Visual Programming Meets Domain Modeling: The ADOM-Tersus Synergy

Ofer Brandes[1], Youval Bronicki[1] , Iris Reinhartz-Berger[2], and Arnon Sturm[3]
[1] Tersus Software Ltd.,
Herzeliya 46000, Israel
Ofer.Brandes@tersus.com, Youval.Bronicki@tersus.com
[2]Department of Management Information Systems,
University of Haifa, Haifa 31905, Israel
iris@mis.haifa.ac.il
[3]Department of Information Systems Engineering,
Ben-Gurion University of the Negev, Beer Sheva 84105, Israel
sturm@bgu.ac.il

1 Introduction

Visual programming refers to programming that uses a visual representation. It enables the manipulation of visual information, supports visual interaction, and allows programming with visual expressions [5]. Another definition of visual programming is that it includes any system where the user writes a program using two or more dimensions [8]. According to a commonly used classification [2], visual programming languages (VPLs) may be further classified into icon-based languages, form-based languages (e.g., Forms/3), Hybrid Text/Visual (e.g., Rehearsal World), Constraint-Oriented languages (e.g., ThingLab, ARK), Programming-by-Example languages (e.g., Pygmalion) and pure visual languages (e.g., Prograph, PICT/D, Cube). This proliferation of VPLs developed as a result of understanding VPL advantages including fewer programming concepts, concreteness, explicit depiction of relationships, immediate visual feedback, lack of necessity to visualize a program in a sequential manner, elimination of an intermediate step in the process of creating a program, less emphasis on syntax, navigable program structure, executable partially specified programs, and integration of pictorial clues, as indicated by [3].

In this paper we propose an extension to a VPL by enriching it with domain engineering capabilities in order to increase its effectiveness. Domain Engineering [4, 11], supports the notion of *domain*, a set of applications that use common concepts for describing requirements, problems, capabilities, and solutions. The purpose of domain engineering is to identify, model, construct, catalog, and disseminate a set of software artifacts that can be applied to existing and future software in a particular application domain [10]. As such, it is an important type of software reuse, validation, and knowledge representation [7]. In particular, in this paper, we present Tersus, a VPL framework, and its domain engineering enhancements utilizing the Application-based DOmain Modeling (ADOM) approach.

2 The Tersus Visual Programming Platform

The Tersus visual programming platform is a development environment, a collection of model libraries and an execution engine – all based on the Tersus modeling language – used to develop various types of software applications [1, 14]. The Tersus platform allows a developer to “draw” the software rather than write it down as a text. **Creating an application** is done by defining a hierarchy of visual models, in which each model may be composed of lower level components. The developer starts at a top-level diagram representing the whole system, and then continues with an iterative top-down refinement process, drilling down from each model to specify its components. Employing an “infinite drawing board” that displays graphically the whole model hierarchy, the developer can fully and precisely specify the required business logic in a visual and intuitive manner. **Deploying an application**, once modeled, is immediate. The models, saved as XML files, are read by the Tersus server, which executes the functionality defined by the models (including user interface, client-side behavior and server-side processing). For debugging and troubleshooting purposes, it is possible to record the full details of the execution and then trace them graphically in the model diagrams. **Maintaining an existing application** is done by amending its model – changing the business flow, adding new components, or disabling redundant components. Upon completion of the required modifications, the application can be redeployed immediately.

The model of a typical application consists of Systems (high level modules), Displays (GUI components), Processes (activity units), and Data Structures and Data Items (information used by the application). Processes (and in certain cases also systems and displays) can receive and send out data through Slots. The flow of data between processes, as well as the sequencing of processes, is governed by flows (a flow appears in the model diagram as an arrow between two model elements, each of which is a slot or a data element.). In fact, the Tersus modeling language is more than a visual language. As suggested by Fred Lakin [6], the Tersus platform provides “Executable Graphics”, where the Tersus server is the execution engine that runs the modeled applications.

Tersus is used to create a range of software solutions, mainly web applications. Companies who choose the Tersus platform have done so due to its ease of use, increased productivity, and the high quality of the resulting applications. The usage of Tersus also widens the development teams, as it helps less skilled developers take part in the development process.

