2011 IEEE Symposium on Visual Languages and Human-Centric Computing, Pittsburgh, USA, Sept 18-22 2011 © IEEE 2011

Automatic Diagram Layout Support for the Marama Meta-toolset

Pei Shan Yap and John Hosking
Department of Computer Science
University of Auckland
Auckland, New Zealand
john@cs.auckland.ac.nz

John Grundy
Centre for Computing and Engineering Software Systems
Swinburne University of Technology
Melbourne, Australia
jgrundy@swin.edu.au

***Abstract*— Automatic layout can be a crucial support feature for complex diagramming tools. Adding suitable layout algorithms to diagramming tools is a complex task and meta-tools should incorporate these for reuse. We present MaramaALM, a generalised set of automatic layout mechanisms. This has been incorporated in the Eclipse-based Marama meta-toolset to support automatic layout in Marama diagrams. It provides an easy-to-use mechanism for tool developers to add such layouts to their generated tools. We describe our motivation for MaramaALM, our approach to its implementation and an example case study of using these tool extensions.**

***Keywords*– meta-tools, user interface design, automatic layout, code generation**

I. INTRODUCTION

Automated or automatic layout is defined as "the use of a computer program to automate either all or part of the layout process" [9]. As visual domain models are increasingly complex, automated support for diagram layout is often essential to improve the effectiveness and efficiency of the modelling processes and resultant diagrams [9], [14], [1]. Although there are different approaches towards the utilization of such automation support, their underlying aims are analogous: to improve tool usability and diagram aesthetics.

We have developed the Marama meta-toolset to support specification and generation of complex multi-view diagramming tools using the Eclipse platform [7]. Marama allows automatic diagram layout algorithms to be developed but only using low-level coding or visual event-based modelling meta-tools. The former requires detailed API knowledge and is time-consuming and complex. The later is quite restricted and still requires detailed knowledge of Marama diagram rendering mechanisms. We were interested in determining whether automatic layout support could be better provided.

II. MOTIVATION AND RELATED WORK

Complex diagramming tools often need layout facilities to aid model visualization. Many use a hierarchy represented by trees e.g. a horizontal tree organising top-level business units. Some use force-directed layout for node and link diagrams e.g. a display of related document relevance for a semantic wiki. We wanted modellers of complex diagrams to be able to use automatic layout support in our Marama-generated [7] diagramming tools. However, in order to achieve a satisfactory level of aesthetic value in the visualization, manual arrangement of elements often consumes too much time and effort. Furthermore, when there is a high level of viscosity [6] in the modelling, a change of one element may require a reorganization of the layout.

However, to realise such layout features usually requires considerable development effort by tool developers. Almost any non-trivial layout algorithm requires developers to write complex code. We wanted to provide Marama tool designers with meta-tool support to quickly and easily add automatic layout support to their Marama-specified diagramming tools. Our past experience indicates that the implementation process of the layout features in Marama is very time-consuming, involving low-level Java coding for each feature [8].

There are a large variety of low-level layout algorithms [5]. In the domain of tree drawing, earlier works such as [15] and [16] have contributed to the tidy renderings of "narrow" tree structures, while more recent work focuses on algorithms that minimize edge crossings. In the domain of force-directed layout drawing, one of the earlier is the spring-based force-directed layout algorithm [3], followed by a technique that uses a simluated annealing approach to determine the termination of the algorithm when the layout achieves an optimal level of aesthetics [4].

