Experiences in Using Java on a Software Tool Integration Project

John Grundy[†], John Hosking[††] and Rick Mugridge[††]

[†]Department of Computer Science
University of Waikato
Private Bag 3105, Hamilton, New Zealand
jgrundy@cs.waikato.ac.nz

[††]Department of Computer Science
University of Auckland
Private Bag 92019, Auckland, New Zealand
{john, rick}@cs.waikato.ac.nz

Abstract

Building and integrating complex software development tools is a difficult task, but one which can result in great usability and productivity gains for software development. We describe our experiences in developing the Banquet set of integrated software development tools, initially using C++, and then Java as the tool interfacing and implementation and languages. We identify where, for us, the advantages of a Java-based approach lie.

1. Introduction

Many diverse software engineering methodologies and notations are available for use by students, practitioners and researchers. To make effective use of these, environments and CASE tools supporting them are proliferating. Many techniques address different or overlapping aspects of the software development life cycle, such as programming environments utilising hierarchical browsers, with notations in common with those of CASE tools. Configuration management and version control tools are useful for not only source code but also designs and documentation. Process modelling tools allow a developer to coordinate their work with multiple tools and among multiple developers.

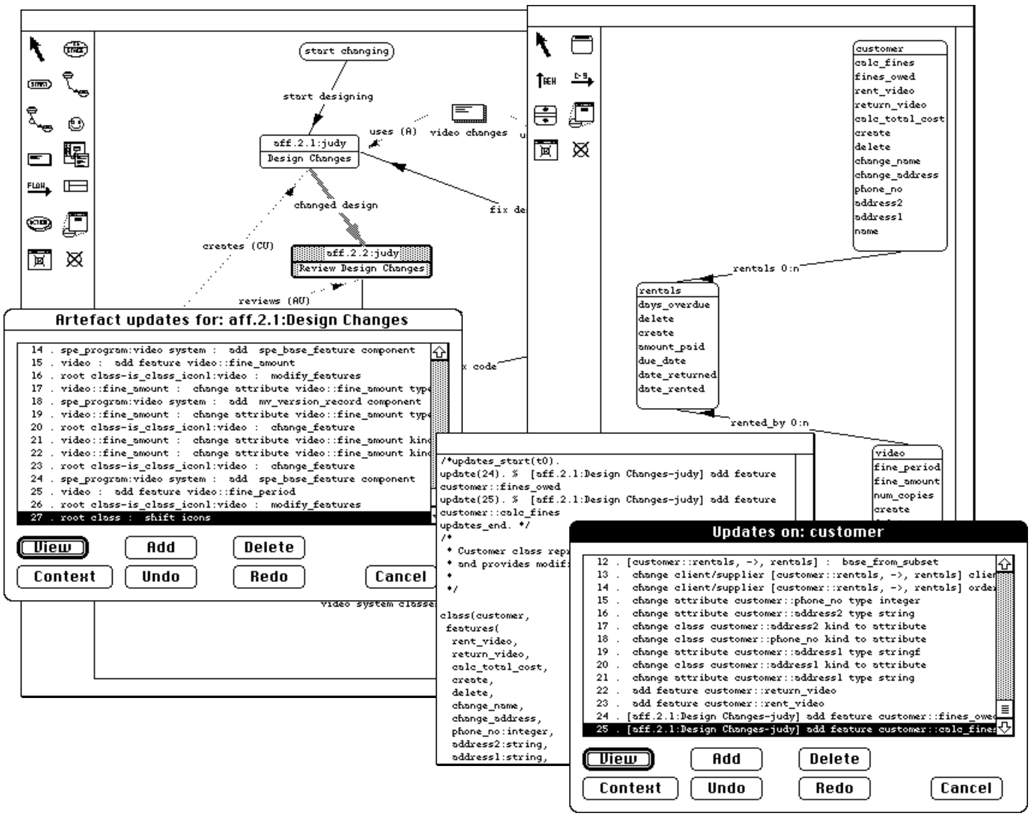
Integrated, coordinated use of these tools is needed for them to be most effective [32, 30]. Many researchers and commercial tool developers have developed integration mechanisms. Examples include FIELD [32, 33], DEC FUSE [19], HP Softbench [3], process modelling tools [1, 6], and collaborative work tools [35, 4]. In this paper, we report on the Banquet tool integration project [12], which aims to build upon our earlier tool development and integration work [7, 13].

