Visual Languages for Event Integration Specification

Na Liu
Department of Computer Science, University of Auckland
Private Bag 92019, Auckland, New Zealand
+64 9 3737599 ext. 82128
karen@cs.auckland.ac.nz

ABSTRACT
We are exploring existing approaches and developing new techniques for visual event-based system integration. We are using domain-specific visual languages with different high-level visual metaphors (including Tool Abstraction, Event-Query-Filter-Action and Spreadsheet) to specify event-handling support and provide backend processing tool support for event integration specification and visualisation of event propagation. We aim to generalise from three exemplar visual event-driven system metaphors and develop a new, generic visual event handling metaphor. From this we will build a visual environment for specifying event-based system integration. The visual metaphor we are developing should adapt the event-based communication model to a wide range of application domains, and also should support complex and intelligent system design and implementation.

Categories and Subject Descriptors
D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering, User interfaces; D.3.2 [**Language Classifications**]: Design languages, Very high level languages

General Terms
Design, Language, Human Factors

Keywords
Domain-specific visual language tools, Meta-CASE tools, Event Handling

1. INTRODUCTION
Event-driven systems are ubiquitous and have demonstrated advantages in loose-coupling system behaviours. Example event-driven systems include GUI systems, simulation systems, real-time management systems and composition systems. Such systems all incorporate events of interest, conditions ("filters") on whether to respond to the event and action(s) to run that may modify the system state. Example approaches for specifying event-handling include scripting, Event-Condition-Action (ECA) rules, and spreadsheets (single direction constraints).

One example event-driven problem domain is web services composition. Web services have become a popular technology for building distributed systems, but there is a lack of languages and tools to specify web service compositions at high abstraction levels, generate lower-level executable process code such as BPEL4WS, and visualise, at high abstraction levels, running web services. Most approaches provide basic flow-like BPEL4WS editors or similar [15], [17]. More abstract approaches [4], [5] only support limited compositional approaches or do not support generation of BPEL4WS or similar executable forms. We have developed a new approach for complex web service composition using a high-level metaphor and visual language (called ViTABaL-WS) [10]. This supports higher level design views for service composition that are complementary to current web services composition standards.

The second problem domain that we have focussed on is visual design tools. These tools have many applications, including software design, engineering product design, E-learning, data visualisation, and tourism. Pounamu [19] is a meta-tool we have developed for building such visual design tools. It incorporates high-level visual specifications of tool meta-models and visual language notations allowing end users to modify aspects of their tools such as appearance of icons and view compositions. However, commonly end users also wish to modify tool behaviour [12], [14] to specify editing constraints, automated diagram modification, semantic constraints, and computation. Several approaches have been used to support reconfiguration of diagramming tools, including direct modification via an API [8], scripting [13], programming by demonstration [2], and Event-Condition-Action rule based languages [1], [9]. Pounamu currently uses the first approach. Many end users of such tools are not programmers and do not wish to learn or use textual, programmatic scripting languages to tailor their design tools.

Most visual design tools are "event driven", i.e. when a user modifies a diagram, events are generated and can be acted upon to modify other diagram content, enforce constraints, etc. We have used the event-driven nature of such tools as the basis for an end user domain specific visual language, Kaitiaki, with which to express both simple and complex event handling mechanisms via visual specifications for their diagramming tools [11]. These include event filtering, tool state querying and action invocation.

MaramaFormula uses a more declarative approach to extend behaviour specification of visual design tools. The focus is to better model relationships in a tool's meta-model definition. This includes constraining relationships via connector types mapping and arities, and specifying formulae for calculating property values and enforcing constraints. Formula construction is similar to a spreadsheet but expressed at a type rather than instance level. Formulae are all interpreted as one way constraints with Java event handler code generated and realized at a model instance level. Errors and to-do list critics provide notification to the user. Visualisations of formula effects are achieved via runtime visual debugging and master-details tabular model instances data views.

