

# HorusCML: Context-aware Domain-Specific Visual Languages Designer

Mohamed Almarsy\*, John Grundy\*, and Ulf Rüegg<sup>+</sup>

\* Centre for Computing and Engineering Software and Systems, Swinburne University of Technology, Hawthorn, Australia

<sup>+</sup> Department of Computer Science, Kiel University, Kiel, Germany

malmorsy@swin.edu.au, jgrundy@swin.edu.au, uru@informatik.uni-kiel.de

**Abstract**—The objective behind building domain-specific visual languages (DSVLs) is to provide users with the most appropriate concepts and notations that best fit with their domain and experience. However, the existing DSVL designers do not support integrating environment and user context information when modeling, editing or viewing DSVL models at different locations, permissions, devices, etc. In this paper, we introduce HorusCML, a context-aware DSVL designer, which supports DSVL experts in integrating necessary context details within their DSVLs. The resultant DSVLs can reflect different facets, layouts, and behaviours according to context it is used in. We show a case study on developing a context-aware data flow diagram DSVL tool using HorusCML.

**Keywords** — domain-specific visual languages; context awareness; adaptive DSVLs

## I. INTRODUCTION

Domain-Specific Visual languages (DSVLs) are widely used in many domains, including software engineering, hardware design, robots, construction, business and finance. The key benefit of DSVLs is that they help end users to visually model their problems/solutions using domain-specific concepts, expressive notations, and suitable abstraction-level that fit their mental models and work practices [1]. The development of such DSVLs and tool support is not an easy task [5]. It usually requires involvement of software engineers to help in developing such DSVLs after building formal or informal domain ontology [2, 3] by domain experts. In our recent work we have developed a tool, Horus, itself providing a set of DSVLs to specify new DSVL tools [5, 6]

The performance of interpreting a given diagram instance by end users greatly depends on the presentation being used in modelling it [1]. Moody [4], in his physics of notations (PON) work, discusses the semiotic clarity principle, which states on the one-to-one relation between DSVL visual notations and the corresponding domain entities. From the DSVL usability point of view, this might be true when a certain group of users with similar background is going to use such DSVL. However, in software engineering for example, we have business analysts, architects, designers, developers, etc. dealing with the same domain, but at different levels of abstraction and often with different notations. When performing different tasks, the same user may use quite different notations and entities. Users may use different platforms to edit/view their DSVLs. Such user, task, platform and domain *context information* needs to be considered when developing and realizing DSVLs and their supporting tools.

A key limitation that we found with existing DSVL designers, including our previous tools [5, 6], is the lack of context-awareness, such as capturing user permissions, preferences, localization, and target device used in modeling or viewing DSVL diagrams. This usually results in either creating multiple DSVLs and tools for every context, or using different notations that are often confusing for end users. It also requires developing language transformers that can facilitate switching between multiple visualizations, layouts and contents of these related, context-dependent DSVLs.

Supporting context-awareness requires facilitating modeling, monitoring, and enforcing of context information that may impact how much information (model entities, or entity attributes) the user can/wants to see; how information should be represented for different end users according to their preferences and context; and how to layout diagram elements. In this paper, we introduce a novel context-aware DSVL designer, HorusCML, which extends our original Horus DSVL designer to support context-awareness. The basic idea of our approach is to provide the DSVL experts with capabilities to model relevant contextual information that should be monitored and how such information could be used to effect adaptations on DSVL model instances according to current context status. We modified Horus to support the specification of multiple visualization elements and linking these elements to corresponding domain entities. We have conducted a preliminary evaluation of HorusCML by developing a context-aware data flow diagram DSVL tool. The evaluation shows that Horus can easily capture, model, monitor, and enforce DSVL context information.

