# End-User Oriented Critic Specification for Domain-Specific Visual Language Tools

Norhayati Mohd Ali [1], John Hosking[1], John Grundy[2], Jun Huh[1]

[1]The University of Auckland
Private Bag 92019
Auckland, New Zealand

[2]Swinburne University of Technology
PO Box 218, Hawthorn Vic 3122
Melbourne, Australia

nmoh044@aucklanduni.ac.nz, j.hosking@auckland.ac.nz, jgrundy@swin.edu.au, designersheep@gmail.com

## ABSTRACT
This paper presents a new approach to specifying critics for domain-specific visual language tools using a visual and template-based approach. In this paper we describe our approach for specifying critics for domain-specific visual language tools. This allows target end-user tool developers to design and implement critics efficiently in a natural manner. We describe a survey that we conducted to evaluate our new approach and the Cognitive Dimensions approach that was applied in the survey questionnaire design. Survey results are briefly discussed along with the issues raised by some respondents to improve our approach.

## Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *evolutionary prototyping.*

## General Terms
Design

## Keywords
critic, critic authoring template, critic specification, critiquing.

## 1. INTRODUCTION
Much research has been devoted to critic tools (or critiquing systems). ArgoUML [11] and Java Critiquer [10] and many others are examples of computer-based critics. These critic tools produce critiques (or critic feedbacks) that are specific to their problem domains. The use and context of critics varies from one domain to another. To date, critics have been applied in a wide variety of domains including education, medicine, design, programming and software engineering [3, 10, 11]. In general, critic tools provide knowledge support to users who lack specific pieces of knowledge about their problem or solution domains. Furthermore, they detect potential problems; give advice and alternative solutions; and possibly provide automated or semi-automated design improvements for users.


*ASE*'10, September 20–24, 2010, Antwerp, Belgium.


Inspired by the existing critic tools work, we made an attempt to apply similar ideas to our Marama meta-modeling tool. Marama is implemented as a set of Eclipse-plugins and includes meta-specification tools as well as modeling tools [5]. Most existing critic tools are not developed within the environment of a meta-modeling tool. Our meta-tools are used to generate complex visual modeling tools, and these modeling tools could benefit from the addition of various critics. Thus, we wanted to extend our Marama meta-tools by embedding a critic design and generation component. The main purpose of our work is to assist end-user tool developers to specify and generate critics efficiently and effectively.

## 2. BACKGROUND AND MOTIVATION
A critic development must involve, at some level, the authoring of critic rules. As [10, 13] claim, critic rules are normally written in advance by system designers to develop a critic system and it is hard for end users to modify existing rules or add new critic rules after a critic system is deployed. However, as [6, 13] argue, critiquing may need to be adjusted depending on various situations. Furthermore, [4] emphasizes that users should not be required to have comprehensive programming knowledge in order to perform the modification of critic rules. For these reasons it is important to allow users to understand critic rules and be able to modify and expand the rule set by authoring new rules to incorporate in critic systems. In general, the capability for rule authoring is to enable end-user designers to construct, tailor and store their own critic rules, thus allows end-user tool developers/customizers to contribute to the system's feedback process [13].

The need to specify critics in a simple way using an easy to use, high-level language is the motivation for our research. We see the opportunity to apply domain specific visual language (DSVL) approaches to reduce end-user barriers. DSVLs are graphical notations specially devised for specific needs and knowledge [12]. These visual languages enable anyone who is a domain expert to use the application development tool for the domain [12].

## 3. VISUAL AND TEMPLATE-BASED APPROACH
We propose a visual and template-based approach to support end-user developers in critic specification tasks. We have developed and applied this in the context of our Marama meta-tool, which provides a range of other facilities to assist end-users to develop modelling tools, but the concept is applicable to other types of diagrammatic tools. In this section, we describe three main parts

to our critic development approach: 1) a visual critic definer, 2) a template-based approach and 3) critic execution-examples.

The main underlying idea in our approach is to use information expressed in a meta-diagram (in our case the Marama meta-model diagram) as input for critics to be realized in a diagram (in our case a Marama diagram in the realized modeling tool specified by the meta-model). It is important to mention that our approach is only minimally dependent on the notation used in the meta-diagram. The Marama meta-model diagram is expressed using an Extended Entity Relationship (EER) notation which specifies entities and relationships, together with their attributes [8]. If a richer notation is used, more information can be extracted from the meta-model diagram and, thus, be used for specifying critics and checking user diagrams.

