Diomidis Spinellis

The Loyal Opposition

The Computer's New Clothes

Software organizations began introducing the client–server system architecture in the late 1980s because CSS offered a technology-driven application design that genuinely addresses corporate and user needs. Widely promoted by software and hardware vendors who sought short-term advantages in a competitive market, CSS has always been a haphazard match of disparate, ad hoc technologies that lack the guidance of a unifying vision, purpose, and theory.

Software professionals seldom admit that the CSS architecture failed inevitably, or that technologies that claim to improve it are only addressing this failure. Our industry would be better served if we honestly appraised the situation, evaluated the lessons learned, and undertook a fresh start.

Summon the Mainframe's Tailor

Client–server systems resulted from the introduction of networked PCs in the 1980s. Companies layered this new desktop infrastructure atop existing mainframe installations controlled by highly centralized Management Information System departments. PC purchases were initially motivated by the lure of user-friendly personal productivity software and freedom from MIS's control. Powerful and responsive spreadsheets and word processors provided improved data analysis and report generation. Software systems such as Symphony and Windows even attempted to assimilate individual applications into an integrated working environment. Low-priced PC hardware and software allowed decentralized purchasing decisions; the vision of a PC on every desktop became a reality.

Yet MIS retained control of the mission-critical applications. These mature, legacy software systems, running in a stable operating environment with scheduled backup and maintenance procedures, ensured the longevity and reliability of the corporation's most vital information. When users needed access to MIS data from the desktop PC, a terminal

0740-7459/98/$10.00 © 1998

emulation package opened a window on the mainframe host.

Seeking a new look…

Three major forces drove the move toward CSS.

♦ The costs of desktop PC hardware and software acquisition and maintenance, and of software development, often proved magnitudes lower than traditional MIS computer spending. The move toward PC-based computing seemed to make financial sense.

♦ Users spoiled by increasingly user-friendly desktop applications began demanding that mainframe applications offer similarly friendly interfaces.

♦ The need to integrate access to mainframe data from PCs increased: users wanted to use PC spreadsheets to analyze and graph mainframe data, and to integrate legacy system reports into word-processed documents. Business process reengineering efforts typically supported the increased productivity, flexibility, and improved customer service that such an environment fostered.

…tailored from new fabrics

In parallel to these driving forces, three enabling technologies made the CSS transition possible.

♦ Standalone PCs became networked, initially to gain access to large storage devices and printers. As networking technology matured, network stacks like TCP/IP and SNA over megabit LANs became available on both mainframes and PCs. This development allowed the efficient interconnection of two disparate worlds.

♦ Relational database management systems such as Oracle and the Microsoft SQL Server—accessible through a structured query language interface and running on Unix machines or PC servers—provided an open alternative to proprietary mainframe databases.

♦ Graphical user interfaces such as Microsoft Windows provided a standardized, user-friendly deployment environment. Rapid application development tools and fourth-generation languages eased application development for those platforms.

Look! The Client's Naked, Too

The truth is that the introduction of CSS architectures has been a dismal failure. Several promises that drove the move toward CSS have not been fulfilled (Peter Duchessi and InduShobha Chengalur-Smith, "Client/Server Benefits, Problems, Best Practices," *Comm. ACM*, May 1998, pp. 87-94).

Client–server systems are not significantly cheaper than their mainframe-based ancestors; they are neither notably more user friendly nor better integrated into the modern desktop environment. In addition, client–server systems have introduced several new problems—they are less robust, efficient, and portable and, at the same time, are more difficult to implement and deploy than the mainframe-based applications they replace.

PC software maintenance is seldom budgeted, described instead as "upgrades."

Uncovering hidden costs

Discussing the total cost of ownership has rightly become fashionable. TCO covers, apart from the initial purchasing costs, other incidental costs such as money spent on software and hardware upgrades, training, system administration, and user support. Initial CSS cost studies optimistically assumed that, for example, adopting OO methodologies would decrease software maintenance costs as much as 25 percent (Alok Sinha, "Client–Server Computing," *Comm. ACM*, July 1992, pp. 77-98).

These studies compared the low acquisition, administration, and maintenance costs of PC-based autonomous end-user computing with the correspondingly higher costs of the MIS departments. However, they failed to account for many hidden costs, or to discuss the costs of managing thousands of PCs—often under the full control of their individual owners—to provide a functional, integrated, enterprise-level computing platform.

Contrary to common MIS practice, PC software maintenance is seldom budgeted, often described instead as "software upgrades." Similarly, the apparently low cost of system administration often translates into the loss of PC owners' productive time as they install new applications or juggle program patches to obtain a stable configuration.

CSS applications cannot match the user-friendliness of the typical PC spreadsheet. First, the complexities and constraints involved in CSS application development are significantly more intricate than those faced by PC application developers. CSS developers must engineer access methods and user interfaces for potentially huge data sets over limited bandwidth, deal with table cursor schemes, enable multi-user access, and use a fixed number of database handles.

In addition, commercial application vendors can use economies of scale and must, to compete, polish their products to a level of versatility beyond the reach of a typical CSS vertical or custom application developer. The executable program sizes of modern packages reveal that versatility and user friendliness result from accumulated raw effort and are not free by-products of the PC operating system or the application development process.

The move toward CSS has negatively affected many end users, MIS, and developers.