Based on the in-depth exploration of the three above visual event-based metaphors, we aim to generalise to a metaphor and a

ICSE'06, May 20-28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

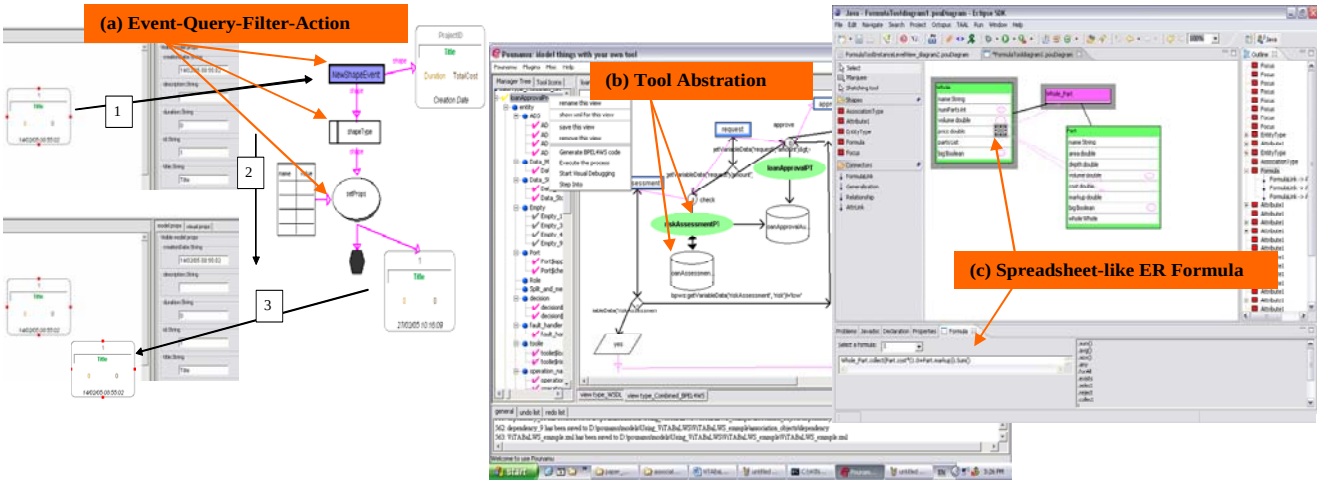

Figure 1 Kaitiaki (a) storyboard, ViTABaL-WS (b) editing in Pounamu, and MaramaFormula (c) editing in Marama.

language/framework that can provide support for generic event integration specification. The generalised approach should incorporate compositional primitives as building blocks and different communication relationships between them. It also should contain mapping/integration schemes as crossover between the above mentioned approaches.

2. OUR APPROACH

As stated above, our aim is to generalise from three exemplars to produce a generic event handling specification visual language and supporting environment. We review each exemplar before indicating our directions for their generalisation.

2.1 ViTABaL-WS

We wanted a metaphor to effectively describe the composition of web services and support the development of a visual language and modelling environment. The web services compositional relationships can be very complex and a range of compositional building blocks are required. We chose to use the "Tool Abstraction" (TA) paradigm [7] as our metaphor for web service compositions and to support reasoning about different relationships between compositional primitives. The TA paradigm is a message propagation-centric approach describing interconnections between "toolies" (the encapsulation of functions) and "abstract data structures" (ADSs: the encapsulation of data) which are instances of "abstract data type" (ADTs: typed operations/messages/ events). Connection of toolies to other toolies and ADSs is via typed ports. The TA paradigm supports modelling data flow, control flow and event flow relationships. Reusability, extensibility and expressiveness are key advantages possessed by TA.

