Towards Declaratively Defined Graphical Editors

O. Beaudoux, F. Jouault, ESEO/TRAME



• Interactors Perspectives

 Anatomy of a Graphical Editor Prototype 1: Loa-JS Prototype 2: TCSVG • Prototype 3: AOF Diagram

Anatomy of a Graphical Editor

• Model (i.e., abstract syntax) • Views • Templates (i.e., "shape" groups) transformed from model • Geometrical constraints (on shapes using model) • Auto-disposition (i.e., layout, routing) • Interactors: interactions to actions • Actions: changes to model Interactions: user events on views

e.g., geometric constraints from semantics (e.g., align elements having a specific relationship)



nteractors

• Maturity • Defined in the '80 by Brad Myers • Transform an interaction into an action (on a domain object) • Especially useful in the scope of diagram authoring apps • Implementations Loa prototype provides simple and predefined interactors \bigcirc Users parameterize interactors by leveraging active operations • Planned • Support modern interactors (e.g., touch-based ones) - easy improvement • Extensions to Consider Support user definition of interactors/interactions



Prototype 1: Loa-JS Web-based Graphical Editors

• Views

Model: custom (JavaScript-based, EMF-like)

• Templates: XML (e.g., HTML, SVG) Auto-disposition: not supported parameterizable interactors • Actions: in JavaScript

incrementally transformed with active operations • Geometrical constraints: recently used Cassowary Interactors: library of predefined and

Loa-JS Demo





Loa: a DSL for Building Interactive Systems Based on the Design of Linked Models Olivier BEAUDOUX^{1,2}, Frédéric JOUAULT¹

"Graft" clase study, see below}, readability, and platform independence.

Loa Parts: the Faces

Loa splits interactive systems into four orthogonal parts which can be represented as the faces of a tetrahedron. The concrete parts (respectively the abstract parts) represent the faces that users directly perceive (respectively do not perceive).

Abstract Part

overall interactive application.

 \leftrightarrow

Actions define which modifications users can perform on the domain data.

Concrete Parts

Interactions define which interaction users can choose to perform a given action.

Templates define the concrete appearance of objects, as presented to users.

Graf Case Study: Faces

A case study called "Graf" has been built to illustrate the relative easiness of Lea use. The figure at left shows - the application running into a Web navigator. It has been generated from the Loa code given below (see 🦠 blocks). The table at right shows 🛈 objects and templates that the developper has to write, and 🥨 actions plus interactions that he/she has to choose from Loa library



dass Graph {
elements : set <= ement>

abstract class Element {

label: one<string>

x, y: one dint>

outgoing.source,

source.outgoing,

class Node extends Element (

incoming...target : set < Arc >

target...incoming : opticNode>

class Arc extends Element {

Actions Create, Edit, Erase

Containers Contaivers are intensively used for class Carvas (modeling the four faces of interactive systems. backLayer : set cline> Unique VVV {backLayer} (frontLayer) 👒 Objects

abstract class Shape { -x, y: one cint>

class Circle extends Shape { coincle r='20' fill='blue' _/> </RAD

class line extends Shape { x2, y2 : one <int>

Jetails on flyer:

¹ TRAME Team, ESEO Group, Angers (France) –² DiverSE Team, IRISA/INRIA, Rennes (France)

Abstract

Building interactive systems is still a complex development task that requires the use of platform-dependent languages. The Loa DSL has been designed to tackle such a complexity by focusing on the design of interactive system. models and their links, rather than on implementing them using a GPL like JavaScript. Loa decreases the programming effort on three points: verbosity (number of lines reduced from 300 in JavaScript to 200 in Loa for the



Prototype 2: TCSVG Web-based Graphical Viewers

Model: none • Views • Templates: in SVG • Geometrical constraints: using Cassowary Auto-disposition: not supported • Interactors • Actions: not applicable Interactions: only drag-drop-move

TCSVG (Trame CSVG) is a prototype reimplementation of CSVG (SVG + constraints) in Javascript, and based on Cassowary. It is not intended to be compatible with the original CSVG.





GridBagLayout UML Diagrams Composite Structure • Sequence: 1, 2, 3 • Activity: 1, 2, 3, 4

dcmCom				
dcmCon				
	batter			
	ounci			





Prototype 3: AOF Diagram **Graphical Editors in Java**

• Model: EMF • Views • Templates: in Java incrementally transformed with active operations • Geometrical constraints: using AOF • Auto-disposition: not supported • Interactors: a few predefined interactors O Actions: in Java



AOF Diagrams Demo

• Graf UML Diagrams

New Message New Object EnableHovers

12C	bus		I2C slave ada
		start	
	byte	e(slaveAddre	ss << 1)
		ack	



Sequence

• State (with actual backing EMF model)





Some Wild Ideas

• Web-based interactive viewers o e.g., for Papyrus exports, letting users change layout outside of the tool • Web-based Papyrus (SVG-based) Projectional editors o e.g., for textual elements in UML syntax such as Features, Transition labels, etc. • to provide document-like views on models

Questions?



adapter	Regi	sters
read(regAddress + 1)	\rightarrow	
read(regAddress + n)		
regValue		