

# Technology Briefing Report

## RAD (Rapid Application Development)

Identifier	D5.1b RAD
Type	Deliverable
Activity	WP5.1b
Date	31 <sup>st</sup> July 1996
Status	Draft
Partner	Lancaster
Availability	Public

# 1. Overview

RAD (Rapid Application Development) is the union of tools, techniques and methods which enable organizations to increase productivity during software development and evolution. Increases in productivity yield a shorter time-to-market and reduced software costs.

RAD technology often allows individuals without a formal background in Computing to contribute to development. Sophisticated end users can develop applications [Linthicum 95]. Other users can participate in extracting system requirements and assist with testing. It is important to note however, that effective RAD requires not only technical support, but an appropriate process model.

Several tools have emerged to support RAD. *Visual programming* is the development of programs through visual, rather than textual means. There is currently much interest in the notion of building applications from reusable components. Reuse is not a new idea, but effective reuse has yet to be experienced. *Componentware* appears a promising RAD technology for application composition from off-the-shelf components. *Fourth generation languages*, since their advent in the mid 1980s have evolved to offer extensive RAD support, particularly for data-oriented applications. *Frameworks* promote RAD by providing complex mechanisms such as Smalltalk's Model View Controller user interface model, upon which applications can be built. For specialized application domains, RAD can be facilitated by *application generators*. Finally, *visual modeling tools* allow the construction and execution of models during design. The identified classes of RAD tools are not mutually exclusive, and are elaborated in the following sub-sections.

## 1.1 Visual programming

Visual programming, as exemplified by Visual Basic [Gillmor 95, Hettler 96] simplifies event-driven GUI construction by dragging and dropping desktop metaphor objects to create screen forms. This level of visual programming still requires traditional programming to associate behavior with each object (button, window etc.). Visual programmers are relieved of issues such as event and window management, and are thus able to focus on the problem rather than the mechanics of the solution.

Another form of visual programming attempts to remove completely the need for traditional coding [Udell 94]. This approach is based on linking visual expressions - high level constructs such as menus, windows and buttons are arranged on screen and their inputs and outputs linked. In practice however, coding in a 3GL like Pascal or C is still required to introduce necessary constructs.

Many visual programming environments extend their use to non-Computing individuals. It is argued however that performance and maintainability of visually developed applications are significantly improved when written by developers with a Computing background [Snell 95]. All visual programming environments allow clients to sit down with developers and prototype user interfaces. Visual programming languages generally run on a virtual machine and are interpreted. In addition to GUI building, visual programming languages often incorporate reusable components and database access.

## 1.2 Componentware

High expectations were made of object-oriented technology for object-reuse. Class hierarchies and libraries emerged, but they have not made the great impact in reuse which was promised. [Udell 94] explains that the primary reason for this is that object-oriented programming languages lack the means to package and distribute objects in binary form. Binary code generated from multiple C++ compilers for example are rarely compatible.

Componentware is a new technology which promotes true off-the-shelf component reuse, with the benefits of shorter development time, greater reliability and more efficient use of resources [Joch 96]. A component does not yet have a universally accepted definition, but the vision of componentware promotes components with the following properties:

- Binary code wrapped by an interface specified in a common language. Components are thus programming language independent.
- Platform independence.
- Distributable.

The capability of building applications from Lego-like assemblies appears attainable with componentware. [Orfali 95] suggests that client/server applications can be assembled within days using componentware. It is often expensive to specialize applications for particular markets [Udell 94] - another issue which componentware aims to address.

### **1.3 Fourth generation languages**

A fourth generation language (4GL) can be loosely defined as non-procedural and user-oriented [Misra 88]. A current 4GL is typified by [Dobson 95, Cote 95, Hettler 96, Stearns 95, Uniface 96]:

- *A visual programming environment.* GUIs, reports and database schemas can be developed visually. The visual environment is usually built on top of an object-oriented programming language. Code generated from visual tools can be edited, and these changes are recognized by the visual tools. A class library is typically supported, which supports reuse of common objects, such as menus, windows and buttons.
- *Componentware integration.* Third-party components can often be integrated into a 4GL application. Many 4GLs also allow components to be produced and used by other componentware.
- *Database access mechanisms.* 4GLs provide mechanisms to a variety of local or remote database servers.
- *Cooperative application development support.* Where teams are involved with application development, support is given for version management and checking-in/out objects to protect them from concurrent editing.