Due to the argument made by Fischer et al. that critics should be considered as embedded systems rather than as stand-alone components [3], we have created a *critic definer* to support the critic specification tasks. Once a tool is equipped with sufficient information, the end-user tool developer then can specify the required critics for that tool via the *critic definer*. Figure 1 (top) shows an example of a meta-model for a simplified enterprise modeling language tool from a business process domain and the critic definer editor (bottom) for specifying the tool's critics.

The visual critic definer editor has three main elements: *CriticShape*, *CriticFeedbackShape*, and *Operator*, and three connectors: *CriticFeedbackConn*, *CriticDependencyLink*, and *OperatorConn*. The *CriticShape* (rounded square shape) allows a target end-user developer (or tool developer) to specify critic(s), whereas the *CriticFeedbackShape* (oval shape) is used to specify the feedback for each defined critic. The *Operator* (AND, OR, and XOR) supports the creation of composite critics. The relationship between critic and feedback is supported by the *CriticFeedbackConn*. In a case where one critic is dependent on another critic, a *CriticDependencyLink* is used for visual representation. The *OperatorConn* links critics with the logical operator (AND, OR, and XOR) to form a composite critic. This allows complex critics to be readily built from simpler parts.

We provide the end-user developers with a critic authoring template to specify their own critic(s). We were inspired by the business rule (BR) templates approach [7], adapting this concept for use in our critic authoring templates. We chose this approach so as to enable end-user developers with limited programming capability to be able to specify critics for their domain-specific visual language (DSVL) tools. Our reasoning was that as BR had been proved useful in an end user oriented business rule specification domain [7], it would be useful for our own domain which had similar levels of complexity.

Our critic authoring templates are applied to a target DSVL tool's meta-model to review its target model instances. The critic specification is defined by selecting a *CriticShape* in the visual critic editor. The *CriticShape* is associated with a form-based interface, designed to ease the task of specifying critics. The target end-user developers specify their critics by selecting from available templates provided in the interface and completing the form to enter required information. The critic authoring templates support three types of template as shown in Table 1. Attribute constraint templates are used to specify essential properties around uniqueness, optionality (null), and value check of an entity's attributes [7]. The relationship constraint templates assert relationship type, cardinality and role constraints of each entity participating in a particular relationship [7]. Action assertion templates specify an action to be activated on the occurrence of a certain event or on the satisfaction of certain conditions [7].