3 The Application-based Domain Modeling Approach

The Application-based DOmain Modeling (ADOM) approach is based on a three layered architecture: application, domain, and (modeling) language. Influenced by the classical framework for metamodeling presented in [9], the applic*ation layer*, which is equivalent to the model layer (M1), consists of models of particular applications, including their structure and behavior. The *language layer*, which is equivalent to the metamodel layer (M2), includes metamodels of modeling languages. The modeling languages may be graphical, textual, mathematical, etc. The only requirement from a modeling language to be used with ADOM is that it has a classification mechanism, which enables categorizing elements according to other elements. The intermediate *domain layer* consists of specifications of various domains. These specifications include the main concepts of the domain and the relations among them. The allowed variability of domain specific applications is specified in ADOM by attaching multiplicity constraints to the various domain concepts. These attachments are done by the classification mechanisms provided by the (modeling) languages used with ADOM. For example, most of the e-commerce applications handle a kernel of activities, such as browsing a catalog, issuing requests, paying, processing delivery, and confirming shipment. The main concepts that they all share are merchants, customers, online catalogs, shopping carts, orders, prices, payments, etc. However, e-commerce applications may also be very different: e-stores vs. auction sites, B2B vs. B2C applications, and so on. In this context, the domain model defines what is allowed, possible, or recommendable while specifying e-commerce applications. Furthermore, the domain model provides a semantic validation template for the applications in the domain.

The ADOM architecture also enforces constraints among the different layers, or more precisely the domain layer enforces constraints on the application layer, while the language layer constrains both the application and the domain layers. Additional information on ADOM appear in [12,13]

4 The ADOM-Tersus integration

The actual usage of Tersus has generated many best practices of modeling and working methods, resulting in model libraries. These model libraries contain building blocks for assembling applications as well as larger modules that can be used within new applications or as skeletons for new applications. In Tersus, many library models are provided as templates. A *template* is a generic/typical model that can be copied into the model hierarchy, and then, if required, modified to fit the specific needs at hand. As such, templates are useful for capturing best practices as well as domain knowledge. Nevertheless, templates are not enough to enforce best practices and modeling conventions. In large systems, it is important to develop sub-systems in a consistent manner and ensure modifications to a template are propagated to all models created from this template. Following that rationale, Tersus has adopted the mechanism suggested by ADOM. In the case of Tersus (as the modeling language), the classification of elements is done using prototypes. A *prototype* is a special kind of a template, which has multiplicity properties (e.g., exactly 1, or 2-4). When a prototype is copied to create a specific model, each of its components is duplicated according to its default multiplicity (e.g. multiplicity ‘+’ indicated at least one instance is required, and 1 is created by default).

A model created from a prototype maintains a reference to its prototype, which enables easy maintenance of the constraints implied by the prototype. This increases productivity by: (1) validating a model to verify its consistency with the constraints imposed by its prototype at any point in time (where violations are graphically highlighted in the model diagram); and (2) maintaining model consistency and prototype conformance when updating the prototype at a later stage. Furthermore, a modeler can easily add instances of a component (up to the maximal number of instances allowed by the prototype). Tersus Studio suggests to the modeler all components that may be added to a model according to the prototype associated with it, and the modeler can add any of them by a single mouse click. Specifically, this enables the creation of prototypes with optional components. An optional component is not created by default when a model is created from a prototype, but the modeler can choose to add it at any later stage.

