Curl®: A Content Language for the Web

Robert H. Halstead, Jr.
Curl, Incorporated
1 Cambridge Center, 10th floor
Cambridge, MA 02142
+1-617-761-1200
rhh@curl.com

Mat Hostetter
Curl, Incorporated
1 Cambridge Center, 10th floor
Cambridge, MA 02142
+1-617-761-1200
mat@curl.com

David A. Kranz
Curl, Incorporated
1 Cambridge Center, 10th floor
Cambridge, MA 02142
+1-617-761-1200
kranz@curl.com

ABSTRACT
The Web has transformed how we access information and applications, but Web application platforms have lagged behind the vision: Highly usable Web applications are too hard to build using established Web technology platforms. Web content needs to span the spectrum from simple formatted text, through graphics, animations, and scripting, to enterprise-scale applications that use client-side computing. The Curl language is the world's first "content language" spanning this whole spectrum in one unified framework. The Curl language and implementation borrow from many intellectual traditions in computer science, notably including the Lisp tradition. The Curl product is now commercially available and is enjoying rapidly increasing adoption.

Categories and Subject Descriptors
D.3.2 [**Programming Languages**]: Language Classifications – *multiparadigm languages, object-oriented languages.*

General Terms
Languages, Human Factors.

Keywords
Curl, content languages, client-side computing, rich client, Web applications.

1. INTRODUCTION
The trend in today's information technology world is to migrate everything to the Web, but many Web-based applications are less user-friendly than the older applications that they replace. To solve this problem, Web applications need to use client-side computing more effectively. A major reason why this hasn't happened is that established Web technologies are too hard to use for this purpose. What's needed is a *content language* that focuses on the real needs of Web applications and provides all the needed functionality in one unified framework. The Curl language described in this paper provides such a framework. Other sources [4, 8, 11, 24] discuss technical details of the Curl language and implementation in more depth. This paper provides an overall perspective on the problem that the Curl technology addresses and relates Curl's major features to the principal requirements for successfully building Web-deployed client-side applications. Existing programming technologies from which the Curl project was able to borrow are identified, and novel aspects of the Curl approach are noted.

This paper was presented at the International Lisp Conference in June 2005 at Stanford University in California. For more information about the conference, see http://www.international-lisp-conference.org/2005/index.html.

For more information about the Curl technology, see http://www.curl.com/.

1.1 The Evolution of Web Applications
From today's vantage point, it is hard to imagine life without the World-Wide Web, but in 1990 the Web had only begun to be prototyped and was known to just a few people [7, 20]. When the Web did start attracting broader attention, at first it was still "just" a mechanism for publishing information on the Internet in a form that was much easier to access than previously used technologies such as FTP sites and newsgroups.

The first information published on the Web was static textual content such as memos and technical reports, supplemented with images and, of course, links to other information on the Web. Appreciation quickly grew, however, for the dramatic way in which the Web model simplifies access to information on the Internet, and creative minds across the world began to dream up new applications for this new mode of communication. Visionaries saw that not only static information, but also active applications and services, could be delivered using this new medium.

Web old-timers remember early server-side CGI scripts that provided the first active functionality to Web users. These scripts are direct ancestors of the massive server-side applications that drive today's Web sites for e-commerce, customer relationship management, enterprise resource planning, etc. The attraction of accessing any information or application from any networked computer is irresistible, and today every enterprise application vendor either already provides a Web interface or is feeling intense pressure to do so.

A classical server-based Web application is structured as a series of one or more *forms*. A user using a client machine enters information into a form and clicks a Submit or Go button (or something similar) to send this information to a server, which in turn creates an HTML page, frequently another form, and sends it back to the user. This architecture makes few demands on the client machine, which promotes the availability of the application

from any networked computer. Unfortunately, this architecture also brings about the "World-Wide Wait" — that phenomenon where users around the world have to look at an unresponsive screen after clicking the submit button, until the server allocates them their slice of server time to generate the next page that they will see. As a result of this tyranny of the submit button, the usability and responsiveness of such server-based Web applications represent a big step backward from the client/server or locally installed applications that they replace.