Revealing shortcomings

The integration of CSS applications and modern GUI environments is usually only skin-deep and, at best, consists of similar user interface elements. CSS applications often lack support for common shrink-wrap features such as drag-and-drop, the clipboard, and application scripting.

Further, this mostly visual integration of CSS applications with the GUI desktop has been achieved at great cost. The multivendor nature of CSS has become a curse for system integrators, administrators, developers, and end users. A CSS typically consists of the client and server platforms and their respective (often different) operating systems, a database server, database connectivity software, the client application, the application environment support libraries, and, sometimes, additional middleware and networking software components. Each of these may be supplied by a different vendor. Obtaining a functioning and stable combination of all components and keeping it working through a barrage of new releases and bug fixes is a sisyphean task.

Client–server systems force an unnatural division of application functionality that results in unneeded complexity and duplication. According to the CSS philosophy, many data validation checks can be performed on the client side. To guarantee the application reliability against malfunctioning or out-of-date client software and raw-data uploads, similar checks are often incorporated as constraints on the server side. Apart from the duplication of effort, this strategy increases application complexity because clients must also handle constraint validation errors arising from the database server. To organize this mess and provide scalable solutions for many clients and multiple servers, even more complicated multi-tier and transaction-processing-monitor architectures have been proposed and implemented.

Exposing a flawed design

The malfeasant design decision to use a data definition and manipulation language, SQL, as the standard client–server interprocess-communication protocol has resulted in many architectural and implementation problems. The desktop metaphor's most common operation—the ability to scroll up and down over a large data set—cannot be efficiently implemented on most CSSs despite the deployment of increasingly sophisticated cursor management schemes. The tabular nature of an SQL query's results necessitates an unrewarding juggling act of balancing the inefficiencies of redundant column fetches, the execution of multiple SQL queries, and the fetching of large result sets for processing on the client side.

Further technical and management difficulties stem from the distributed deployment and support of client software. A typical client installation package consists of not just an executable program but also runtime libraries, database connectivity packages, component objects, and, in many cases, changes to some operating system modules.

Installation difficulty is compounded by the inability of most desktop OSs to be remotely managed: most installation procedures require an operator's presence during installation. The installation process is complicated by the interference between installed components that other user-installed application packages share. The primitive version-checking and reference-usage-counting installation protocols, coupled with bugs in the often minimally tested installation procedures, result in shared components that can be removed without warning, regress to older versions, or be replaced with newer but incompatible versions when an application is installed. All these problems are multiplied by the number of clients that must be installed. In addition, because clients perform local processing, client updates must be performed in a single step on all desktops, and be synchronized with updates to the database structure on the server.

Further, the adoption of CSS architectures has resulted in an unfortunate loss of program portability. Popular CSS application-development environments such as Powerbuilder and Visual Basic use proprietary languages or language extensions. Version by version, the suppliers of these environments reinvent mainstream language features such as strong typing, modules, exception handling, structured data types, and OO programming. This proliferation of languages has resulted in the balkanization of training, software reuse, tool support, documentation, coding standards, and revision control. Such development environments mainly target the Microsoft Windows OS and Intel hardware architecture, resulting in a dramatic backslide in application portability. Even though we resolved these important language and software engineering issues in the middle 1970s, they are resurfacing with a vengeance 20 years later.

Applying a Politically Correct Fig Leaf

Overhyped and oversold, CSS architectures have been used to implement many mission-critical applications. Consequently, every day both large and small enterprises face the problems I've described.

Industry analysts now propose several new alternatives that ostensibly improve on the CSS paradigm. A leading example, Cyrix's thin-client technology, is apparently endorsed by Microsoft. Joel Kanter (*Understanding Thin-Client/Server Computing*, Microsoft Press, 1998) defines a thin-client application as one in which all application processing is performed on the server; the client only transmits keyboard and mouse-movement events to the server. This approach, similar in structure to the X Windows system, departs radically from the traditional CSS architecture by moving toward systems based on terminal emulators.

Web solutions based on HTML data exchanges stem from a similar concept, so labeling them client–server systems is misleading. Network computers that run Java clients are, however, closer in spirit to the CSS architecture. They appear to solve the problems of application deployment, implementation language functionality and portability, and installation difficulties. However, their integration with popular applications, the client–server communication protocol, and the client's user friendliness have not been satisfactorily addressed. In summary, these new approaches try to address the CSS architecture's fundamental problems in a politically correct way: they silently retreat and cover up the problem by using fashionable words and technologies.

The move toward CSS has been a large and expensive technological mistake. It has negatively affected many end users, MIS departments, balance sheets, and developers. If we view the industry's response in terms of Elisabeth Kubler-Ross's stages-of-grief model, we are currently past the *denial* and *anger* phases. As we recognize the problems with CSS, we experience the *bargaining* phase in which the architecture's stakeholders strive to re-establish their position with minimal loss, or gain strategic advantages from the impending changes. However, after the inevitable *depression* about the costs and lost opportunities of the move to CSS architectures, we can now *accept* the situation and work toward the design of robust, user-friendly, practical, efficient, portable, cost-effective, and scalable architectures for developing MIS applications. ❖

Diomidis Spinellis is a senior software engineer at SENA S.A. He also lectures at the Department of Information and Communication Systems at the University of the Aegean. Contact Spinellis at SENA S.A., Byzantiou 2, 142 34 Nea Ionia, Greece; dds@senanet.com.