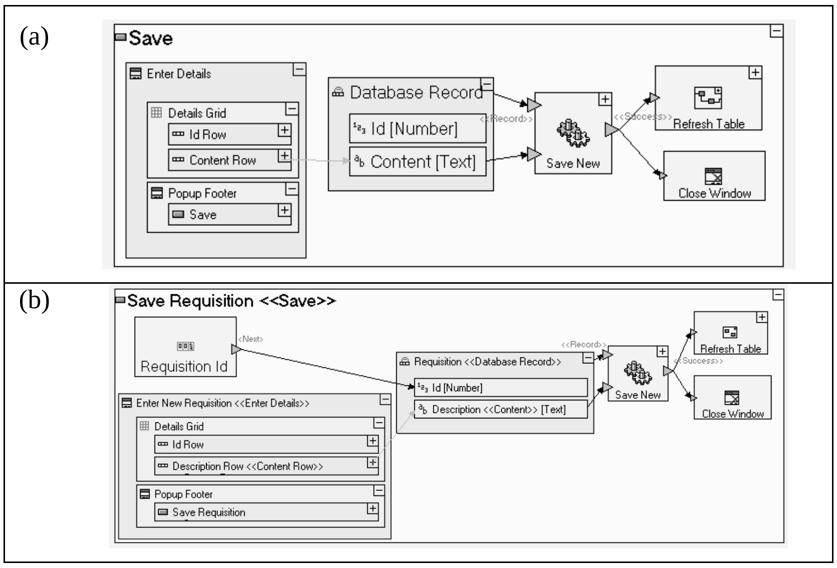

Figure 1. (a) **Save** button – part of the **Table Display and Management** domain model; (b) **Save Requisition** button – part of a requisition management application.

In Figure 1, a component of the **Table Display and Management** domain model is presented (Figure 1a) along with a corresponding component of a requisition management application model (Figure 1b). **Save** is a process prototype that creates a database record by populating one of its fields from data entered by the user through a pop-up window. Then, the process writes the record to the database, closes the pop-up window and refreshes the display of the records. **Save Requisition** is a component of a requisition management system, classified as **Save**, which creates a simple record with the description of a purchase requisition. In that model each application element is classified with its corresponding prototype model element. In places were no changes are made with respect to the domain model, the classification is not displayed. Note that the **Save Requisition** model contains an additional action **Requisition Id**, which does not appear in the prototype, to generate the record's identifier. Following the ADOM approach, a modeler can add to each application elements that are not part of the prototype from which it has been created, as long as this does not violate constraints explicitly imposed by the prototype.

Using the ADOM approach has helped the Tersus platform become more productive and produce more consistent software. Prototypes are used to define modeling patterns and to enforce compliance to modeling conventions like ensuring consistent look and feel of screens or enforcing full logging of updates to provide audit trail.

References

1. Bronicki, Y., Brandes, O., Raskin, Y., Shaked, Y., Szekely, S. “Method, a language and a system for the definition and implementation of software solutions”, United States Patent Application 20040107414, 2004.
2. Burnett, M. and Baker, M. “A Classification System for Visual Programming Languages”, Journal of Visual Languages and Computing, 5 (3), pp. 287-300, 1994.
3. Burnett, M. “Visual Programming”, Encyclopedia of Electrical and Electronics Engineering In John G. Webster, editor, Encyclopedia of Electrical and Electronics Engineering. John Wiley and Sons Inc., New York, 1999.
4. Carnegie, M. “Domain Engineering: A Model-Based Approach”, Software Engineering Institute, http://www.sei.cmu.edu/domain-engineering/, 2002.
5. Golin, E. and Reiss, S. “The specification of visual language syntax”. Journal of Visual Languages and Computing, 1(2), pp. 141-157, 1990.
6. McIntyre, D. Visual Languages, http://www.hypernews.org/~liberte/computing/visual.html, 1994.
7. Meekel, J., Horton, T. B., France, R. B., Mellone, C., and Dalvi, S. “From domain models to architecture frameworks”, Proceedings of the 1997 symposium on Software reusability, pp. 75-80, 1997.
8. Myers B. “Taxonomies of Visual Programming and Program Visualization”. Journal of Visual Languages and Computing, 1, pp.97-1231990.
9. OMG-MOF, “Meta-Object Facility (MOF™)”, version 1.4, 2003.
10. Pressman, R.S. Software Engineering: A Practitioner's Approach, 6th Edition, New York: McGraw-Hill, 2004.
11. Prieto-Diaz, R. Domain Analysis and Software Systems Modeling. Los Alamitos, CA: IEEE Computer Society Press, 1991.
12. Reinhartz-Berger, I. and Sturm, A. “Behavioral Domain Analysis – The Application-based Domain Modeling Approach”, the 7th International Conference on the Unified Modeling Language (UML'2004), LNCS 3273, pp. 410-424, 2004.
13. Sturm, A. and Reinhartz-Berger, I. “Applying the Application-based Domain Modeling Approach to UML Structural Views”, the 23rd International Conference on Conceptual Modeling (ER'2004), LNCS 3288, pp. 766-779,2004,.
14. Tersus Web site, http://www.tersus.com/, 2006.