The traditional approach to provide automatic layout support is by textual specification, followed by the execution of some layout algorithm. Layout-by-Example is a notational approach suggested in [16] to specify automatic layout of diagrams using the concept of fuzzy theory. In this approach, a layout is generated based on the layout rules that are applied explicitly or extracted automatically from the stereotypes of diagram layout. These stereotypes are specified by a fuzzy visual language. Layout by interactive example is a complementary approach using interaction to specify layout constraints [2], [11], [16]. Constraint hypergraph grammars [12] are used to specify the synthetic structure and layout requirements of diagram in a consistent way. In this approach, the automatic layout support is able to accommodate user-defined adjustments. Visual specification of diagram layout has been carried out in other meta-tools, using interaction to build up layout constraint rules [10]. Kaitiaki allows tool developers to specify event handling using visual specifications [7]. These include event filtering, tool state querying and action invocation. Tool developers can compose handlers from a high-level view and incorporate them into the diagramming environment.

III. The MaramaALM Approach

Marama is a set of Eclipse-based meta-tools that facilitates rapid specification and construction of domain-specific visual language (DSVL) modelling tools [7]. There are two types of Marama user: specification tool and modelling tool users. Specification tool users are modelling tool designers who design and develop their software tools using facilities provided by Marama. Modelling tool users perform tasks using the Marama-generated software tools. The current Marama approach allows customisation of layout primitives using Marama event handlers i.e. plug-in Java code, the Kaitiaki visual event handler specification tool [8], or the use of textual Object Constraint Language (OCL) behaviour constraints [8]. Using these facilities to build complex diagram layout is challenging and time-consuming, even for experienced Marama specification tool users.

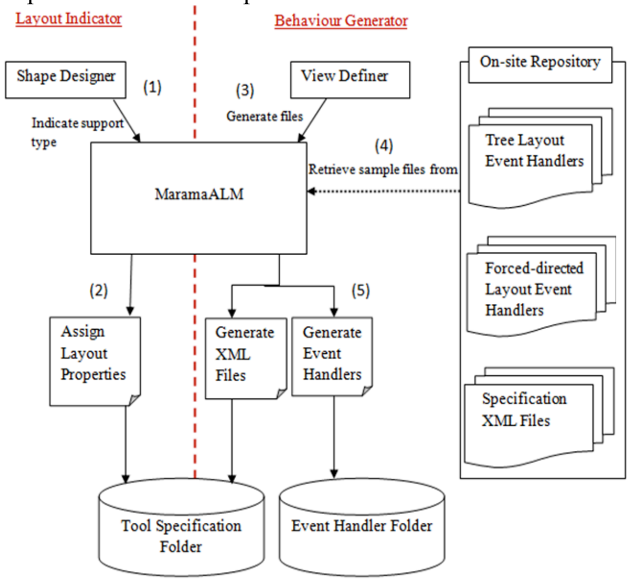

Figure 1. High-level architecture of MaramaALM.

Our primary goal was to provide an automatic layout mechanism for Marama that benefits both modelling and specification tool users. This mechanism will provide overall layout management from the viewpoint of a tool user and a tool designer. In the modelling domain, we wanted to offer better ways for Marama tool users to manage their model layouts. In the specification domain, a more accessible approach is desired for the tool designers to specify and generate layout support for their modelling tools. We have developed *Marama Automatic Layout Manager* (MaramaALM) to achieve this goal.MaramaALM comprises two main components: *Layout Indicator* and *Behaviour Generator*. The layout specification process begins with the Layout Indicator, followed by the Behaviour Generator. Figure 1 illustrates the addition of MaramaALM support into the Marama meta-toolset. The *layout indicator* augments the Marama Shape Designer allowing a specification tool user to annotate shape types that will need to have automatic layout algorithms applied to them. Currently we support the annotation of shapes to allow them to participate in force-directed layout and tree layout. The *behaviour generator* augments the Marama View Designer to allow specification of multiple tree or force-directed layout behaviours on elements of a diagram. It generates infrastructure for the resultant tool to realise the layouts.