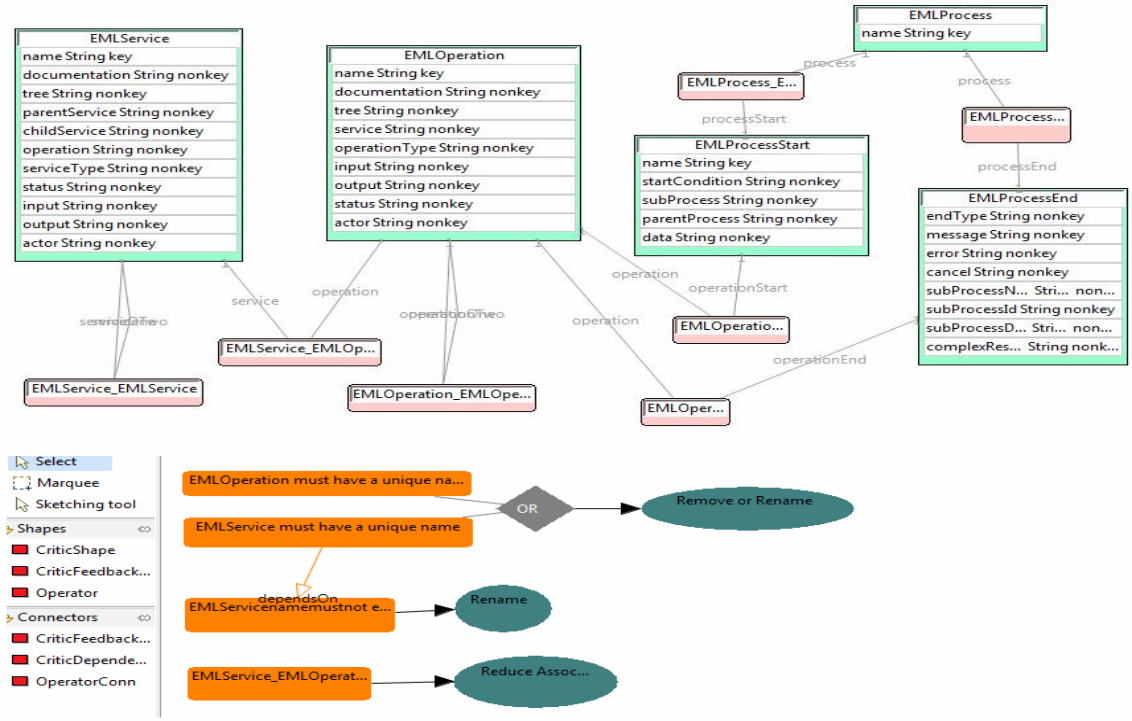


**Figure 1: Meta-model of *SimplifiedMaramaEML* tool defined in the meta-model editor (top) and critic definer editor (bottom).**

Once the critic(s) has been defined in the visual critic definer editor, the next task is to specify feedback for the defined critic(s). This is done via the *CriticFeedbackShape* which is also associated with a form-based interface, i.e. *Critic Feedback View*. The critic feedback view has the following properties: (i) *critique strategies* that determine the execution mode of the critic. This can be either active or passive. An active critic will monitor continuously a user's tasks and warns the user as soon as a critic is violated and then provides feedback (a critique). A passive critic only works when a user asks explicitly to check for a critic violation. (ii) *modalities of critiques* involve the presentation of the critique. This can be textual, graphical or a combination of both; (iii) an *explanation* represents a justification of a critique; (iv) a *suggestion* indicates an action to resolve the critic violation; and (v) a *critique message* specifies a textual message that is displayed for each critic that has been defined.

Figure 1 (bottom) shows critic examples, specified against a simplified version of *MaramaEML* [5], a business process specification tool. The topmost critic is a complex critic, where two critic conditions, in this case two name uniqueness constraints, have been connected by OR to share a common feedback element. The bottom-most critic is an example of an action assertion template in use. Here the tool developer wants the service entity to have no more than four operations. A critic can be specified for this case by defining the relevant properties for event, condition and action in an action assertion template. Here the event is the creation of an association link, the condition is the cardinality is greater than 4 and the action is to delete the new association. When the user runs the tool, a critique will be displayed if the event occurs to warn the user, followed by execution of the action. In a situation where one critic might be dependent on another, this can be represented visually with a *CriticDependencyLink* as shown in Figure 1.

**Table 1: Constraint and Action Assertion Templates [7]**

| Types | Templates |
|---|---|
| Attribute Constraint | *<entity> must have \| may have a [unique] <attributeTerm>* |
| | *<attributeTerm1>must be \| may be <relationalOperator> <value> \| <attributeTerm2>* |
| Relationship Constraint | *[<cardinality>]<entity1> is a/an <role> of [<cardinality>]<entity2>* |
| | *[<cardinality>]<entity1> is associated with [<cardinality>]<entity2>* |
| | *<entity1> must have \| may have [<cardinality>]<entity2>* |
| | *<entity1> is a/an <entity2>* |
| Action Assertion | *When <event> [if <condition>] then <action>* |

Once a critic and feedback are specified and defined, these two elements are linked to indicate that a critic comes with a fix action. Although [4] stated a critic does not necessarily solve a user's problem, in our approach we expect the end-user developers to indicate a fix action, where possible, for each critic defined for their DSVL tool. All critics and feedbacks that have been defined are saved in an XML format in the meta-tool's repository. Critics and feedbacks are instantiated when the tool is started by a user. Figure 2 shows an example of specifying a critic (top) and its execution after triggering (bottom).

We wanted end-user tool developers to be able to specify their own critics. In a case where the available template does not support the desired critic specification, we allow the developer to construct a new critic template via a *Critic Template* editor. The end-user developer initially needs to construct the new critic statement. Based on the critic statement, the developer selects the necessary properties to form a new critic template that represents the new critic statement that has been defined. Our critic authoring templates are not as highly expressive as natural language rule statements, but provide sufficient expressiveness to allow end-user developers to understand, modify and possibly author critic rule expressions without expert tool developers.