ViTABaL [7] is a hybrid visual programming environment that we had previously developed for designing and implementing TA-based systems. It uses the TA paradigm to compose systems by integrating, and coordinating toolies and ADS components. TA paradigm appeared to us to be well suited for the web services composition domain by permitting specification of an abstract model involving a series of co-ordinated invocations to web services operations. Accordingly we adapted our earlier work to develop a new visual language and environment, ViTABaL-WS, which specialises the ViTABaL visual composition language to the domain of web services composition. It supports modelling of both event-dependency and dataflow in designing complex web service compositions. Figure 1 (b) is a ViTABaL-WS diagram illustrating examples of compositional primitives in the Tool Abstraction paradigm. Toolies (web services, shaded green ovals) encapsulate data processing and interact with each other through both direct and indirect operational invocations using shared data structures (message ADT instances: rectangular, shaded icons); and event-driven dependencies indicating state changes to a Data Store ADS (data storage service). A system of typed input and output ports on toolie and ADS services provide message sources and sinks. Services are wired together using these ports with ports supporting only certain kinds of connection and message ADTs. Messages generated by a service output port are distributed to connected web service input ports. Many interconnection schemes are supported including one-way flow, request-response, asynchronous flow, and subscribe-notify. Additional controls support conditional flow, dynamic type checking, synchronisation, iteration etc.

The specified web services are linked together by composition rules enforced in the ViTABaL-WS tool. ViTABaL-WS supports generation of WSDL and BPEL4WS from its abstract composition model. We use the Business Process Execution Language for Web Service Java Run Time (BPWS4J) as the deployment engine for generated BPEL4WS processes. A deployed process is provided with a SOAP interface and a WSDL file, and thus can be invoked by a requesting web service client. BPWS4J is tightly integrated with ViTABaL-WS: a ViTABaL-WS process can be directly deployed and step-by-step visualization of its execution can be obtained, with running process state information shown in ViTABaL-WS diagrams.

2.2 Kaitiaki

Our Pounamu meta-tool provides a textual code-based event handler specification tool unsuitable for end users. We wanted to replace this with one using a visual language suitable for non-programmer end users. To develop this replacement visual language, *Kaitiaki*, and its specification tool we carried out an analysis of Pounamu event handlers from a wide range of tools

to identify key constructs used to specify different tool behaviours. All had aspects of (1) specifying the event(s) of interest; (2) querying the tool state in various ways; (3) filtering event/query results and making decisions; and (4) performing state changing actions on filtered objects. We also looked at the metaphors used in existing rule-based and event-condition-action event handler specification tools to see how these manifested the behavioural specifications and how suitable these were for end users. From this analysis and survey, we developed a set of key requirements and design approaches for our new *Kaitiaki* visual event handler designer:

- A need to represent key "building blocks" of state query, data filtering and state modification (actions).
- A need to represent event objects and their attributes; various objects from the Pounamu tool state (both view and model); and query results (typically collections of Pounamu state objects).
- A need to represent "data" propagation between event, query, filter and action representations.
- A need to represent iteration and conditional data flow.

The metaphor used by *Kaitiaki* is an Event-Query-Filter–Action (EQFA) model conceptually interpreted as: an end user selects an event type of interest; adds queries on the event and Pounamu tool state (usually diagram content or model objects that triggered the event); specifies conditional or iterative filtering of the event/tool state data; and appropriate state-changing actions to be performed. Complex event handlers can be built up in parts and queries, filters and actions can be parameterised, and reused. Ordering is handled by dependency analysis in the code generator. Domain specific tool icons are also incorporated into the visual specification of event handling as placeholders for the Pounamu state, to annotate and make the language more expressive (as shown in Figure 1 (a)). Step-by-step visualizations of EQFA element invocation and data propagation are supported for incremental development and debugging of visual event handler specifications.

2.3 MaramaFormula

We have adopted a spreadsheet-like metaphor to construct meta-model formulae as another approach to specifying visual design tool event handling. A formula is constructed by clicking on entity-relationship metamodel elements (i.e. entity type, association type, and attribute) in a metamodel view and a list of library provided functions as shown in Figure 1 (c). Formulae can be attached to an element in the metamodel and detached or removed from it. Context and dependency relationships regarding to a constructed formula are automatically inserted/updated reacting to user's clicking actions. Constraints on clicks are also enforced to complement design time semantics. Users can choose to show or hide selected formulae in the view. Bi-directional consistency between a visual formula and the corresponding textual entity-relationship formula is supported by parsing textual changes to the formula and modifying the graphical view. Cycle detections are possible while a formula is constructed and de-cycle options at design time are provided to aid error handling.