Many of our earlier tools were implemented in an object-oriented Prolog [13, 14]. These tools suffered from development and modification overheads, performance problems and lack of sufficiently open architectures. Our initial approach to correcting these faults was to port the Banquet tools and their underlying architecture to C++ [12, 16]. Unfortunately, our C++ prototypes proved to be difficult to build and modify, the tool components developed with it were neither robust nor reusable, and they tended to lack third party tool integration support. We describe our C++ development experience, followed by our more recent experience in reimplementing our tool architecture and some tools using Java and the Java Beans componentware API [16]. Our Java prototype architecture and tools have proven to be easier to develop, more robust and reusable, and provide more open, extensible architectures than our C++ prototypes.

2. Banquet Project

The Banquet project [12] aims to develop software architectures, meta-CASE tools, software development tools and an integration framework to allow a wide range of software development tools to be effectively integrated. We want to effectively integrate tools developed using our own meta-CASE tools and also third-party tools.

We have produced many software development tools using MViews, the Prolog-based framework [14]. MViews provides abstractions for building multiple view, multi-user environments, with a novel consistency management mechanism.



Artefact updates for: aff.2.1:Design Changes

```
14 . spe_program:video system :  add  spe_base_feature component
15 . video :  add feature video::fine_amount
16 . root class-is_class_icon1:video :  modify_features
17 . video::fine_amount :  change attribute video::fine_amount type
18 . spe_program:video system :  add  mv_version_record component
19 . video::fine_amount :  change attribute video::fine_amount type
20 . root class-is_class_icon1:video :  change_feature
21 . video::fine_amount :  change attribute video::fine_amount kind
22 . video::fine_amount :  change attribute video::fine_amount kind
23 . root class-is_class_icon1:video :  change_feature
24 . spe_program:video system :  add  spe_base_feature component
25 . video :  add feature video::fine_period
26 . root class-is_class_icon1:video :  modify_features
27 . root class :  shift icons
```

```
/*updates_start(t0).
update(24). %  [aff.2.1:Design Changes-judy] add feature
customer::fines_owed
update(25). %  [aff.2.1:Design Changes-judy] add feature
customer::calc_fines
updates_end. */
/*
 * Customer class repr
 * and provides modifi
 *
 */

class(customer,
 features(
  rent_video,
  return_video,
  calc_total_cost,
  create,
  delete,
  change_name,
  change_address,
  phone_no:integer,
  address2:string,
  address1:string,
```

Updates on: customer

```
12 . [customer::rentals, ->, rentals] :  base_from_subset
13 .  change client/supplier [customer::rentals, ->, rentals] clien
14 .  change client/supplier [customer::rentals, ->, rentals] order
15 .  change attribute customer::phone_no type integer
16 .  change attribute customer::address2 type string
17 .  change class customer::address2 kind to attribute
18 .  change class customer::phone_no kind to attribute
19 .  change attribute customer::address1 type stringf
20 .  change class customer::address1 kind to attribute
21 .  change attribute customer::address1 type string
22 .  add feature customer::return_video
23 .  add feature customer::rent_video
24 . [aff.2.1:Design Changes-judy] add feature customer::fines_owed
25 . [aff.2.1:Design Changes-judy] add feature customer::calc_fines
```

Figure 1. Example of SPE-Serendipity integrated environment in use.

Systems built using MViews include:

-) [7], a multi-phase environment for OOA/D/P, including Object-Z specification views [8].
- CernoII [5], supporting visualisation and debugging of OO programs for SPE.
- MViewsDP [13], supporting dialog box design and Skin [21], supporting user interface components.
- MViewsER and MViewsNIAM, providing ER and NIAM modelling facilities [9, 38].
- ViTABaL [10], supporting visual tool abstraction based programming.
- C-SPE, providing collaborative extensions to SPE for multi-user OO development [11].

15].

SPE-Serendipity is one example of how we have integrated out tools together. We have integrated a number of our tools together loosely or tightly, while others are yet to be integrated in any way. For example, SPE and MViewsER have been fully integrated to produce OOEER, an environment which supports integrated OOA and EER modelling [9]. In an earlier paper we presented our aim of integrating all of the tools described above into, what appears to be from a user's perspective, one ISDE called Banquet [12]. Banquet will support all aspects of software development from planning and requirements formulation to program

- SPE (Snart Programming Environment
- Serendipity, supporting integrated software process modelling, enactment and work coordination [18].

As an example of the type of environment we aim to produce, and scale-up for large multi-developer software development projects, SPE-Serendipity is illustrated in Figure 1. This environment is an integration of the SPE object-oriented software development environment [7], and Serendipity, a software process modelling and enactment environment [18]. These environments have been tightly integrated so that, to a developer, SPE and Serendipity appear as one system [

implementation, visualisation and documentation. This paper reports on progress towards that aim.