The roots of 4GLs stem from prototyping. Prototyping has proven successful for extracting users' requirements and demonstrating the look and feel of an application [Linthicum 95]. Historically, developers have often thrown away the prototype to develop the final product in a 3GL. 4GLs extend prototyping tools to allow a prototype to be transformed into a fully functional application.

4GLs are most suited to data-oriented applications, in particular database front-ends and form-driven data management [Linthicum 95]. In a client/server architecture, the client provides the front-end, and brokers requests to the server. Client applications with GUIs can be developed quickly and bound to appropriate database servers.

### **1.4 Frameworks**

Many RAD technologies are accompanied with claims of "zero lines of code" required to develop an application. This is rarely true of any RAD technology, and frameworks are without exception. Frameworks facilitate rapid application development by extending and compiling existing source code with new application code. Frameworks are thus language dependent. *Application frameworks* provide client domain functionality like GUI support and text/graphics editors. Other frameworks exist for *problem domains*, such as 2D or 3D graphics. *Support frameworks* cater for system level issues like application interoperability, with a CORBA compliant implementation for example.

Modern frameworks aim to be extensible, portable and platform independent [Myers 95]. Extensibility is employed by writing frameworks in an object-oriented language, allowing users to extend existing classes where appropriate. Middleware is used to shield users from platform dependencies.

### **1.5 Application generators**

An application generator is a tool which allows application builders to generate an application from a specification which essentially comprises a parameter list [Cleveland 88]. Application generators can increase development productivity by the ease of which they allow prototyping and testing of specifications. For a family of related applications where a consistent look-and-feel for each interface is required, application generators can ensure such consistency.

Application generators are however associated with significant problems. In particular, application generators are difficult and expensive to build, and are severely limited to a narrow range of closely related applications. Code generated by application generators is often very difficult to read. A generator is not able for example, to generate meaningful identifiers in a 3GL. It can be very difficult to re-implement parts of a generated application to achieve necessary performance gains.

## 1.6 Design Tools

The classes of applications which can be produced from 4GLs and visual programming languages do not include real-time and math-intensive systems [Pountain 96]. Time constraints and the requirements for reliability and fault-tolerance demanded by many real-time systems are difficult to express visually. Math applications typically contain problem specific algorithms which are also not suited to visual specification. However, such application domains can benefit from RAD tools which improve productivity during design.

## 2. Role in evolution process

At this point, we define evolution terminology<sup>1</sup> prior to describing the relationship between evolution and RAD.

- *Reverse engineering.* Any activity that improves one's understanding of software [Arnold 93].
- *Software reengineering.* Changes to the underlying technology of a system without affecting the overall function [Yourdon 89]. Some reverse engineering activity may precede reengineering.

*Software evolution.* The accommodation of perfective, corrective and adaptive maintenance. Software evolution may involve some reengineering activity. Maintenance activities are defined by [Sommerville 96]:

- *Perfective maintenance.* Changes which improve the system in some way without changing its functionality.
- *Adaptive maintenance.* Maintenance which is required because of changes in the environment of a program.
- *Corrective maintenance.* The correction of previously undiscovered system errors.

RAD technology is a means to rapid development and subsequent evolution of applications. Some RAD technology assists with understanding requirements (notably tools with visual programming facilities). Frameworks and componentware provide for rapid implementation. RAD design tools allow developers to design and verify their designs relatively quickly to traditional approaches. RAD is not a reverse engineering tool, but can be used as a new underlying technology during software reengineering.

## 3. Relevance to RENAISSANCE

The specific objectives of RENAISSANCE to which RAD is relevant are:

1. Support application evolution from centralized to distributed client/server architectures.
2. Support evolution through the reuse of sub-systems recycled from existing systems.
3. Provide a method for project managers to assess the costs, risks and benefits of evolution options.

The 4GL and componentware technologies provide mechanisms to realize the client/server architectures referred to in objective 1. The vast majority of current 4GLs provide the means to implement clients with GUIs which access remote data servers. Componentware can be used to compose distributed client/server systems. One motivation for migrating from a centralized structure to a client/server architecture is to exploit the ease-of-use of a client with a GUI. Legacy data may be maintained in a server, with the possibility of reorganizing the data in the future. Incremental migration is thus supported.