## II. MOTIVATING SCENARIO

Consider SwinSoft, an imaginary multi-national software house focusing on developing business applications. SwinSoft adopts the global software development (GSD) model, where development teams are geographically distributed in different countries. SwinSoft depends heavily on data flow diagram (DFD) as a key model in the software requirements engineering phase. However, DFDs have two possible notations (summarized in Figure 1). Users would prefer to model and view software models localized to their own cultures and education notations. Thus, the DFD notations used by different teams often raise confusion between SwinSoft members. **Figure 1** shows a summary of symbols that are used in developing data flow diagrams (DFD). SwinSoft is interested in rendering DFD diagrams with symbols suitable to

every team member. Furthermore, SwinSoft’s technical sales and business analysts usually discuss software requirements with both software customers (who mainly understand business details) and with the SwinSoft software architecture team (who are interested in more details and understand systems features and architectural details). Thus, they would like to be able to save time required to transform models developed for the customers to models used internally. Additionally, some analysts have iPads and want to use these to model DFDs with touch approaches. Some developers want to use large screens to view multiple DFDs and zoom, move and query elements highly interactively. To achieve these aims, SwinSoft is currently adopting traditional bidirectional model transformation between models and representations. However, this technique is time consuming and error-prone.


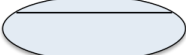





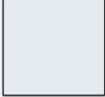
Symbol Name	Gane and Sarson	Yourdon
Process		
Data Flow		
Data Store		
External Entity		

Figure 1. Differences between Sarson and Yourdon DFD notations

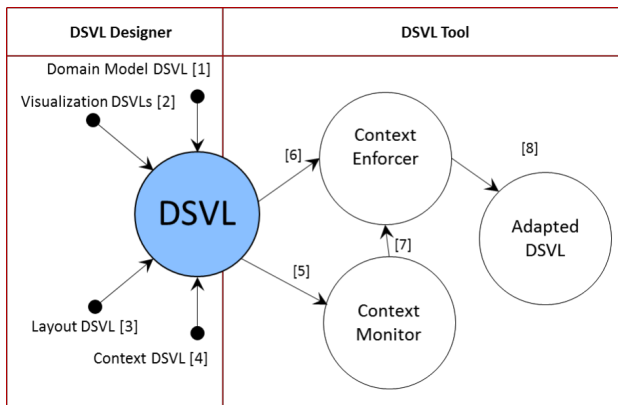


Figure 2. Context-aware DSVL Designer

### III. RELATED WORK

The area of domain-specific visual languages is one of the growing areas in model-driven engineering as it helps domain experts in fulfilling their tasks using their own terminologies at a desirable abstraction level. We categorize these efforts into domain-oriented DSVLs [2, 3], and DSVL design tools [12]. The existing DSVL design tools are categorized into [5]: (i) general-purpose graphical editing tools such as Unidraw, a C++ graphical tool development platform, and Eclipse graphical editing framework (GEF), DrawIO [7] a JavaScript web modeling tool delivers many visual notations for UML, BPMN, etc. While these efforts help in developing graphical

editors quickly, they do not capture domain model, entities, attributes, etc. Moreover, they require developers’ involvement in developing features of diagramming tools such as views of shared model entities; (ii) diagram generating toolkits such as DiaGen and VisualDiaGen [8], and VisPro. Such efforts use meta-models while others use visual grammars; and (iii) meta-modeling tools [9]. Some of these use meta-models and associated editor characterization as their source specifications. Some industrial tools such as Microsoft Visual Studio DSL designer provide a suite of tools for model creation, editing, and visualization. One of the key limitations with these tools is that DSVLs need to be built inside the Visual Studio or Eclipse modeling framework (EMF). Marama tool suite [5] is an Eclipse based DSVL designer with meta-modeling and visual designers. Marama has very limited context modeling and web-based modeling support. All existing DSVL design efforts do not support capturing, monitoring, nor enforcement of context information within DSVLs.

### IV. HORUSCML

Figure 2 shows the main components and basic workflow of our HorusCML context-aware DSVL designer. We have two phases: a design time phase, which is done by the DSVL experts using our DSVL Designer (steps 1-4), and a runtime phase where DSVL end users use the generated DSVL tool to fulfill their tasks (steps 5-8, automated by the underlying platform, HorusCML).