The Layout Indicator operates at the shape level in the Shape Designer. It allows meta-modellers to drag and drop a special notation, *LayoutManager* icons, onto shapes. These specify the desired layout support for these annotated shapes. Figure 2 (a) shows an example of such layoutManager annotations (green octagons) on a Marama shape designer specification. The Marama tool developer has indicated these shapes will participate in a tree layout algorithm (nodes and edges). A shape can be either a tree node or a force-directed node, determined by the node type indicated on the LayoutManager annotation. When generating the target tool, a number of necessary properties are assigned to the specified shapes in the underlying Marama tool XML representation. This ensures that their layout behaviours are reflected in the generated modelling environment.

The Behaviour Generator functions in the View Designer. A special notation named *ViewLayoutManager* is provided to customize each desired layout mechanism. It allows the tool designer to indicate the layout support type and connectors between automatically laid out shapes to be used in the diagram type. The tool designer selects a *Generate Tree Visual Handler* or *Generate Force-directed Visual Handler* menu item, depending on the specified ViewLayoutManager, as in Figure 2 (b). MaramaALM retrieves the relevant layout specification files and Java-based event handlers to implement the layout from an existing repository and allocates them to the Marama tool specification folders.

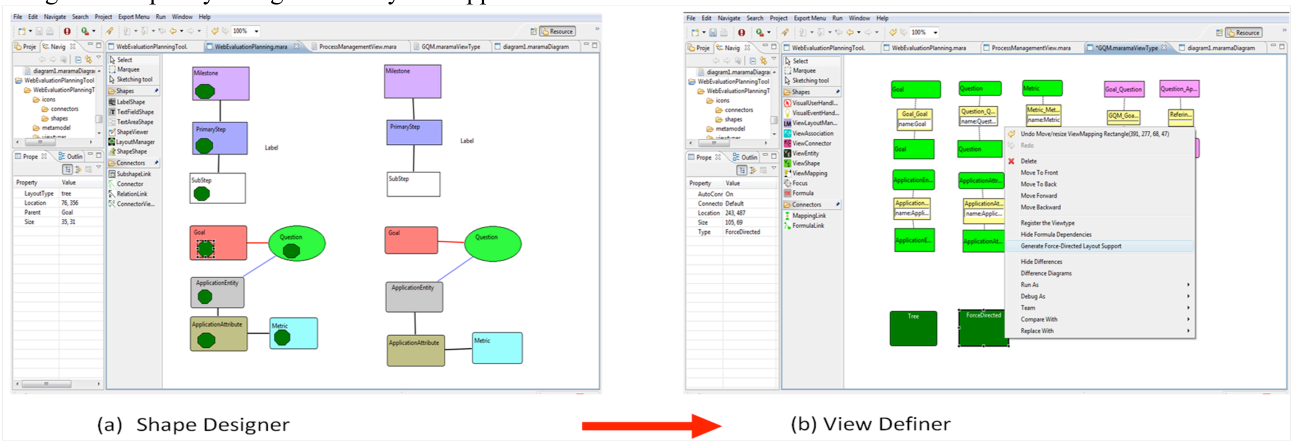

Figure 2. (a) LayoutManager visual annotations added to shapes; and (b) View Designer layout event handler specification.

In our prototype of MaramaALM we have incorporated the choice of tree or force-directed layout behaviours and functionalities to provide an effective yet easy-to-use layout management for modelling tool users. The tree layout mechanism is useful for information visualisations that aim to highlight a hierarchical structure. It comprises layout support including automatic node attachments, tree style switching, collapsible or expandable subtrees, detachable subtrees and dynamic resizing. Layout switching can be activated using a menu triggered event handler. The descendent nodes of a tree are switched from being aligned vertically to horizontal alignment by level. Elision can be activated via a menu triggered event handler.