MViews, the OO Prolog-based framework that we have previously used to develop our environments has proved to be an inadequate platform to continue our tool development and integration efforts. Being Prolog-based, MViews tools are difficult to integrate with third-party tools and have poor interactive performance, compared to systems built with compiled, imperative languages. MViews systems use an OO framework approach, where much useful tool functionality is abstracted into the framework classes. Nevertheless, tool builders must still write considerable amounts of code to build tools, as well as understand the ways in which the classes of the framework interact to provide the various capabilities of MViews.

3. C++ Approach

To address some of the problems with the Prolog-based MViews implementation, we initially adopted C++ as a platform with which to port (and redesign) the existing MViews framework. The principle advantages we anticipated in making this move were: a major increase in performance, allowing better scaling of applications; better access by others to our work, by using a more "standard" language; and portability. On the other hand, we could see the following risks in moving to C++:

- A lack of high level portable graphics support: Particularly critical to success was the definition of graphics support, to allow cross-platform implementation of applications. At the stage we were commencing, there was very little offering in the way of support for cross platform 2-D graphics at any reasonable level of abstraction. We saw production of a high-level graphics support library as being essential for permitting rapid development of MViews-based applications; the high level of graphics support provided by LPA MacProlog was one of the main attractions to this platform.
- A lack of garbage collection: In earlier work, we had already successfully ported much of the functionality of low–level MViews classes to a C++-like implementation, and hence felt that this would not be a high risk area. However lack of automatic garbage collection was seen as being a hindrance to successful porting of the code.
- A lack of dynamic code loading and manipulation capabilities: The OO Prolog we used (Snart) is extended with constraint processing capabilities which we have been experimenting with for providing graphics support [29]. We were very keen to develop these capabilities further, but had made use of Prolog's dynamic code manipulation capabilities to support their implementation. It was unclear initially whether C++ would provide sufficient flexibility for an efficient implementation. In addition, environments such as SPE, allowed code to be developed compiled, loaded and debugged, all from within the one environment. It was clear that this could prove difficult from within C++.

Having recognised these risks, we decided on the first step in an evolutionary approach to performing the port. This focussed initially on the graphics support, aiming to implement sufficient support to construct the MViews "display layer" of graphics, and the constraint capabilities (initially independently) with garbage collection capabilities to be explored as part of implementing the graphics support.

The approach taken to graphics support was to provide high level graphics abstractions, similar in style to those we were familiar with from LPA MacProlog. This style of abstraction had permitted us to rapidly construct the user interface aspects of the Prolog-based MViews environments. Low level platform-specific graphics support for implementing these abstractions was to be isolated as much as possible to allow ready porting to other platforms. The Metroworks PowerPlant application support libraries were used in the initial implementation [28]. While we feel this approach was architecturally correct, it readily became apparent that progress in implementation was going to be extremely slow, and that comprehensive memory management would prove to be a major component of the implementation as large numbers of objects would be created and destroyed to support the high level abstractions.

In parallel, constraint processing support was developed by extending C++ with constraint expression constructs. A preprocessor compiles these, adding triggers to assignments involving manipulations of attributes and object references included in the constraint expressions. Much of the dynamic aspects of the original constraint implementation needed to be compiled into these static triggers. While this approach appeared initially promising, the size of C++ meant a major implementation effort was required. A particular problem that was never resolved was the inability to accurately identify what items of data were object references and what were not (due to the weak typing system of C++, aliases through references (&), and the ability to perform pointer manipulations).

Experience from these initial steps suggested that the port was feasible, and any problems had workarounds (albeit somewhat inelegant in cases). However, the resources that would be expended in completing the port were far more than our group could support. Accordingly we abandoned C++ as a platform for further development.

4. Java Approach

Java was gaining momentum early in 1996, when it became clear that the problems of using C++ for Banquet were serious. Java seemed to provide an answer to most if not all of our difficulties with C++, as well as providing new opportunities (such as the expected persistence features that were later provided in Java1.1). Advantages we saw in Java over C++ were:

- The Java system provides a platform-independent GUI package (awt), adequate for our needs [22]. This is a lower-level graphics approach than that of

MacProlog, but in many ways provides more power (such as the use of LayoutManagers).

- Java provides automatic garbage collection. This makes it much easier to program in Java, but does leave us vulnerable to the performance hit of the garbage collector [22].
- Java permits later binding than C++. Given the name of any class (as a String), new objects of that class can be created. The source code of a Java class may be written to a file during program execution, the source compiled, and then objects of that new class created. We saw this as being useful for environments where tools can be integrated on the fly by developers. It proved to be extremely difficult to adequately mimic such a mechanism using C++.
- Java does not have the C++ notion of a reference (&) or pointer manipulation, so that it is not possible to construct an alias to a primitive type. So, as long as all attributes of a class are private, and can only be accessed through a method of the class, it is possible to determine statically all assignments to attributes which are referenced within a constraint, simplifying and making more robust constraint implementation.

