received strong demand for integrating UML views e.g. sequence diagrams, activity diagrams and state diagrams, into the tool, to complement the other notational views. This would require notation exchange between EML, BPMN and UML meta-models. The current version of MaramaEML includes basic consistency checking between EML views and BPMN views. We use an event log to keep the usage records and trace the model changes. If the user changes the name of a model element in an EML view, an event handler checks the corresponding mapped element in the BPMN view and updates the name automatically. However, this approach cannot cope with situations requiring complex logical analysis. For example, if the user tries to delete a node that has several sub-nodes in EML, the system at this stage will automatically delete all the sub-nodes in an EML view and all the corresponding nodes in the BPMN view. This solution is insufficient. To address this, we need to incorporate a more comprehensive consistency mechanism (e.g. link the sub-nodes with other nodes if there is an underlying relationship between them). We also intend to make use of the OCL-based technique provided in the Marama meta-tools (Li et al 2008) to define dependency and consistency constraints in the EML meta-model.

We are working on approaches to integrate MaramaEML's high level business process modelling with other tools we have developed in the software architecture and application level domains. MaramaMTE (Middleware Testing Environment) is a tool for modelling complex software architectures and generating performance test beds from these models to assess likely software architecture performance (Cai and Grundy et al 2007). We are exploring the possibility to integrate software architecture specification and performance modelling with MaramaMTE via the Marama platform. This would allow performance engineering of complex business process orchestrated web services. ViTABaL-WS (Li et al 2004) is a tool providing event-based web services composition, supporting modelling of both event-dependency and dataflow in designing complex web service compositions. Integration of EML-based business process modelling and ViTABaL-WS-based web service composition and co-ordination modelling would provide more powerful, comprehensive support for both process modelling and web service composition.

During our end user evaluation, we observed that an achromatopsia (colour blind) participant became totally lost in the overlay integration view. An overlay integration view normally mixes process flows (blue color), trigger flows (red color) and exception flows (green color) in the same view. With this version of EML, color is a very important design component to represent and distinguish the information. However, from the evaluation, we found this important piece of visual information disappeared when the notation was viewed by a color blind user. Future research is required to adapt the notation for this group of users probably by using some form of dual coding mechanism.

During the end user evaluation, we found that some users, with low performance computers had response issues with the tool when they were using the zooming or fisheye view function for large EML trees. Improved zooming and fisheye view algorithms are required to improve performance.

7. Summary

Visual languages play a critical role in the business process modelling field. They are used in all areas and all levels of business process modelling practice, from strategic planning to active design. BPMN has emerged as an international standard for business process modelling. Like most other business process modelling notations, BPMN uses workflow based visual approaches. Research shows that "cobweb" and "labyrinth" (Recker and Rosemann et al 2007) or "map shock" (Blankenship and D.F. 2000) problems appear heavily in the complex work flow based diagrams, and lead to cognitive overload. Meanwhile, our earlier work (Anderson and Apperley 1990; Phillips 1995; Li and Phillips et al 2004) on complex user interfaces and their behaviour modelling demonstrated that a tree-based overlay structure can effectively mitigate complexity problems. Other researchers also recommend hierarchical structure as the most cognitively effective way of managing the complexity of diagrams (Sliver 2000; De Marco 1978).

To the best of our knowledge, EML is the first tree overlay structure-based hierarchical decomposition visual language in the area of business process modelling. EML is a novel business process modelling language based on using a tree hierarchy to represent organisationally structured-services and overlay metaphors to represent process flows, conditions, iteration and exceptions. Complex business systems are represented as service trees. Business processes are modelled as process overlay sequences on these service trees. By combining these two mechanisms EML gives the user a clear overview of an entire 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 (e.g. BPMN) to provide high-level business service modelling. Functions for distortion-based fisheye and zooming

views enhance complex diagram navigation ability. MaramaEML can also generate BPEL code automatically from the graphical representations and map it to a LTSA-based model checker for verification.