The force-directed layout mechanism is useful to prevent node overlapping, maintain node proximity and highlight visibility of clusters in the modelling space. Unlike the tree layout, it supports acyclic graphs and any information visualization that does not have a strong hierarchical structure. This mechanism improves the aesthetic value of an initial graph by promoting overall layout symmetry. It provides optional automatic node attachments, layout optimization through node redistribution, an emphasis mechanism and dynamic node resizing. The behaviour emphasis mechanism can be activated using the menu triggered event handler when pointing to a target node. The directly connected nodes of the selected target node are now enlarged and brought closer to the focal point, whereas the rest of other non-directly connected nodes are pushed apart to create an effect of focus and context. We have implemented one force-directed layout and horizontal and vertical tree layout algorithms in MaramaALM to date.

IV. Case Study

The Web Evaluation Planner (WEP) is a Marama-generated tool to assist Web developers in measuring the usability of their Web applications. It comprises a Process Scheduler and a Goal-Question-Metric Builder that utilize both the MaramaALM tree and force-directed layouts to enhance modelling. In a GQM model, the breakdown of a goal into several questions and a refinement of one question into several metrics, form a top-to-bottom tree structure. Goal-Question-Metric Builder offers a tree mechanism to support this presentation. It includes a force-directed layout to assist outlining of the entity-attribute-metric relationships.

The tool building process begins with the meta-model specification of entities and associations for WEP. Then WEP visual notations are created in the Shape Designer to represent the underlying entities and associations. For the Process Management view type, tree layout support is chosen. In order to generate the tree behaviours for the target tool, the modeller drags and drops the *ViewLayoutManager* notation into the view type specification diagram and specifies the desired layout and connector types in a property sheet. For the GQM view type, both the tree and the force-directed layout are required. MaramaALM generates event handlers to enable these in the WEP tool.

Figure 3 (a) shows the sequence of activities that are to be performed in order to measure a health-related Web site. The root Health Web Site Measurement is decomposed into 5 main steps: Determine Project CMM level, Identify GQM Goal, Collect Data, Analyse Data and Document Results. The main step Identify GQM Goal is then further divided into 5 substeps: Identify Entities, Identify Attributes, Identify Metrics, Build GQM tree and Associate Entity-GQM tree Collect Data and Document Results also have subgoals. The plan can be represented in two different styles of tree. MaramaALM related descendent nodes into a vertical form in the first style and horizontal form in the second. Web developers can switch between tree styles using the Switch Tree Style option. Figure 3 (b) shows the tree in horizontal layout automatically arranged by the tool.

The Goal-Question-Metric builder is used to generate GQM trees, specifying the measurement areas of the application components to support the evaluation steps such as Identify GQM Goal mentioned above. Figure 4 (a) shows a GQM tree and its associated set of application entities, attributes and metrics. A goal is refined into several defined questions and the decomposition is portrayed in a tree structure, whereas the application entities, attributes and metrics are created as a mind map of the evaluation modules. To achieve a more symmetric presentation, the force-directed layout optimization process is activated, shown in Figure 4 (b).

V. Evaluation

We used the Cognitive Dimensions framework [6] to analyse the properties of our MaramaALM specification and diagram layout features. MaramaALM significantly reduces the effort of both modellers and tool designers by providing a generalised automatic mechanism to facilitate layout management. It offers a *terse notation* with simple abstractions (*low abstraction gradient*) and *low viscosity* by encapsulating the low-level implementations into a generalised component that can be easily applied to any Marama-generated tools. It also assists in reducing *viscosity* problems in the generated modelling tools by providing automatic layout in those tools. The tool-designers can change the involved shapes and connector in one place and this modification will be reflected throughout the whole mechanism (relatively *low hidden dependencies* when modelling). In the modelling environment, the modellers can easily add, change or delete the shapes and connectors in the provided structure.

However, our approach comes with some trade-offs. These include some *hidden dependency* issues and *premature commitment* problems for specification tool users. During the specification process, the use of Shape Designer and View Designer is inseparable. Each depends on the another to generate necessary properties and manage shape-entity mappings in order for MaramaALM to function properly, hence from one view there is a hidden dependency to the other. Some layout features are highly associated and cannot be isolated. *Premature commitment* is required, as meta-modellers need to decide which shapes and connectors are to be included during specification. MaramaALM achieves a relatively high level of *closeness of mapping, consistency,* and *visibility* while keeping *hard mental operations* and *error proneness* to a minimum.