However, we had some doubts about Java at that stage: Would it be adopted as a wide-spread language? Would implementations on the various platforms get beyond a beta-stage? Would the loss in efficiency over C++ be problematic? With these caveats, we commenced redevelopment in Java, and this was well underway towards the end of 1996. The Banquet work over this period concentrated on two main elements: JViews, the port of MViews to Java, and BuildByWire, a visual notation construction kit. BuildByWire is constructed as a complex Java bean that is used by tools based on JViews. Progress has developed much more rapidly than with C++. The development of JavaBeans and other features of Java 1.1 have confirmed that we made the right decision. As we see later, JavaBeans have had an important influence on the design of our architecture.

Java has been generally adopted much quicker than we expected. However, the quality of implementations have been poor and slow to improve. Speed of execution is poor compared to C++, even with the use of Just-In-Time (JIT) compilers. So far this has not been a problem, as speed of execution has been more than adequate.

5. BuildByWire

BuildByWire (BBW) is a constraint-based visual notation construction kit that was originally developed using our OO constraint Prolog [29]. BuildByWire allows a designer to create a visual notation (such as for Entity-Relationship diagrams) by composing visual elements together through direct manipulation. Visual elements are "wired" together using constraints (eg an OOA class box with class name, attributes and methods). Visual elements include simple ones such as rectangles and lines; collections, such as vertical lists of methods; and GUI elements, such as buttons and pop-up menus.

In porting BuildByWire to Java, two major aspects needed to be addressed: the implementation of constraints, used extensively to maintain connectedness between visual components as they are dragged, resized, etc; and the implementation of the visual components themselves. Constraints were initially hand-compiled as "trigger" classes which handle constraint propagation between the properties of the visual components, which were defined as JavaBeans. This made it trivial to allow arbitrary JavaBeans (such as Buttons) to be plugged into BuildByWire, with constraints defined between them and other visual elements. The use of beans for visual components also enabled the use of standard "property sheets" to define the properties of any bean in BuildByWire, including our standard visual components. BuildByWire has turned into a rather sophisticated bean builder, making use of constraints and allowing new beans to be composed of existing beans.

We had originally planned to extend the Java language with our constraints. However, experience with JavaBeans and the BeanBox [16] has convinced us otherwise. In a Bean builder, such as the BeanBox, an event of one bean may be connected to a method of another bean visually; the code for an adaptor class is generated automatically and immediately to make the connection. We plan to allow constraints to be defined in the same way, but between bean properties instead; trigger code can then be generated immediately in an adaptor class that manages constraint propagation.

The Java re-implementation of BuildByWire has been most successful, with adequate performance of the constraints for effective notation use. JavaBeans has had a strong influence on the implementation and evolving design of BuildByWire.

6. JViews

The initial development of JViews using Java has been very successful. JViews provides abstractions for:

- Modelling repository components, inter-component relationships and semantic constraints. Tool developers typically build these elements by specialising and composing JViews classes. JViews relationship classes explicitly model interclass aggregation and association relationships.
- Propagating changes between JViews components via relationship objects to maintain consistency. JViews supports mechanisms to "listen" to state changes which affect other components before and after they occur [16]. Java objects representing descriptions of the changes are propagated between

components and used to represent events and state changes. Java interfaces are used to allow any Java class implementing an appropriate interface to listen to components generating change descriptions.

- Various components supporting different kinds of repository and view component persistency, multiple user component updating, and change description presentation and storage.
- View-level components which correspond to BuildByWire GUI objects. These are used by developers to specify how BuildByWire and repository components are linked and how view edits modify repository component data. BuildByWire shapes implemented as Java Beans "plug into" view-level JViews components, and changes are propagated between these two layers using Java Beans-style event notification mechanisms.

Our Java implementation of JViews has a similar level of complexity to our MViews implementation in our OO Prolog. However, the Java implementation has a number of advantages, including better type checking by the Java compiler, a huge improvement in the performance and response time of editing tools, and the ability to readily interface JViews-based tools to third-party applications. The complexity of JViews compared to our C++ prototype is similar, but JViews has no memory management code as this is automatic in Java. JViews has also proved to be more robust than the C++ prototype, with components providing their own exception-handling mechanisms and automatic inter-component relationship management.