**DSVL Designer:** Our HorusCML DSVL designer is based on an existing web-based graph-editing tool (DrawIO [7]). We modified this tool to support definition of DSVLs via meta-DSVLs as in Marama, resulting in Horus. The Horus DSVL designer supports capturing domain meta-model as a graph instance, where each model entity is linked to a visualization symbol. Once the DSVL specification is completed, Horus extracts relevant details from the DSVL model (mainly nodes, and their corresponding shapes) and registers the generated DSVL within the tool itself. This way it becomes available for end users to use it along with other registered DSVLs. For this paper, we have improved Horus to support: (i) modeling one-to-many relationships between DSVL domain entities and visualization symbols; (ii) shape designing and mapping to domain entities’ attributes – i.e. how and where domain entity attributes should be visualized; (iii) specifying possible layout algorithms that should be applied with DSVL entities; and (iv) modeling context details that need to be satisfied on every domain entity, visualization symbol, or layout algorithm. The DSVL registration was modified to extract and incorporate these details within DSVL definition. Figure 3-A shows a part of the context-aware data flow diagram DSVL. The figure shows two DFD entities process and external entity (cloud shapes). The process entity is linked to two possible visualization entities. Because both visualizations are composites (have many attributes to be displayed), we have to put each one in a container and have the process entity point to such containers. The content of each container represents how a process entity should be visually represented. In the external entity we do not have this composite, so we point the external entity directly to the corresponding visual elements. Given that

we have two groups of notations (Yourdon and Sarson), we link each visualization element to the corresponding constraints – in our example, this is a user preference to use either Yourdon or Sarson. Figure 3-A shows that each visual element is connected to its preferred layout (here we exemplarily use three layout algorithms – layered, used with Sarson notations, force-based, used with Yourdon, and vertical layout, used with mobile devices). Figures 3-E,F show example DFD instances with different context information as we discuss in the next section.

**Visual Notation DSVLs:** The DSVL expert can visually design the shape (symbols) to be used in modeling the instances of a DSVL model. The expert creates a shape container and then drag and drop shapes from the toolbox onto a shape container. The shapes in the container are named with the corresponding domain entity attributes and are arranged in this container as needed. Thus, different visualizations of the same entity can reflect different abstract levels. Figure 3-A shows an example of a process visual element showing the fields (process Id, process name). This composite is visually designed from elementary visual elements available in Draw.IO (a part of Draw.IO toolbox is shown Figure 3-B).

**Layout DSVL:** The DSVL expert specifies how the DSVL model entities should be laid out when put together. In our DSVL Designer we make use of the KIELER Web Service for Layout (KWebS)<sup>1</sup>. It provides a variety of layout algorithms such as force-based layout, Sugiyama-style layout, circular, tree, etc. We abstract the service into our DSVL designer in a “Layout DSVL” where every element represents an abstract reflection of a layout algorithm. Designers can visually configure layout service attributes to be used when arranging model instances. Figure 3-C shows available layout entities.

**Context DSVL:** Once the DSVL expert has captured and modeled all DSVL details (entities, visualization and layout), they can define context information for certain scenarios – e.g. given an entity E, with two visual elements V1, V2, specify to use layout algorithm L1 when using V1, and layout algorithm L2 when using V2; the context DSVL helps in specifying constraints to be considered when adapting DSVL according to a given situation. We have developed our Context DSVL to take into consideration a set of context elements including: time of the day, location (important in GSD – global software development, and in communicating models with clients), device used to view the model (desktop, mobile, etc.), abstraction level the user is interested to work on or view, user permissions (What entities and attributes they can view or edit). Context-information could be marked as “mandatory” which means that it cannot be overridden by user preferences. Figure 3-D shows the Context DSVL toolbox. Figure 3-A shows that the DFD DSVL tool will use user preference (Sarson and Yourdon) to render the suitable notations. It also use user device to decide on the suitable layout – i.e. use force-based (with Yourdon), layered (with Sarson) on a desktop, and vertical layered (with Sarson) on mobile devices.

Both the Layout and Context DSVLs are extensible to incorporate new algorithms and context items. This requires updating the tool with the necessary information of the layout service – e.g. URL of the service and expected parameters. It also requires specifying how new elements’ can be monitored.

**Context Monitor:** The Context Monitor component is responsible for monitoring and providing both the current environment and user context information. Such information is always available for the Context Enforcer component. The context monitor triggers the Context Enforcer whenever there is an update in the tool environment or the user context. The current implementation status of this component is that it has predefined modules that collect context information. Figure 3-E,F show the same model instance rendered in two different environments. We plan to enable DSVL experts to extend this component, so that it can monitor customized context information based on newly defined context entities.