The visually specified metamodel level formulae are compiled into data structures and java event handlers to be applied to selected mode views, and are attached to those model views as realised event handlers. The data structures incorporate lists of composed primitive elements of the formulae and executable building blocks to generate java event handler code bits.

We adopt the same runtime visualization technique (i.e., visual debug and step into) as in ViTABaL-WS and Kaitiaki to visualize formula effects, with a complementary tabular display of instance values as in a spreadsheet, where master-details of related data are shown with formulated columns non-editable prompting tool-tips to show their non-interpreted formulae.

2.4 Generalisation

We are working on generalising from the above three exemplar approaches to develop a metaphor and a language and provide tool support for generic event integration specification. Our generalization approach employs the Three Examples pattern of the Evolving Frameworks Pattern Language [3]. By abstracting from the three exemplars, a general meta-model representation that combines atomic primitives (either shared or non-shared) extended by the three visual languages will be defined. This common model will support multiple metaphoric views in the style of the three exemplars and will support generation to a range of underlying implementation technologies for execution or interpretation (OCL [18], RuleML [16], stylesheets etc.).

3. CONTRIBUTIONS

We have evaluated and extended the current event-handling approach in the Pounamu metaCASE tool. We have investigated thoroughly visual language metaphors suitable for specifying event handlers, addressed existing problems and applied examples to demonstrate the metaphors and prove concepts.

We have developed ViTABaL-WS using the tool abstraction metaphor to allow web services composition via higher level data and control flows and BPEL4WS code generation. Kaitiaki uses the event-query-filter-action metaphor to allow visual primitives composition and java event handler code generation. MaramaFormula uses a spreadsheet-like metaphor to construct metamodel formulae visually and generate java event handler code to be executed and realized at the model instance level.

The generalisation of domain models and their dependencies and development of a new visual language and environment to support general purpose event integration is the next, capstone step of the thesis research. This will result in the development of multiple visual language support for event-based integration.

4. METHODOLOGY

Our approach was based around the following methodological steps, repeated for each metaphor:

- We began with literature review of event systems in general and compared major visual event-handling techniques.
- We then focused on problems and issues in existing event-based approaches and tool support and selected a choice of metaphor as our target of research focus.
- We designed event handling support for the selected metaphor.
- We proved our concept for that metaphor by developing prototype systems and examples.
- We undertook evaluation of our visual language and environment.

Having examined several metaphors, our next step is to develop a high-level abstraction of these metaphors suitable for a range of application domains.

5. EVALUATIONS

We have carried out three evaluations against predefined criteria each of ViTABaL-WS and Kaitiaki to gauge their effectiveness: a Cognitive Dimensions [6] analysis, a domain specific requirements evaluation and an informal evaluation with experienced Pounamu users and some novice users.

Cognitive Dimensions provides a framework to assess usability characteristics of visual languages and their supporting tools. Key issues seen in both ViTABaL-WS and Kaitiaki include Abstraction Gradient, Closeness of Mapping, Error Proneness, Progressive Evaluation support, Viscosity, requirement for Hard mental operations and use of Secondary Notation.

With respect to domain specific requirements, ViTABaL-WS provides a generally effective environment for web service composition. The TA paradigm used as the compositional metaphor allows expression of complex web service interactions at a higher level of abstraction than languages like BPEL4WS and most existing BPEL4WS generation tools, which usually provide abstractions directly related to BPEL4WS constructs. Kaitiaki's EQFA metaphor captures event generation, Pounamu state querying, filtering and iteration over query results, and state changing actions to describe event handler specifications. The dataflow metaphor describes the composition of these event specification building blocks and seems to map well onto users' cognitive perception of the metaphor.

Feedback from user evaluations suggests that both ViTABaL-WS and Kaitiaki provide good support for end user specification of complex event handling but require enhancement in several areas, notably spreadsheet-type formula specification support. A multi-paradigm, general purpose event handling framework would support event integration specification of large complex systems that work across domains.