7. BBW Composer and JComposer

To build JViews-based tools with BBW-based editors, developers must still write a considerable amount of code to specialise, compose and instantiate BBW and JViews framework classes. This was a key problem with our OO Prolog-based tools; tool developers expended considerable effort to not only initially design iconic components, view-level data models and repository data models, along with appropriate semantic constraints at each level, but also to implement these designs using OO code.While developing BBW and JViews, we decided to minimise the amount of code developers would need to write to use these frameworks by first using these frameworks to build tools for composing BBW framework components into editors and GUI components, and for designing JViews repository and view data models and constraints. BBW Composer is a BBW/JViews tool which provides multiple, editable views for designing BBW iconic components [16].

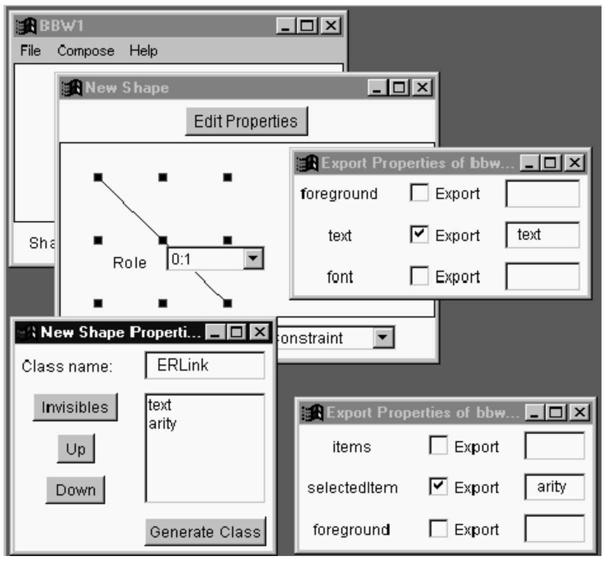


Figure 2. BBW Composition of ERLink shape.

Figure 2 shows an example of BBW Composer being used during the development of an ER modeller application. BBW shapes are defined by the tool developer drawing and composing shapes which have underlying correspondence to BBW framework shape, connector, constraint and pin classes. Appropriate Java Beans specialisation code is then generated by BBW Composer to construct and manipulate these iconic components for use in JViews (or other toolkit) environments.

JComposer is a BBW/JViews tool which allows developers to specify tool repository and view data models, along with semantic constraints and behaviour for these data models [16, 17]. Figure 3 shows JComposer being used to specify components for an ER modeller. Designers build up conceptual data models using the JComposer notation and then attach "listeners" to components to implement semantic constraints and editing behaviour of the components. JViews multi-user support components such as event propagation, storage and presentation of event objects, version control and synchronous view component editing can be utilised from JComposer. Developers can generate JViews class specialisations from within JComposer, and add arbitrary Java code to implement complex constraints or editing semantics, or compose appropriate constraints and event handling using a graphical language in JComposer.

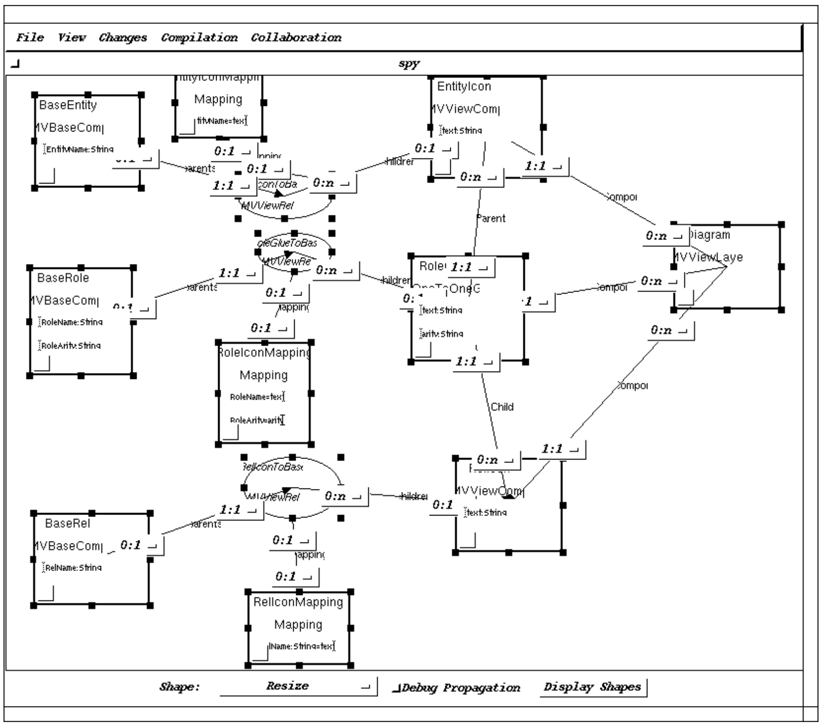


Figure 3. An example of JComposer being used to specify ER modeller components.