**Context Enforcer:** The Context Enforcer component is responsible for the DSVL context adaptation at runtime. With any interaction of the user with the DSVL tool or any change-of-status event raised by the Context Monitor, the Context Enforcer checks the DSVL specified constraints looking for satisfied constraints, and then fires their corresponding adaptation actions that should be enforced. Examples for adaptation actions include hiding certain domain instance entities because of security permissions, rendering entities using certain visualization symbols due to time/location/etc., and arranging entities using certain layout algorithms.

## V. CASE STUDY

For the evaluation we implemented an exemplarily data flow DSVL tool. The main requirements to satisfy are: (i) to support reflecting the same DFD using either Sarson or Yourdon notations according to the user preferences; (ii) to change the layout of a given DFD when using tablet or smart phone to the vertical layout to allow better readability on a small screen. To address the first requirement, we represented each entity with two visual elements and linked each visual element to a user preference. Thus, whenever the user updates his preferences, the tool will switch to the visualization that satisfies these constraints. To address the second requirement, we have specified two different layouts: one for the desktop version and the other one when viewing the model on a smart phone. Figure 3-A shows the DFD design with the specified requirements satisfied. Figure 3-E shows an example DFD for a banking system using Yourdon notations. Figure 3-F shows the same example rendered on a tablet (using Sarson for visual notations and vertical layout). The evaluation results reflect that HorusCML is capable to model, monitor and enforce context-aware DSVLs.

## VI. CONCLUSION

We have introduced a novel context-aware DSVL designer with tool support to design and realize DSVLs taking into consideration context information. This enables developing, editing, and viewing models refined to fit within user context, experience, level of abstraction, device used, time, location,

---

<sup>1</sup> <http://rtsys.informatik.uni-kiel.de/confluence/x/nQEF>

etc. HorusCML supports modeling and integrating context-information within different DSLV aspects including domain, visualization, and layout models. We have evaluated our tool in developing a context-aware DFD DSLV tool that takes into consideration context information when rendering (visual representation and layout) data flow diagrams. We plan to apply our approach in more case studies including security engineering and scientific applications development.

### VII. ACKNOWLEDGMENT

This research is supported by the Australian Research Council under Discovery Project DP120102653. Ulf Rüegg is funded by a doctoral scholarship (FITweltweit) of the German Academic Exchange Service.

### REFERENCES

[1] M. S. Teixeira, R. A. Falbo, and G. Guizzardi, "Can Ontologies Systematically Help in the Design of Domain-Specific Visual Languages?," in *OTM 2013 Conferences*. vol. 8185, Eds., ed: Springer Berlin Heidelberg, 2013, pp. 737-754.

[2] J. P. Diprose, B. A. MacDonald, and J. G. Hosking, "Ruru: A spatial and interactive visual programming language for novice

robot programming," in *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2011, pp. 25-32.

[3] M. Almorsy and J. Grundy, "SecDSVL: A Domain-Specific Visual Language To Support Enterprise Security Modelling," in *Australian Conference on Software Engineering Sydney*, 2014.

[4] D. Moody, "The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 35, pp. 756-779, 2009.

[5] J. C. Grundy, J. Hosking, et al, "Generating Domain-Specific Visual Language Tools from Abstract Visual Specifications," *IEEE Transactions on s/w Eng.*, vol. 39, pp. 487-515, 2013.

[6] M. Almorsy, J. Grundy, R. Sadus, et al, "A Suite of Domain-Specific Visual Languages For Scientific Software Application Modelling," in *2013 IEEE Symposium on Visual Languages and Human-Centric Computing*, San Jose, CA, USA, 2013.

[7] JGraph. (2013). *Draw.IO*. Available: <https://www.draw.io/>

[8] M. Minas, "Visual Specification of Visual Editors with VisualDiaGen," in *Applications of Graph Transformations with Industrial Relevance*. vol. 3062, Eds., ed: Springer Berlin Heidelberg, 2004, pp. 473-478.