Objective 2 is satisfied by componentware. [Grehan 95] reports on Micro Focus' Visual Object COBOL - an integrated support environment for object-oriented COBOL. Visual Object COBOL allows existing COBOL programs to be packaged into Microsoft's OLE (Object Linking and Embedding) objects. An OLE object is essentially a component which is built on Microsoft's COM (Component Object Model) infrastructure. The ability to componentize a legacy program means that the business logic encapsulated in the legacy code survives evolution. Further, selecting particular

---

<sup>1</sup> These definitions will contribute to the glossary of terminology for the RENAISSANCE project, which is being developed by Lancaster. The glossary will be available for comment by 31<sup>st</sup> August 1996.

programs of a legacy application to transform into components allows for the incremental migration from a centralized system to a distributed client/server architecture.

The technical RAD technologies are supported by a development standard - Dynamic Systems Development Method (DSDM) [Millington 95]. The standard is the product of a UK consortium of RAD developers and addresses the issues of project management, personnel, tools and techniques, and quality assurance. DSDM may prove to be a useful input to objective 3.

## 4. Available support

Perhaps the greatest success of visual programming to date is Visual Basic. Release 4.0 [Hettler 96] extends its earlier ability to rapidly construct GUI applications with componentware integration and database support. Visual Basic 4.0 applications can exchange data and control with local and remote OLE objects. In addition, a Visual Basic 4.0 program can be packaged as an OLE server to be used by other OLE-compliant objects. Release 4.0 is ODBC (Open Database Connectivity) compliant. ODBC is a middleware standard for accessing different database systems. An application can submit statements to ODBC using the ODBC flavor of SQL. ODBC then translates these to whatever flavor the database understands. Visual Basic 4.0 has evolved to be a serious candidate for data-oriented and distributed client/server development.

Visual programming based on manipulating visual expressions (introduced in Section 1.1) is supported by Prograph's Prograph, Novell's Visual AppBuilder, and Digitalk's Parts [Udell 94].

For componentware to provide true off-the-shelf components from which applications can be composed very rapidly, an infrastructure is required which governs component naming, distribution, and interaction. The infrastructure is essentially an open bus which components may be plugged into. The goal is to have end users and developers enjoy the same levels of application interoperability that are available to hardware manufacturers [Adler 95].

Dominant infrastructures include Microsoft's COM-based OLE model and the CORBA (Common Object Request Broker Architecture) standard. OLE has originated from desktop interoperability and compound documents (for example, the ability to embed a Microsoft Excel document in a Microsoft Word document). The CORBA standard was explicitly designed to manage distributed component-based applications and offers greater operating system and programming language freedoms over OLE [Halfhill 96]. OLE is available today with a host of OLE-compliant applications. Initial implementations of CORBA have emerged (for example Iona's ORBIX), but generally conform only to CORBA 1.2. CORBA 2.0 defines how different implementations of CORBA may communicate with each other [Orfali 95]. These technologies are the subject of the Distributed Object Technology report, so are not discussed further by this report.

The 4GL functionality described in Section 1.3 is very well supported by Borland's dBASE V [Carls 95] and Delphi [Cote 95, Hettler 96, Stearns 95], Microsoft's Visual Fox Pro [Linthicum 95a], Gupta's SQL Windows [Hettler 94, Stearns 95], PowerSoft's PowerBuilder [Stearns 95], and Compuware's Uniface [Uniface 96]. All packages support distributed client/server development with a strong visual programming element. The first three products are dependent on PC hardware running a Windows operating system. They are however OLE-compatible and thus able to use OLE servers and package their programs as OLE components. Uniface is largely platform independent.

The 4GLs allow applications to be prototyped with a local server. Working prototypes can be transformed to distributed client/server systems by replacing the local database server with a remote server. Many of the 4GLs support the ODBC standard. In addition, Delphi and Uniface support remote connectivity to established SQL databases such as Oracle, Sybase, Microsoft SQL server and Informix. Uniface extends this with DB2, DB2/2, DB2/6000 and CA-Ingress. The 4GL tools typically allow interaction with, and manipulation of data from several different sources.

Uniface is unique with its support for distribution. A Uniface application can be transparently deployed using multiple execution architectures: host-centric, client/server, or three-tier for true distributed processing. The deployment manager also handles seamless integration with TP monitors. SQL Windows, Delphi and Uniface also allow binary application executables to be generated; the development environment is not required by application users, and applications' performance is not compromised by an interpreter.

