Tersus Visual Programming Platform

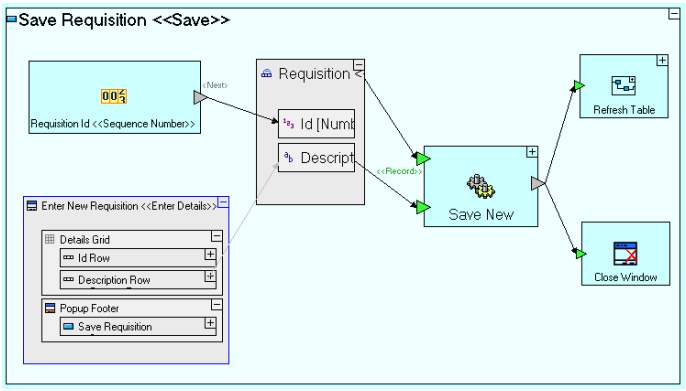
Ofer Brandes
Tersus Software Ltd.
Herzliya, Israel
ofer.brandes@tersus.com

Youval Bronicki
Tersus Inc.
Santa Clara, California, USA
youval.bronicki@tersus.com

Abstract—**A visual programming language that has emerged from real life needs to become a general purpose software development platform. Providing a unified language for modeling user interface, client-side behavior and server-side processing, it is mainly used by individuals and enterprises to develop web applications as well as cross-device mobile applications.**

I. Executable Graphics

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], 4]. The Tersus platform allows a developer to "draw" the software rather than program it by code. 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. The ability to zoom from one level of the model to lower levels and back practically overcomes the Deutsch Limit regarding the ability of visual programming to represent highly complex systems [3].



To borrow Fred Lakin's term [2], the Tersus platform provides "Executable Graphics". The models are saved as XML files, are read by the execution engine, which executes the functionality defined by the models.

II. A Product Emerging from Real Life Problems

The Tersus Visual Programming Platform has emerged from over a decade of development. Our initial goal, back in 1998, was to develop a data transformation engine to convert financial messages from one format standard to another without writing code (i.e. by business experts rather than programmers). The solution was to define message structures as hierarchies of data elements and to define a transformation between such structures as a set of field-to-field (or multiple-fields to field) atomic calculations. At runtime we performed parsing of each input message and applied the relevant transformation rules to create the appropriate output message.

Very quickly it became apparent that in real life systems, message transformation is just part of a larger business process, and we enhanced our platform to a full scale dataflow system that also enabled the definition of processes, conditional flows and iterations.

The last stage in this evolution was to add the ability to model the GUI of applications, thus making the platform a general purpose software development platform.

III. The Terus Modeling Language

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) are able to 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).

IV. Unified Modeloing Language

AJAX is a rather complex technique that involves client side programming in one language (usually Javascript) and a different language (like Java or PHP) on the server side. Tersus uses AJAX techniques "under the hood", and hides this complexity from the developers, who can use the same modeling language to define user interface, client-side behavior and server-side processing.

V. More Than Yet Another Dataflow Language

The Tersus Modeling Language is a dataflow language, using data flow diagrams and defining applications in terms of the data flowing between operation components.

But the Tersus language has a significant additional expressive power over other dataflow languages, equivalent to the expressive power of the leading procedural and object oriented language. This extra expressive power stems mainly from 3 sources: (1) In traditional dataflow languages, an operations component is a "black box" with inputs and outputs, all of which are always explicitly defined, and it is run as soon as all its inputs become available. In Tersus, optional input slots can be specified. (2) Tersus allows the modeling of arbitrarily complex data structures, and flows can be defined from/to any element or sub-element of a data structure. (3) Tersus supports type inference.

VI. Reuse, Templates and Prototypes

Any model defined in Tersus can be reused in multiple contexts. This is equivalent to a method that is called from various places in the code of a computer program.

A **Template** is a generic/typical model that is cloned rather than reused. This allows the developer to benefit from best practices and domain knowledge of other developers. The Tersus libraries of models contain many templates of useful models, including atomic models provided as part of the platform (mathematical and textual operations, GUI components, database access components, etc.).

A **Prototype** is a special kind of a template, which has multiplicity properties for each of its elements (e.g., exactly 1, or 2-4). When a prototype is cloned to create a specific model, each of its elements is duplicated according to its default multiplicity. A model created from a prototype maintains a reference to its prototype, which enables easy maintenance of the constraints implied by the prototype and increases productivity.

VII. Integrations

A Tersus-based application can be a standalone application, but may also integrate with other applications. The platform supports various methods of integration, including interaction with multiple databases (any database supporting JDBC), HTTP requests, and SOAP web services (a WSDL file defining SOAP web services can be imported to create Tersus models, which are then used like any other model).

VIII. Developing Cross-Device Mobile Applications

While Tersus is a general purpose language, it has been mostly used in the last years by individuals and enterprises to develop rich web applications, ranging from small ad-hoc applications to complex multi-user ERP systems.

In the last couple of years, with the rise of iPhone and Android devices, smartphones became capable of running Tersus applications as well, and Tersus is also used to develop cross-device mobile applications (i.e. the developer develops the application once, and then deploys it on either an iPhone device or an Android device).

IX. Who Can Use Tersus?

Programmers skilled in mainstream programming languages are, of course, the natural candidates to use visual programming, although some of them resent the notion. Yet, as Tersus uses flow diagrams, which can be understood by non-programmers, Tersus is also used by "soft developers" (like application designers and web site builders).

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 by using a visualizable computer executable modeling language", United States Patent 7,694,272, 2010.

[2] R. Baeza-Yates, Comments on Visual Programming, http://www.dcc.uchile.cl/~rbaeza/cursos/vp/todo.html.

[3] D. McIntyre, Visual Languages, http://www.hypernews.org/~liberte/computing/visual.html, 1994.

[4] Tersus web site, http://www.tersus.com, 2006-2011.