[9] J.-P. Tolvanen and M. Rossi, "MetaEdit+: defining and using domain-specific modeling languages and code generators," in *Proc. of 18th Conf. on Object-oriented programming, systems, languages, and applications*, CA, USA, 2003.

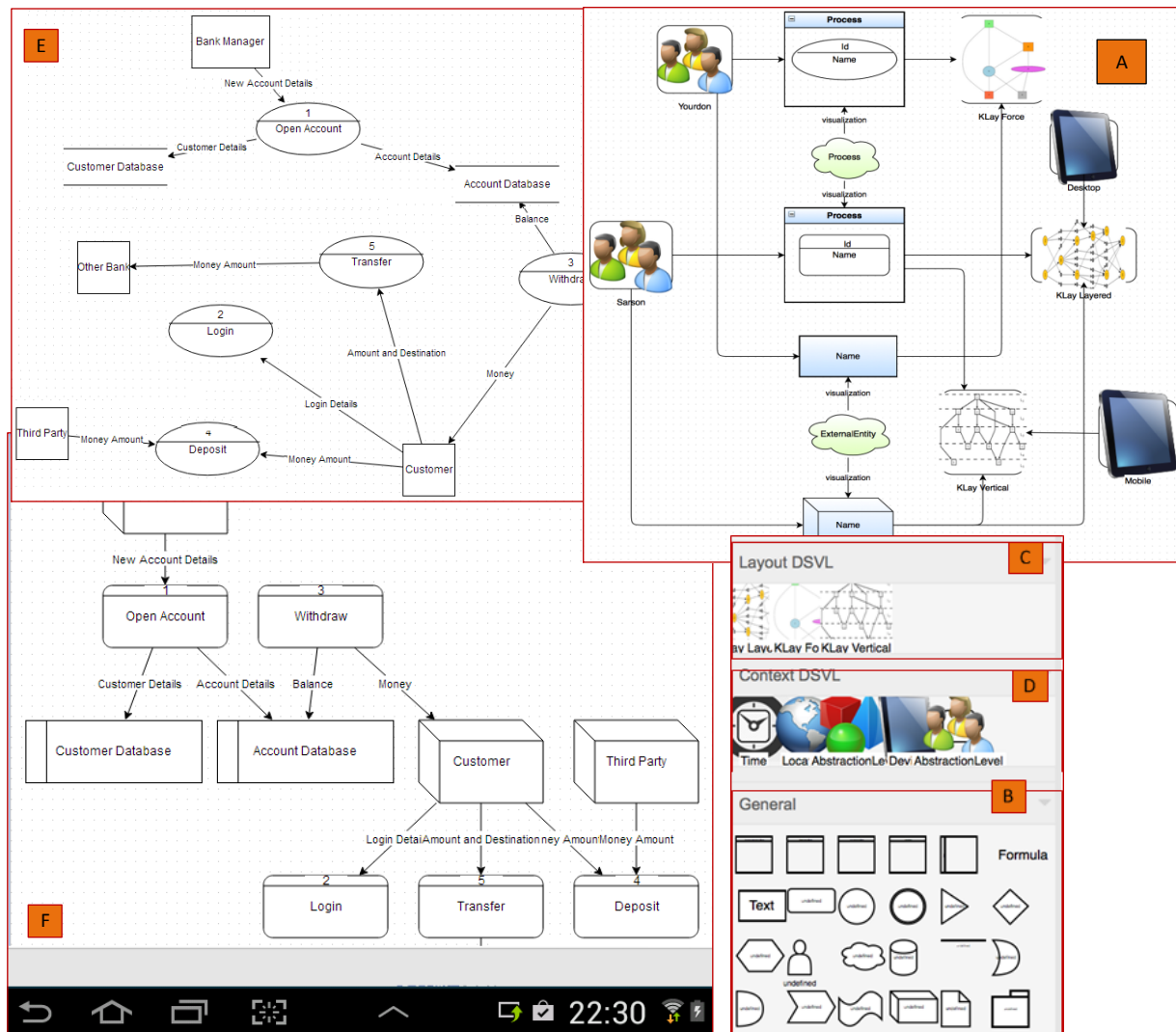


Figure 3. Snapshots of the context-aware DFD DSLV design and runtime