We are aware of little support for frameworks and application generators. Taligent is optimistic about its recently developed object-oriented application environment named CommonPoint

[Myers 95]. CommonPoint is a collection of 100 frameworks with nearly 2000 classes. Taligent expect that with a learning curve of between 4 and 5 months, applications can be rapidly developed with a suitable framework. Beyond research tools, there is a lack of application generator support.

Visual modeling tools include Visual Solutions' VisSim [Varhol 94] and an SDL-based visual toolset [Pountain 96]. SDL is an established formal language for modeling communicating finite state machines. SDL-based tools allow developers to design and analyze real-time systems. Such tools help expose design errors early on in the design process, where they are less expensive to correct than during testing.

## 5. Maturity assessment

Table 1 summarizes our impression of the level of maturity which each RAD technology has reached. Visual programming languages have evolved over the last 6-7 years from simple GUI generators to professional programming environments with capabilities to produce distributed client/server applications which are able to interact with popular databases. A wealth of third party components (VBXes and OCXes) have appeared for use with Visual Basic which extend the power of Visual Basic applications.

Technology	Level of maturity		
	Low	Medium	High
Visual programming			✓
Componentware		✓	
Fourth generation languages			✓
Frameworks		✓	
Application generators	✓		
Design tools		✓	

**Table 1 RAD technology maturity assessment**

Componentware is a relatively new technology with much potential for RAD. There are however a number of issues which have to be resolved before componentware is sufficiently mature for use in distributed application development. Microsoft's work on componentware for application interoperability on the desktop is now relatively stable. Microsoft's latest applications are OLE-compatible, as are many third party applications which run on PCs with Windows operating systems. Support for distributed component-based applications is coming from Microsoft with Network OLE, but the technology appears platform dependent. CORBA 2.0 compliant implementations are also awaited. Further work is in progress to define interoperability between OLE and CORBA.

In addition to the remaining technical hurdles for componentware, business considerations regarding licensing, packaging and distribution of components need to be addressed before a component market can thrive [Udell 94]. Building components and composing applications from components require different skills.

4GLs are perhaps the most mature of all the RAD technologies discussed in this report. They have evolved over the last decade from interpreted text-based centralized application development environments to visual event-driven GUI distributed client/server environments. Today's PC-based 4GLs are generally OLE compatible and thus able to interact with components. Enterprise 4GLs such as Delphi and Uniface also provide support for teams of developers working on a particular application.

Frameworks are built on established object-oriented technology and so benefit from the properties of well-engineered software. Frameworks are not as powerful RAD tools as componentware and 4GLs, but once over the relatively long learning curve, developers should see significant productivity improvements through using appropriate frameworks. Application generators are the most immature of the RAD technologies reviewed. Application generator support has failed to emerge because of the problems associated with their development and the limited range of applications which can be generated from a single generator.

## 6. Inapplicability

Fundamentally, RAD technology should only be employed on a project if it is suitable [Linthicum 95, Millington 95, Reilly 95]. Suitability is determined by technical, organizational, and contractual factors. A 4GL for example, would not be a suitable development technology for an application with stringent time constraints. Organizational factors include staff skills and experience with particular technologies. Contracts may specify that an application should be coded in a particular 3GL.

Visual programming languages and 4GLs are primarily intended for development of GUI data-oriented applications. Classes of application which these RAD technologies are unsuitable for include math-intensive, real-time, safety-critical, fault-tolerant and parallel applications. Componentware, frameworks and design tools may be specifically developed for the rapid development of applications from these classes.

## 7. Future development

Componentware will develop further to tackle the technical and business issues identified in Section 5. Componentware should evolve to embrace an open systems architecture which will facilitate application composition from components across platform, network and language boundaries. The next generation of client/server systems are expected to be three-tier [Hettler 96]. A three-tier application adds a third component (the application server) in between the client and server of a two-tier system. The new application server maintains some data and behavior which reflects business logic. The advantage of a three-tier system over a two-tier system is that when business logic changes, the change can be localized to the application server. In contrast, a two-tier system requires changes to many clients.

The implication for 4GLs in supporting a three-tier architecture is that they should provide for partitioning an application across a distributed architecture. Of the 4GLs reviewed, only Uniface offers explicit partitioning support; SQL Windows and PowerBuilder are expected to follow Uniface with new releases.

## 8. Other comments

The technical RAD technologies discussed in this report should be managed by an appropriate process model. In Section 3, we introduced a standard for RAD development, DSDM. The standard identifies three critical success factors for RAD projects: easy access to end users, a stable and skilled development team, and a commercial application. The DSDM standard provides advice on how to manage RAD technology and exploit its potential for improving productivity.