Although we have used BPMN as a benchmark during the development of EML, we are not proposing EML as a replacement or rival notation for BPMN. In fact, we believe BPMN is an important and successful visual notation in the field of business process modelling. Our end user evaluation also shows BPMN and EML both have advantages and weaknesses, and most importantly, these advantages and weaknesses between BPMN and EML, or at wider level, between work flow based notations and tree overlay based hierarchical decomposition notations, are not overlapping, so there is a space for them to be complementary. Based on this constructive spirit, we developed EML to promote this complementarity and raise the possibility of hierarchical based business process notation.

Acknowledgements

The authors gratefully acknowledge the Foundation for Research, Science and Technology's support for this research under the Research For Industry fund, the University of Auckland for support of this research and Richard Li's PhD, and Jun Huh for his considerable work on Marama meta-tools making MaramaEML possible.

References

1. Adams, M., ter Hofstede, A.H.M., La Rosa, M.: Open Source Software for Workflow Management: The Case of YAWL. IEEE Software 28(3), Mar 2011, pp. 16-19.
2. Aguilar-Savén, R.S. Business process modeling – review and framework, International Journal of Production Economics, vol. 90, no. 2, 28 July 2004, Pages 129-149
3. Ali, N., Hosking, J.G., Huh, J. and Grundy, J.C. Template-based Critic Authoring for Domain-Specific Visual Language Tools, In Proceedings of the 2009 IEEE Symposium on Visual Languages and Human-Centric Computing, Cornwallis, Oregon, USA, Sept 20-24 2009, IEEE CS Press
4. Anderson P.S. and Apperley M.D. (1990): An interface prototyping system based on Lean Cuisine, Interacting with Computers, vol 2, No 2, 217~226.
5. Baeyens, T. (2007, May 02) The state of workflow, http://www.jbpm.org/state.of.workflow.html
6. Baker, J. (2002, June 07) Business Process Modelling Language: Automating Business Relationships, Business Integration Journal, www.bijonline.com
7. Blankenship, J. and D.F. Dansereau (2000) The Effect of Animated Node-Link Displays on Information Recall. The Journal of Experimental Education, 2000. 68(4): p. 293~308
8. Box, D., et al. (2006, January 24) Web Services Eventing (WS-Eventing), http://www.w3.org/Submission/WS-Eventing/
9. BPMI (2010, March 18) http://www.ebpml.org/bpml.htm
10. Buchmann, A., Bornhövd, C., Cilia, M., Fiege, L., Gärtner, F., Liebig, C., Meixner, M., and Mühl, G. (2004) DREAM: Distributed Reliable Event-based Application Management, In Web Dynamics, Springer
11. Cai, R., Grundy, J.C. and Hosking, J.G. (2007) Synthesizing Client Load Models for Performance Engineering via Web Crawling, IEEE/ACM International Conference on Automated Software Engineering, Atlanta, USA, Nov 5-9 2007, IEEE CS Press
12. Chen, P. (2002) Entity-relationship modeling, Contributions to SE, p296 ~ p310, Springer-Verlag, NY.
13. Citrin, W., Strategic Directions in Visual Languages Research. ACM Computing Surveys, 1996. 24(4).
14. De Marco, T. (1978): Structured Analysis and System Specification. 1978, Yourdon Press
15. Draheim D and G. Weber (2005) Form-Oriented Analysis, Springer-Verlag, Berlin Heidelberg.
16. Engels, G. and Erwig M. (2005) ClassSheets: Automatic Generation of Spreadsheet Applications from Object-Oriented Specifications, In Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering, Long Beach, CA, USA, pp. 124-133
17. Eriksson, H.E. and Penker, M., (2000). Business modeling with UML: business patterns at work, Wiley
18. Feng Yi-Heng and Lee C. Joseph (2010) Exploring Development of Service-Oriented Architecture for Next Generation Emergency Management System. AINA Workshops, 557-561
19. Foster, H., Uchitel, S., Magee, J. and Kramer, J. (2003) Model-based verification of web service compositions, In Proceedings of the 18th IEEE international conference on automated software engineering, Montreal, Canada, Oct 2003, IEEE CS Press.
20. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995) Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley
21. Genon, N., Heymans, P., & Amyot, D. Analysing the cognitive effectiveness of the BPMN 2.0 visual notation. In *Software Language Engineering*, Springer 2011, pp. 377-396.