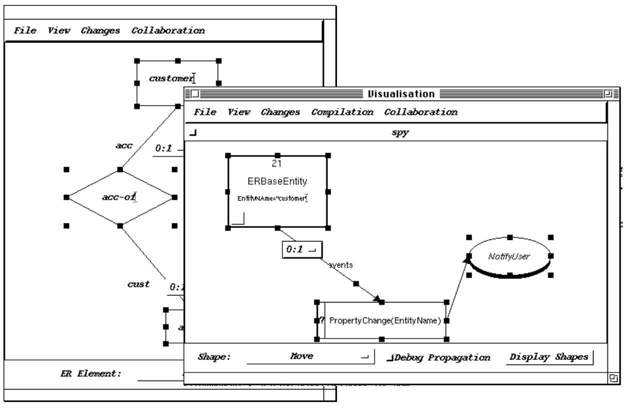


Figure 4. Run-time JComposer visualisation.

JViews environments can also make use of generic JComposer views for visualising and modifying running JViews component structures. This allows tool users to view running JViews component states, invoke component methods on the fly, and create new components. Component interrelationships can also be manipulated, and multiple views of running objects opened. Tool users can also create event handling components and attach these to running component visualisations, to modify the behaviour of their tools e.g. add notification of change mechanisms, integrate tools with other tools, etc. Figure 4 shows an example of the generated ER modeller components being visualised at run-time, with the user having added event handling components which notify them of EntityName property changes to the visualised entity.

8. Related Research

Many problems need to be overcome when attempting tool integration [37, 32]. Examples include: tools need to share data and keep partially duplicated data consistent ("data integration"); tools need to request other tools to perform operations for them or be notified of events in other tools ("control integration"); all tools should have consistent user interfaces ("presentation integration"); and tool use needs to be coordinated on large projects among multiple developers ("process integration").

PECAN [31], MELD [23], the Cornel Program Synthesiser [34] and Mjølner environments [27] use a large centralised database to store shared information with structure editors allowing modification of views of data. Dora environments [30] integrate multiple textual and graphical views of software development via a structure editing approach. CASE tools utilise code generation and reverse engineering to partially keep design and code consistent [39, 36]. While such approaches solve the data sharing problem in different ways,they often fail to satisfactorily support control and process intergation. FIELD environments [32, 33] and DEC FUSE [19] utilise selective broadcasting of events between Unix tools to achieve limited forms of control and user interface integration. Federated approaches use a database, such as PCTE, spread over several locations [2], and tend to better support both data and control integration. Unfortunately these approaches alone do not provide adequate process integration on large scale projects.

Tools supporting collaboration between multiple users include ConversationBuilder [24], wOrlds [25], and GroupKit [35]. Process-centred environments include CPCE [26], Oz [20], SPADE [1] and ProcessWEAVER [6]. The process-centred environments tend to support process integration well, as they support the coordinated use of multiple tools. But they often do not adequately support other aspects of tool integration, as it can be very difficult to tightly integrate third party tools in these environments.

All of these systems do not solve all tool integration problems, and generally lack adequate end-user configuration for tool integration. Our JViews and BBW frameworks, along with our BBW Composer and JComposer tools, provide solutions for data integration (via interconnected tool repositories using JViews components and relationships), control integration (via the propagation of JViews event objects between tool components), presentation integration (via BBW Java Beans specialisations), and process integration (via the visual JComposer event handling language and a process modelling environment we are porting to JViews). Our use of Java Beans specialisations means our environments are able to interoperate with other Java Beans, exchanging events and method invocations. Java Beans implementing Active X, InfoBus and CORBA interface bridges also allow JViews-based tools to interoperate with tools utilising these architectures.

9. Summary

We have described the Banquet vision for integrated software development environments. Our attempts to port our OO Prolog-based MViews architecture to C++ for the next generation of Banquet tools has proved to be a comparative failure. Tools were difficult and time-consuming to build, tool components tended to have only limited reusability and robustness, and interoperation support with third paty tools was limited. In contrast, our more recent experience using Java and JavaBeans to build a prototype Banquet infrastructure and exemplar tools has been very successful. We have ported all the MViews functionality to our new JViews framework, and added to this considerably, including persistence, multi-user and versioning support components. BuildByWire provides iconic editors for JViews tools and these tend to be much more user-customisable than those built with most existing user interface development toolkits. Our JComposer and BBW Composer environments, both implemented with the BuildByWire and JViews frameworks, provide tool developers with high-level, visual languages for constructing and integrating tool components. End-user configuration of JViews-based tools is supported by a graphical run-time visualisation and event handling language.