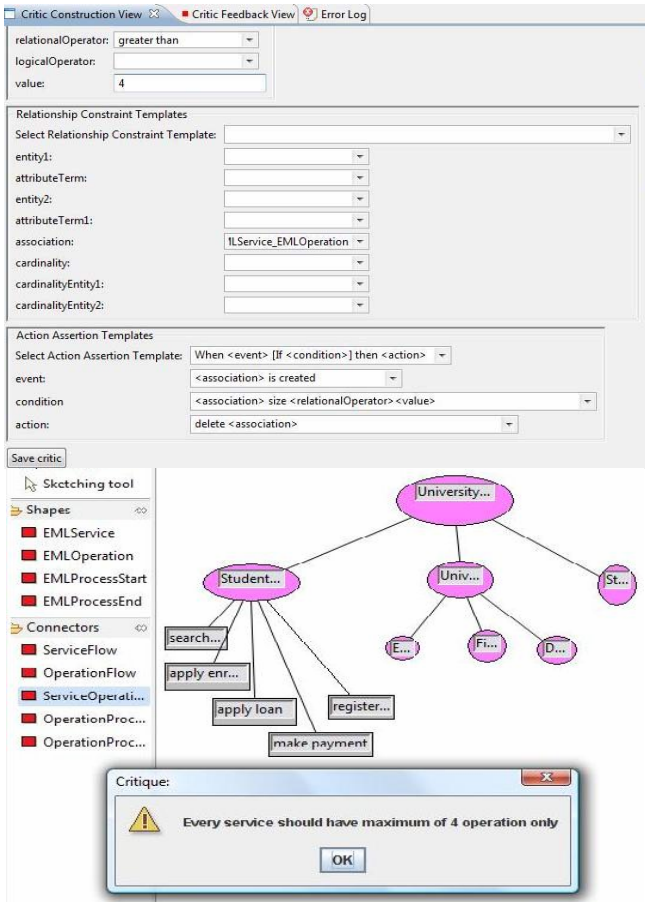


**Figure 2: Action assertion critic specification (top) and execution (bottom) after the trigger event occurs**

The critic rule templates also leverage the rich meta-model diagram facilities, allowing experienced end-user developers to author moderately complex domain specific templates. After specification, new critic templates are listed in the available templates and can be used to specify critics. Thus, the available templates list can be expanded according to the new critic templates created in the critic template editor.

End-user developers can construct single critics based on one preference. They can also construct complex critics which extend the expressive power, while still retaining the relative simplicity of the BR template approach. A complex critic is a critic that has

multiple features that need to be measured. This can be done using the action assertion templates and logical operators AND, OR and XOR. This allows users to specify complex critics by building them from parts and also facilitates reuse of simple critic parts.

# 4. EVALUATION

To evaluate our approach we conducted a user evaluation with ten volunteer researchers and students who had basic background knowledge of the Marama meta-tools and who were interested in modeling and the development of modeling tools to support their work. We adapted the questionnaire designed by [1] based on the Cognitive Dimensions of Notations framework [2]. This provided questions targeted at each of the cognitive dimensions as we were interested in the tradeoffs amongst those dimensions that respondents observed.