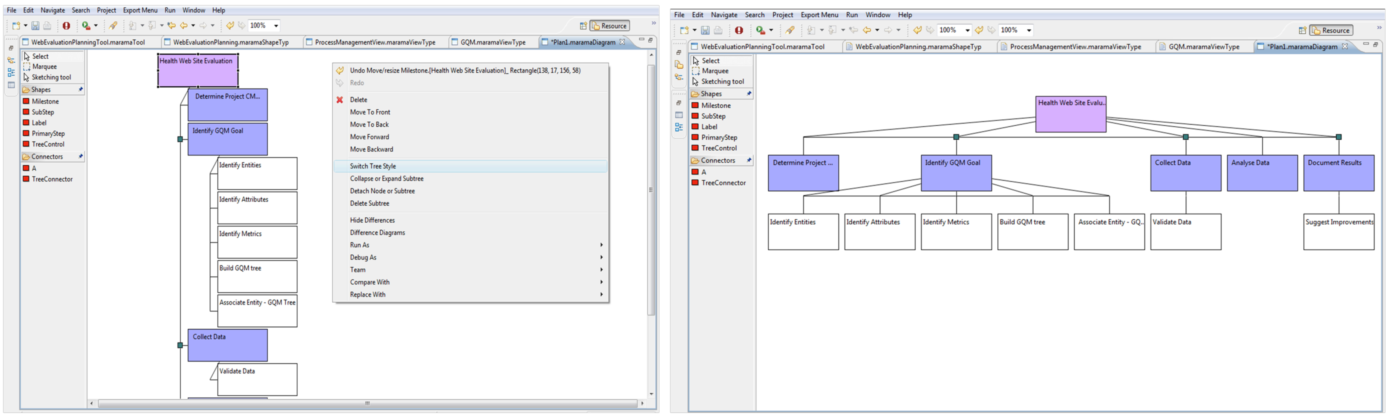

Figure 3. (a) MaramaWEP vertical tree layout and (b) MaramaWEP horizontal tree layout.

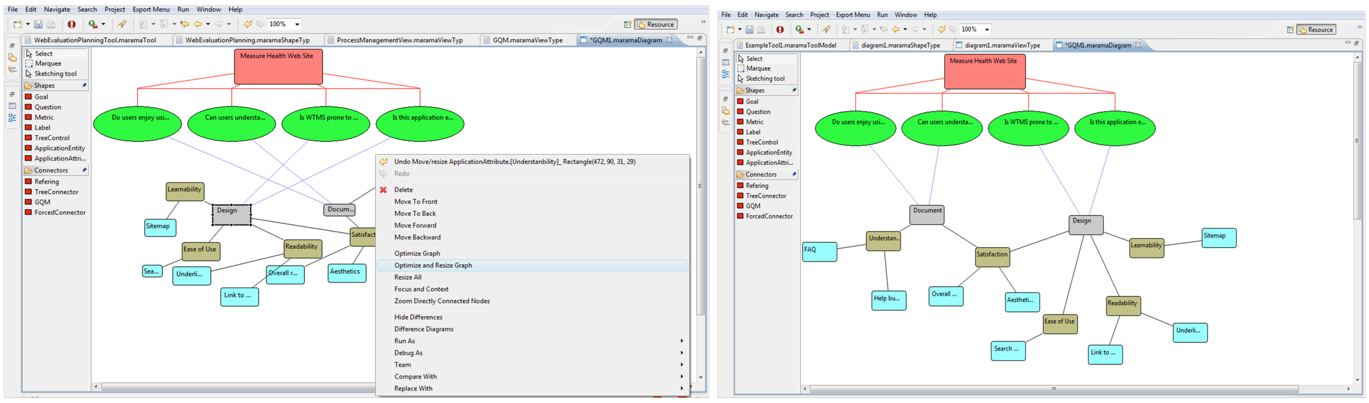

Figure 4. (a) MaramaWEP Goal-Question-Metric view with force-directed layout and (b) with optimized layout.