Future work with JViews, BBW and Banquet includes further development of our frameworks and meta-CASE environments so they can be used to develop more sophisticated new tools and integrate our tools with a wider range of third-party tools. Applications include software development tools, design tools, components for virtual reality modelling and visualisation, and agent-based systems utilising the JComposer event handling language to support composition and programming of agents. BBW Composer is being extended to support more sophisticated visual collections and to permit the use of click-through tools (which can also be composed visually). JComposer is being extended to support a wider range of reusable modelling and event handling components, distributed systems data storage/retrieval and event propagation components, and bridge compoents to support integration with third party software development tools. We are continuing to port various MViews-based environments to JViews in addition to developing new Banquet tools, using BBW Composer and JComposer.

References

[1] Bandinelli, S., Fuggetta, A., and Ghezzi, C., “Process model evolution in the SPADE environment,” *IEEE Transactions on Software Engineering*, vol. 19, no. 12, 1128-1144, December 1993.

[2] Bounab, M. and Godart, C., “A Federated Approach to Tool Integration,” in *Proceedings of CAiSE'95,* LNCS 932, Springer-Verlag, Finland, June 1995, pp. 269-282.

[3] Champine, M.A., “A visual user interface for the HP-UX and Domain operating systems,” *Hewlett-Packard Journal*, vol. 42, no. 1, 88-99, 1991.

[4] Ellis, C.A., Gibbs, S.J., and Rein, G.L., “Groupware: Some Issues and Experiences,” *Communications of the ACM*, vol. 34, no. 1, 38-58, January 1991.

[5] Fenwick, S., Hosking, J.G., and Mugridge, W.B., “Visual debugging of object-oriented systems,” in *Proceedings of TOOLS Pacific 94,* 1994.

[6] Fernström, C., “ProcessWEAVER: Adding process support to UNIX,” in *2nd International Conference on the Software Proces,* IEEE CS Press, Berlin, Germany, February 1993, pp. 12-26.

[7] Grundy, J.C., Hosking, J.G., Fenwick, S., and Mugridge, W.B., Connecting the pieces, Chapter 11 in *Visual Object-Oriented Programming*. Manning/Prentice-Hall, 1995.

[8] Grundy, J.C., and Hosking, J.G., “Support for Integrated Formal Software Development,” *Proceedings of the 1995 Asia-Pacific Conference on Software Engineering,* IEEE CS Press, Brisbane, Australia, Dec 6-9 1995, pp. 264-273.

[9] Grundy, J.C. and Venable, J.R., “Providing Integrated Support for Multiple Development Notations,” in *Proceedings of CAiSE'95,* LNCS 932, Springer-Verlag, Finland, June 1995, pp. 255-268.

[10] Grundy, J.C. and Hosking, J.G., “ViTABaL: A Visual Language Supporting Design By Tool Abstraction,” in *Proceedings of the 1995 IEEE Symposium on Visual Languages,* IEEE CS Press, Darmsdart, Germany, September 1995, pp. 53-60.

[11] Grundy, J.C., Mugridge, W.B., Hosking, J.G., and Amor, R., “Support for Collaborative, Integrated Software Development,” in *Proceeding of the 7th Conference on Software Engineering Environments,* IEEE CS Press, April 5-7 1995, pp. 84-94.

[12] Grundy, J.C., Hosking, J.G., and Mugridge, W.B., “Towards an environment supporting all aspects of software development,” *Proceedings of 1996 Conference on Software Engineering: Education and Practice,* IEEE CS Press, Dunedin, New Zealand, January 1996.

[13] Grundy, J.C., Hosking, J.G., and Mugridge, W.B., “Supporting flexible consistency management via discrete change description propagation,” *Software - Practice & Experience*, vol. 26, no. 9, 1053-1083, September 1996.

[14] Grundy, J.C. and Hosking, J.G., “Constructing Integrated Software Development Environments with MViews,” *International Journal of Applied Software Technology*, vol. 2, no. 3-4, 133-160, 1996.

[15] Grundy, J.C., Hosking, J.G., and Mugridge, W.B., “Low-level and high-level CSCW in the Serendipity process modelling environment,” in *Proceedings of OZCHI'96,* IEEE CS Press, Hamilton, New Zealand, Nov 24-27 1996.

[16] Grundy, J.C., Mugridge, W.B., and Hosking, J.G., “A Java-based toolkit for the construction of multi-view editing systems,” in *Proceedings of the Second Component Users Conference,* Munich, Germany, July 14-18 1997.

[17] Grundy, J.C., Mugridge, W.B., and Hosking, J.C., “Supporting end-user specification of work coordination in workflow systems,” in *2nd International Workshop on End-user Development,* University of Hull Press, UK, Barcelona, Spain, June 15-16 1997.