We plan to use the same evaluation techniques to evaluate MaramaFormula and our generalized visual language and environment. We also plan a more formal evaluation with novice users at a later stage to better gauge our visual specification approaches.

6. SUMMARY

We are investigating three exemplar visual event-driven system metaphors to specify event-handling support; they are Tool Abstraction in ViTABaL-WS, Event-Query-Filter-Action in Kaitiaki and Spreadsheet in MaramaFormula. We aim to generalise from the three exemplars and develop a generic high-level visual event handling metaphor and build a visual environment for specifying event-based system integration.

7. REFERENCES

[1] Costagliola, G., Deufemia, V., Ferrucci, F., Gravino, C.: The Use of the GXL Approach for Supporting Visual Language Specification and Interchanging. Proc HCC'02, Arlington, Virginia, 2002, pp131-138.

[2] Cypher, A. and Smith, D.C. KidSim: end user programming of simulations, Proc CHI'95, Denver, Colorado, pp. 27-34.

[3] D. Roberts, R. Johnson. Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks. In Proceedings of the Third Conference on Pattern Languages and Programming, Montecillio, Illinois, 1996.

[4] Fensel, D. and Bussler, C. The web service modeling framework WSMF, Electronic Commerce Research and Applications, vol. 1, no. 2, pp. 113--137, 2002.

[5] Foster, H., Uchitel, S., Magee, J. and Kramer, J. Model-based verification of web service compositions. In Proc. 18th IEEE ASE, 2003, Montreal, Canada.

[6] Green, T. R. G., Burnett, M. M., A Ko, J., Rothermel, K. J., Cook, C. R., and Schonfeld, J., Using the cognitive walkthrough to improve the design of a visual programming experiment, In Proc IEEE VL'2000, 172-179.

[7] Grundy, J.C. and Hosking, J.G. ViTABaL: A Visual Language Supporting Design by Tool Abstraction, In Proc IEEE VL'95, Germany, IEEE CS Press, pp. 53-60.

[8] Kelly, S., Lyytinen, K., and Rossi, M., Meta Edit+: A Fully configurable Multi-User and Multi-Tool CASE Environment, in Proceedings of CAiSE'96, LNCS 1080, Springer-Verlag, Crete, Greece, May 1996.

[9] Ledeczi A., Bakay A., Maroti M., Volgyesi P., Nordstrom G., Sprinkle J., Karsai G.: Composing Domain-Specific Design Environments, Computer, 44-51, Nov, 2001.

[10] Liu, N., Grundy, J.C. and Hosking, J.G. A Visual Language and Environment for Composing Web Services, In *Proc. 2005 IEEE/ACM Int. Conf. on Automated Software Engineering*, Long Beach CA, Nov 7-11 2005.

[11] Liu, N., Hosking, J.G. and Grundy, J.C. A Visual Language and Environment for Specifying Design Tool Event Handling, In *Proc. VL/HCC'2005*, Dallas, Sept 2005.

[12] Morch, A. Tailoring tools for system development, Journal of End User Computing, 10:2, 1998, pp. 22-29.

[13] Myers, B.A., The Amulet Environment: New Models for Effective User Interface Software Development, IEEE TSE, vol. 23, no. 6, 347-365, June 1997.

[14] Peltonen, J. Visual Scripting Language for UML-based CASE tools, In Proc. of ICSSEA 2000, volume 3, 2000.

[15] Srivastava, B., Koehler, J. Web Service Composition - Current Solutions and Open Problems, ICAPS Workshop on Planning for Web Services, 10 June 2003, Trento, Italy.

[16] The Rule Markup Initiative, http://www.ruleml.org

[17] Thone, S., Depke, R. and Engels, G. Process-oriented, flexible composition of web services with UML, Proc ER-Wkshp on Conceptual Modeling Approaches for e-Business, Tampere, Finland, LNCS, 2002.

[18] UML 2.0 OCL, http://www.omg.org/docs/ptc/03-10-14.pdf

[19] Zhu, N., Grundy, J.C. and Hosking, J.G., Pounamu: a meta-tool for multi-view visual language environment construction, Proc VL/HCC'04, pp. 254-256.