22. Gillain, J., Faulkner, S., Jureta, I. J., & Snoeck, M. Using goals and customizable services to improve adaptability of process-based service compositions. In *IEEE Seventh International Conference on Research Challenges in Information Science (RCIS 2013),* IEEE, May 2013, pp. 1-9.
23. Goel A (2006) Enterprise Integration --- EAI vs. SOA vs. ESB, Infosys Technologies White Paper
24. Gottfried, H. J. and Burnett M. M. (1997) Programming Complex Objects in Spreadsheets: an Empirical Study Comparing Textual Formula Entry with Direct Manipulation and Gestures, Seventh Workshop on Empirical Studies of Programmers (ESP'97).
25. Green, T. R. G. and Petre, M. (1996) Usability analysis of visual programming environments: a 'cognitive dimensions' framework. J. Visual Languages and Computing, vol. 7, 131-174
26. Grundy, J.C., Mugridge, W.B. and Hosking, J.G. Constructing component-based software engineering environments: issues and experiences, Information and Software Technology, Vol. 42, No. 2, January 2000, pp. 117-128.
27. Grundy, J.C., Hosking, J.G., Li, L. and Liu, N. (2006) Performance engineering of service compositions, ICSE 2006 Workshop on Service-oriented Software Engineering, Shanghai, China.
28. Grundy, J.C., Hosking, J.G., Li, N. and Huh J (2008): Marama: an Eclipse meta-toolset for generating multi-view environments, Formal demonstration at the 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 2008, ACM Press.
29. Grundy, J.C., Hosking, J.G., Li, N., Li, L., Ali, N.M., Huh, J. (2013): Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications, to appear in IEEE Transactions on Software Engineering.
30. Hanna, K. (2002) Interactive visual functional programmings, Proceedings of the seventh ACM SIGPLAN international conference on Functional programming ICFP '02, vol. 37, no. 9.
31. Hill, R. D., Brinck, T., Rohall, S. L., Patterson, J. F. and Wilner, W. (1994) The Rendezvous Architecture and Language for Constructing Multiuser Applications. ACM Transactions on Computer-Human Interaction (TOCHI). Vol. 1, no. 2, pp.81-125, ACM Press
32. Hudak, P. (1989) Conception, evolution, and application of functional programming languages, ACM Computing Surveys, vol. 21, no,. 3, pp. 359-411
33. Jung, M.C. and Cho, S.B. (2005). A novel method based on behavior network for Web service composition, In Proceedings of the International Conference on Next Generation Web Services Practices, 22-26 Aug. 2005
34. Kim, J. , J.Hahn and H.Hahn (2000) How Do We Understand a System with (So) Many Diagrams? Cognitive Integration Processes in Diagrammatic Reasoning. Information System Research, 2000.11(3): p.284-303
35. Kornkamol J., S. Tetsuya, and T. Takehiro, (2003) "A Visual Approach to Development of Web Services Providers/Requestors", Proc. of VL/HCC'03, Auckland, p251~p253
36. Kraemer, F. A. and Herrmann P. (2007) Transforming Collaborative Service Specifications into Efficiently Executable State Machines, In Proceedings of the Sixth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2007)
37. Leymann 2001
38. Li, G., Muthusamy, V., & Jacobsen, H. A. A distributed service-oriented architecture for business process execution. ACM Transactions on the Web (TWEB), 4(1), 2, 2010.
39. Li, L., Phillips, C.H.E. and Scogings C.J., (2004) Automatic Generation of a Graphical Dialogue Model from Delphi, Proc of APCHI 2004, Rotorua, p221~p230
40. Liu, N., Grundy, J.C. and Hosking, J.G. Integrating a Zoomable User Interfaces Concept into a Visual Language Meta-tool Environment, In Proceedings of the 2004 International Conference on Visual Languages and Human-Centric Computing, Rome, Italy, 25-29 September 2004, IEEE CS Press, pp. 38-40
41. Li L., Hosking J.G., and Grundy J.C (2007): Visual Modelling of Complex Business Processes with Trees, Overlays and Distortion-Based Displays, In Proceedings of the 2007 IEEE Conference on Visual Languages/Human-Centric Computing (VL HCC 07), Coeur d'Alène, Idaho, U.S.A
42. Li, L., Hosking, J.G. and Grundy, J.C. (2008): MaramaEML: An Integrated Multi-View Business Process Modelling Environment with Tree-Overlays, Zoomable Interfaces and Code Generation Demo session, In Proceedings of the 2008 IEEE/ACM International Conference on Automated Software Engineering, L'Aquilla, Italy, 15-19 September 2008, IEEE CS Press.
43. Li, L. An Integrated Visual Approach for Business Process Modelling**,** PhD Thesis, University of Auckland, 2010.
44. Marshall C. (2004) Enterprise Modelling with UML. Designing Successful Software Through Business Analysis, Addison Wesley
45. Martinez A and M. Patino (2005) "ZenFlow: A Visual Web Service Composition Tool for BPEL4WS", Proc of VL/HCC'05, Dallas, P181~P188