Visual programming, 4GLs and componentware are key RAD technologies which are also well-suited to the business domain - a core property of the RENAISSANCE project. Each technology offers great potential for improving productivity. Visual programming languages and 4GLs are mature, and the level of interest in componentware promises to evolve the technology into a useable form of RAD.

## 9. References

- |                 |  |
|-----------------|--|
| [Adler 95]      | R. M. Adler. <i>Emerging Standards for Component Software</i> . Computer March, 68-77. 1995.   |
| [Arnold 93]     | R. S. Arnold. <i>A Road Map Guide to Software Reengineering Technology</i> . Software Reengineering, IEEE Computer Society Press, 3-22. 1993.  |
| [Carls 95]      | J. Carls. <i>dBASE Does Windows</i> . Byte January, 193-196. 1995.   |
| [Cleaveland 88] | J. C. Cleaveland. <i>Building Application Generators</i> . IEEE Software 5(4), 25-33. 1988.  |
| [Dobson 95]     | R. Dobson. <i>RADical Databases</i> . Byte July, 24-25. 1995.  |
| [Gillmor 95]    | S. Gillmor. <i>Storming the Enterprise</i> . Byte November, 219-222. 1995.   |
| [Grehan 95]     | R. Grehan. <i>New Micro Focus Tool Converts COBOL into Components</i> . Byte December, URL <a href="http://www.byte.com/art/9512/sec4/art10.htm">http://www.byte.com/art/9512/sec4/art10.htm</a> . 1995. |

- [Halfhill 96] T. R. Halfhill and S. Salamone. *Components Everywhere*. Byte January, 97-104. 1996.
- [Hettler 94] M. Hettler and S. Higgs. *SQL Front Ends for Windows*. Byte October, URL <http://www.byte.com/art/9410/sec12/art1.htm>. 1994.
- [Hettler 96] M. Hettler. *New Leaders of the Client/Server Migration*. Byte June, 124-131. 1996.
- [Joch 96] A. Joch. *Killer Components*. Byte January, 81. 1996.
- [Linthicum 95] D. S. Linthicum. *The End of Programming*. Byte August, 69-72. 1995.
- [Linthicum 95a] D. S. Linthicum. *Foxy Move to Client/Server*. Byte August, 117-120. 1995.
- [Misra 88] M. K. Misra and P. J. Jalics. *Third-Generation versus Fourth-Generation Software Development*. IEEE Software 5(4), 8-14. 1988.
- [Millington 95] D. Millington and J. Stapleton. *Developing a RAD Standard*. IEEE Software 12(5), 54-55. 1995.
- [Myers 95] W. Myers. *Taligent's CommonPoint: The Promise of Objects*. Computer March, 78-83. 1995.
- [Orfali 95] R. Orfali and D. Harkey. *Client/Server with Distributed Objects*. Byte April, 151-162. 1995.
- [Pountain 96] D. Pountain. *RAD for Real-Time Applications*. Byte May, URL <http://www.byte.com/art/9605/sec18/art4.htm>. 1996.
- [Reilly 95] J. P. Reilly and E. Carmel. *Does RAD Live Up to the Hype?* IEEE Software 12(5), 24-26. 1995.
- [Sommerville 96] I. Sommerville. *Software Engineering*, Addison-Wesley. 1996.
- [Stearns 95] T. Stearns. *Comparison: Visual Programming Tools for Database Front-Ends*. LAN Times Online URL <http://www.lantimes.com/lantimes/archive/506b081a.html>. 1995.
- [Snell 95] M. Snell. *A New Visual Attitude*. LAN Times Online May, URL <http://www.lantimes.com/lantimes/archive/505a047a.html>. 1995.
- [Udell 94] J. Udell. *Componentware*. Byte May, URL <http://www.byte.com/art/9504/sec5/art1.htm>. 1994.
- [Uniface 96] *An Introduction to Uniface*. URL <http://www.compware.com/products/uniface/unintro.htm>. 1996.
- [Varhol 94] P. D. Varhol. *Visual Programming's Many Faces*. Byte July, URL <http://www.byte.com/art/9407/sec12/art2.htm>. 1996.
- [Yourdon 89] E. Yourdon. *RE-3 Reengineering, Restructuring, Reverse Engineering*. American Programmer 2(4), 3-10. 1989.