VI. Summary

We have successfully incorporated an automatic layout generator into the Marama meta-toolset. Using the tree and force-directed layout features, modellers can much more easily produce an aesthetically pleasing layout. MaramaALM significantly reduces the effort of both modellers and meta-modellers by providing a generalised automatic mechanism to facilitate layout management. It offers simple abstractions and low viscosity but comes with trade-offs of hidden dependency and premature commitment. MaramaALM greatly reduces the effort of the tool developers in specifying such features for their tools. Future enhancements include implementing different algorithms for the tree and force-directed layout, and extending the tree layout modes. Further research on how to provide a formal mechanism to support secondary notations during the layout specification process would be useful.

Acknowledgment

Support from the New Zealand Foundation for Research, Science and Technology is gratefully acknowledged.

References

[1] Barone, R., Cheng, P. C.-H., Representations for Problem Solving: On the Benefits of Integrated Structure. 8th Int.Conf. on Information Visualisation, pp. 575-580, 2004.

[2] Dwyer, T., Marriott, K., Wybrow, W. Interactive, Constraint-based Layout of Engineering Diagrams. Volume 13: Layout of (Software) Engineering Diagrams 2008, Electronic Communications of the EASST.

[3] Eades, P. A heuristic for graph drawing, In Congressus Numerantium, vol. 42, pp. 149-160, 1984.

[4] Fruchterman, T. M. J and Reingold, E. M., Graph drawing by force directed placement. Software: Practice and Experience, vol. 2, no. 11, pp. 1129-1162, 1991.

[5] Graham, M. and Kennedy, J. A Survey of Multiple Tree Visualisation. Information Visualization 2010 9:235.

[6] Green, T.R.G. & Petre, M. Usability analysis of visual programming environments: a 'cognitive dimensions' framework. JVLC 1996 (7).

[7] Grundy, J.C., Hosking, J.G., Liu, K., Huh, J., Marama: an Eclipse meta-toolset for generating multi-view environments, ICSE 2008, 819-822.

[8] Liu, N., Grundy, J.C., Hosking, J.G. A Visual Language and Environment for Specifying User Interface Event Handling in Design Tools, 8th Australasian User Interface Conf. pp. 87-94. 2007.

[9] Lok, S. and Feiner, S., A Survey of Automated Layout Techniques for Information Presentations, SmartGraphics, pp. 61-68, 2001.

[10] Maier, S. and Minas, M. A Static Layout Algorithm for DiaMeta. ECEASST 10, 2008.

[11] Maier, S. And Minas, M. Interactive diagram layout. CHI Extended Abstracts 2010: 4111-4116.

[12] Minas, M. & Viehstaedt, G., Specification of Diagram Editors Providing Layout Adjustment with Minimal Change, IEEE Symp. Visual Languages, pp. 324-329, 1993.

[13] Purchase, H. C., Allder, J., and Carrington, User Preference of Graph Layout Aesthetics: A UML Study. Graph Drawing 2000.

[14] Purchase, H.C., Metrics for Graph Drawing Aesthetics, Journal of Visual Languages & Computing, vol. 13, no. 5, pp. 501-516, Oct. 2002.

[15] Reingold, E. M. and Tilford, J. S., Tidier drawings of trees, IEEE Transactions Software Engineering, vol. 7, no. 2, pp. 223–228, 1981.

[16] Sugihara, K. et al., Layout-by-example: A Fuzzy Visual Language for Specifying Stereotypes of Diagram Layout, IEEE WS on Visual Languages , pp. 88-94, 1992.