46. Moody, D.L. Complexity Effects On End User Understanding of Data Models: An Experimental Comparison of Large Data Model Representation Methods. in Proceedings of the 10th European Conference on Information Systems (ECIS' 2002.) Gdansk, Poland
47. Moody, D.L., What Makes a Good Diagram? Improving the Cognitive Effectiveness of Diagrams in IS Development. in Proceedings of the 15th International Conference on Information Systems Development (ISD 2006). 2006. Budapest, Hungary.
48. Moody, D.L., The "Physics" of Notations: Towards a Scientfic Basis for Constructing Visual Notations in Software Engineering. IEEE Transactions on Software Engineering, 2009. 35(5): p.756-777
49. Nordbotten, J.C. and M.E. Crosby, The Effect of Graphic Style on Data Model Interpretation. Information Systems Journal, 1999. 9(2): p. 139~156
50. Pautasso, C. and Alonso, G. (2005) The JOpera Visual Composition Language JVLC, vol. 16, no. 1-2, pp. 119-152.
51. Phillips C.H.E. (1994a): Review of Graphical Notations for Specifying Direct Manipulation Interfaces. Interacting with Computers, 411~431
52. Phillips C.H.E (1995): Lean Cuisine+: An Executable Graphical Notation for Describing Direct Manipulation Interfaces. Interacting with Computers, 49~71
53. Recker, J., Rosemann, M., Indulska, M. and Green, P. (2009): Business Process Modeling: A Comparative Analysis. Journal of the Association for Information Systems, Vol. 10, No. 4, pp. 333-363
54. Recker, J. (2010): Opportunities and Constraints: The Current Struggle with BPMN. Business Process Management Journal, Vol. 16, No. 1, pp. 181-201
55. Recker, J. and Rosemann, M. (2010): The Measurement of Perceived Ontological Deficiencies of Conceptual Modeling Grammars. Data & Knowledge Engineering, Vol. 69, No. 5, pp. 516-532
56. Schnieders, A. and Puhlmann, F. (2005) Activity Diagram Inheritance. In Proc of the 8th ICBIS, Poland, p3 ~ p15
57. Silver, B., (2009) BPMN Method and Style: A Levels-based methodology for BPM process modeling and improvement using BPMN 2.0. Vol. New York. 2009: Cody-Cassid Press
58. Urbas. L., Nekarsova. L. and Leuchter. S. (2005) State chart visualization of the control flow within an ACT-R/PM user model, In Proc. IWTMD05, p43~p48.
59. Xiong, P., Fan, Y., & Zhou, M. A Petri net approach to analysis and composition of web services. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 40*(2), 2010, 376-387.
60. Winn, W.D. (1993) An Account of How Readers Search for Information in Diagrams. Contemporary Educationl Psychology, 1993. 18: p.162~185