[18] Grundy, J.C. and Hosking, J.G., “Serendipity: integrated environment support for process modelling, enactment and work coordination,” *Automated Software Engineering*, vol. 5, no. 1, January 1998 (in press).

[19] Hart, R.O. and Lupton, G., “DECFUSE: Building a graphical software development environment from Unix tools,” *Digital Tech Journal*, vol. 7, no. 2, 5-19, 1995.

[20] Heineman, G.T. and Kaiser, G.E., “An Architecture for Integrating Concurrency Control into Environment Frameworks,” in *Proceedings of the 17th International Conference on Software Engineering,* IEEE CS Press, Seattle, Washington, April 1995, pp. 305-313.

[21] Hosking, J.G., Fenwick, S., Mugridge, W.B., and Grundy, J.C., “Cover yourself with Skin,” in *Proceedings of OZCHI'95,* Wollongong, Australia, Nov1995, pp. 101-106.

[22] JavaSoft Inc., “The Java Language: An overview,” White Paper, 1997.

[23] Kaiser, G.E. and Garlan, D., “ Melding Software Systems from Reusable Blocks,” *IEEE Software*, vol. 4, no. 4, 17-24, July 1987.

[24] Kaplan, S.M., Tolone, W.J., Carroll, A.M., Bogia, D.P., and Bignoli, C., “Supporting Collaborative Software Development with ConversationBuilder,” in *Proceedings of the 1992 ACM Symposium on Software Development Environments,* ACM Press, 1992, pp. 11-20.

[25] Kaplan, S.M., Fitzpatrick, G., Mansfield, T., and Tolone, W.J., “Shooting into Orbit,” in *Proceedings of Oz-CSCW'96,* DSTC Technical Workshop Series, University of Queensland, Australia, August 1996, pp. 38-48.

[26] Lonchamp, J., “CPCE: A Kernel for Building Flexible Collaborative Process-Centred Environments,” In *Proceedings of the 7th Conference on Software Engineering Environments,* IEEE CS Press, Netherlands, April 1995, pp. 95-105.

[27] Magnusson, B., Asklund, U., and Minör, S., “Fine-grained Revision Control for Collaborative Software Development,” in *Proceedings of the1993 ACM SIGSOFT Conference on Foundations of Software Engineering,* Los Angeles CA, December 1993, pp. 7-10.

[28] Metrowerks Inc., "Inside Powerplant for CW11", 1996.

[29] Mugridge, W.B., Grundy, J.C., Hosking, J.G., and Amor, R., *Snart94 Reference/User Manual*, Department`of Computer Science, University of Auckland, 1995.

[30] Ratcliffe, M., Wang, C., Gautier, R.J., and Whittle, B.R., “Dora - a structure oriented environment generator,”*IEE Software Engineering Journal*, vol. 7, no. 3, 184-190, 1992.

[31] Reiss, S.P., “PECAN: Program Development Systems that Support Multiple Views,” *IEEE Transactions on Software Engineering*, vol. 11, no. 3, 276-285, 1985.

[32] Reiss, S.P., “Connecting Tools Using Message Passing in the Field Environment,” *IEEE Software*, vol. 7, no. 7, 57-66, July 1990.

[33] Reiss, S.P., “ Interacting with the Field environment,” *Software practice and Experience*, vol. 20, no. S1, S1/89-S1/115, June 1990.

[34] Reps, T. and Teitelbaum, T., “Language Processing in Program Editors,” *COMPUTER*, vol. 20, no. 11, 29-40, November 1987.

[35] Roseman, M. and Greenberg, S., “Building Real Time Groupware with GroupKit, A Groupware Toolkit,” *ACM Transactions on Computer-Human Interaction*, vol. 3, no. 1, 1-37, March 1996.

[36] *TurboCASE Reference Manual*, StructSoft`Inc, 5416 156th Ave. S.E. Bellevue, WA, 1992.

[37] Thomas, I. and Nejmeh, B., “Definitions of tool integration for environments,” *IEEE Software*, vol. 9, no. 3, 29-35, March 1992.

[38] Venable, J.R. and Grundy, J.C., “Integrating and Supporting Entity Relationship and Object Role Models,” in *Proceedings of the 14th Object-Oriented and Entity Relationship Modelling Conference,* LNCS 1021, Springer-Verlag, Gold Coast, Australia, 1995.

[39] Wasserman, A.I. and Pircher, P.A., “A Graphical, Extensible, Integrated Environment for Software Development,” *SIGPLAN Notices*, vol. 22, no. 1, 131-142, January 1987.