The Marama Critic Definer provides good *visibility* and *viscosity* for the target end users. Nine out of total 10 respondents answered that it is easy to see various parts of the tool and make changes. The only respondent who doubted the easiness to see various parts of the tool commented that due to a lack understanding of meta-tool concept and as a novice user it was hard to see the function of various parts of the tool. *Diffuseness* refers to the verbosity of language, i.e. the number of symbols required to express the meaning. Eight respondents answered that the notation is succinct and not long-winded. The Marama Critic Definer suffers from a *hard mental operations* (degree of demand on cognitive resources) problem as four respondents claimed that they have to concentrate and think carefully to use the critic templates for specifying a critic. This may be because users were unfamiliar with the critic authoring templates. Three respondents disagreed and three were undecided as to whether using the tool required hard mental effort. The Marama Critic Definer is likely to be less error prone as five respondents claimed that the notation is very straightforward and supported by a form-based interfaces which is familiar to most users. The respondents that answered it is easy to make mistakes raised the issue that unfamiliarity with the templates can cause users to make mistakes in specifying critics. The Marama Critic Definer offers good c*loseness of mapping.* All of the respondents agreed that the Marama Critic definer view provides a notation that closely related to the domain. *Role Expressiveness* for the Marama Critic Definer is obvious where seven respondents answered it is easy to tell what each part is for when reading the notation. Eight respondents said that the dependencies are visible and two respondents are undecided. *Hidden dependencies* are primarily between the visual critic definer view and the form based template views. Moody argues that this type of hierarchical dependency is of positive benefit in his Principal of Complexity Management [9]. The critic definer supports *progressive evaluation* well. Nine respondents answered it is easy to stop and check work progress. Critics and feedbacks properties can easily be edited and any new changes will take effect during the model execution of the tool. All of the respondents agreed that there are no *premature commitments* in the Marama Critic Definer view. The user can freely specify a critic using any critic templates. However, the user needs to define a critic first before a critic feedback can be specified and linked with the defined critic. The user can add a critic as well as the critic feedback for the Marama tool incrementally as he/she encounter new critics.

Issues that raised by some respondents to improve the tool are: i) consider using more artificial intelligence (AI) techniques; ii) transform the critic templates into visual entities; iii) give a more detailed explanation of templates; iv) consider coloring to differentiate different types of critic; v) add a more explicit visual representation of the relation between critics and the tool's meta-model elements; vi) consider highlighting the design item that triggered a critic; vii) add feedback information into a visual critic element with connection and layout automatically generated.

# 5. REFERENCES


[1] Blackwell, A.F. and Green, T.R.G. 2000. A Cognitive Dimensions questionnaire optimised for users. In Proceedings of the 12th workshop of the Psychology of Programming Interest Group, Cozenza Italy, April 2000.

[2] Blackwell, A.F et al. 2001. Cognitive Dimensions of notations: design tools for cognitive technology. Cognitive Technology 2001 (LNAI 2117). Springer-Verlag, pp. 325-341.

[3] Fischer, G., Lemke, A.C. and Mastaglio,T. 1991. Critics: an emerging apporach to knowledge-based human computer interaction. International Journal of Man-Machine Studies, 1991(35), pp. 695-721.

[4] Fischer, G. et al. 1999. The role of critiquing in cooperative problem solving. ACM Transactions of Information Systems, Vol.9, No.3, April 1999, pp. 123-151.

[5] Grundy, J.C., Hosking J.G., Huh, J. and Li, N. 2008. Marama: an Eclipse meta-toolset for generating multi-view environments. Formal demonstration paper, ICSE 2008, Liepzig, Germany, May 2008, ACM Press.

[6] Irandoust, H. 2006. Critiquing systems for decision support. DRDC Valcartier TR 2003-321. http://pubs.drdc.gc.ca/PDFS/unc44/p524782.pdf.

[7] Loucopoulus, P. and Wan Kadir, W.M.N. BROOD: Business rules-driven object oriented design. Journal of Database Management, 2008, 19(1), pp. 41-73.

[8] Markowitz, V. Extended Entity Relationship Diagram, http://sdm.lbl.gov/OPM/DM_TOOLS/OPM/ER/ER.html.

[9] Moody, D.L. The "Physics" of Notations: Towards a scientific basis for construction visual notations in software engineering. IEEE TSE 2009.

[10] Qiu, L. and Riesbeck, C.K. 2008. An incremental model for developing educational critiquing systems: experiences with the Java Critiquer. Journal of Interactive Learning Research, 2008(19), pp.119-145.

[11] Robbins, J.E. and Redmiles, D.F. Cognitive support, UML adherence, and XMI interchange in Argo/UML. Journal of Information and Software Technology, January 2000, vol. 42, issue 2, 25, pp. 79-89.

[12] Sprinkle, J. and Karsai, G. 2004. A domain-specific visual language for domain model evolution. Journal of Visual Languages and Computing, June-August 2004, vol.15, issues 3-4, pp. 291-307.

[13] Oh,Y., Gross, M.D. and Do, E.Y.-L. 2008. Computer-aided critiquing systems, lessons learned and new research directions. In Proceedings of the 13th International Conference on CAADRIA, Chiang Mai (Thailand) 9-12 April 2008, pp. 161-167.