1.2 Client-Side Execution Platforms

The value of improving the user experience of Web-based applications was recognized early on, and mechanisms for autonomous client-side computation in browser-based applications were introduced. 1995 saw the introduction of both the Java™ applet mechanism and of JavaScript for simple client-side scripting in the Netscape® browser [2, 3].

Using JavaScript, Web application builders were able to introduce simple interactive elements into Web applications, such as objects whose appearance changes when the mouse pointer is over them. Some rather impressive interactive applications have been built using JavaScript, such as the Google™ Maps beta at http://maps.google.com/, but the design of JavaScript doesn't really support building large-scale applications very effectively, so creating this sort of application using JavaScript is a very expensive proposition.

Meanwhile, proponents of Java applet technology hoped that it would be the technology of choice for serious development of client-side interactive applications that could be deployed on a Web server and executed within a Web browser. Over the ten years since its introduction as a Web technology, Java has greatly influenced the information technology world. It has been widely used for programming on Web *servers*; it has seen increasing use as a tool for developing non-Web applications; it has been widely adopted as a tool for teaching computer science; and it has legitimized the mainstream use of automatically managed, garbage-collected storage for "real" application development. However, even after ten years it has not seen all that much adoption for client-side execution of Web-deployed applications.

Why hasn't Java seen more use for building Web-deployed client-side applications? Several reasons probably have contributed:

- Unpredictability of the client-side platform. The "Java wars" of the late 1990s between Sun and Microsoft, along with ferment in the Java architecture itself (e.g., AWT vs. Swing), left Web application developers facing the challenge of writing code that might run on many different versions of the Java platform and left them feeling unable, in many cases, to use the latest Java features in their applications.
- Distrust of the Java sandbox. Security holes publicized early in the history of Java [18] caused some system administrators to ban or restrict the use of Java applets, reducing the audience for applet-based Web applications.
- Application download size. Although using client-side computing can greatly reduce the total communication with a server over the long run, the initial download for a sophisticated Java application can easily get large, lengthening the initial delay experienced when starting up the application.
- Mismatch between the Java architecture and the requirements for Web content. Web content and applications are first and foremost about presenting information to people. Java's innovations focused on mobile code and sandbox technology, not on the question of how presentation-oriented applications should be structured. In the latter dimension, Java is a fairly typical object-oriented programming language with a fairly typical GUI toolkit.

All of these problems offer learning opportunities for developers of future Web technologies, but the last problem in particular attracted the attention of a research team at MIT headed by Steve Ward, David Kranz, and Chris Terman who mapped out the original Curl language design starting in 1995. They observed that HTML and Java have complementary strengths. Although HTML [21] lacks the active element that would make it usable for building client-side applications, it is actually pretty good for presenting textual and graphical data. It is not at all unreasonable to write a memo, a technical report, or even a textbook using HTML, and in fact many such documents have been created and published on the Web. Java [9], on the other hand, is good for building and operating on data structures — an essential capability for building applications — but hardly anybody would propose to write a memo in Java.

The MIT team recognized that Web content really spans a spectrum from simple textual content to classical application programs. Content that occupies intermediate points on this spectrum is awkward to build using the technologies discussed above. Builders of a presentation-intensive client-side Web application therefore faced two unattractive alternatives: build the entire application on a platform that supports the required programmed behavior but is not ideal for the presentation aspect, or build parts of the application on different platforms (for example, use HTML as much as possible for the presentation part and use a Java applet or applets for the part that requires computation) and endure the extra complexity of integrating these disparate components.

2. THE CURL APPROACH

The Curl language is based on a third approach: define a platform that excels for *both* presentation and programming, so that applications spanning the entire content spectrum can be built efficiently within one unified framework. The term "content language" has been coined to describe the resulting language architecture [24].

Content expressed using the Curl language can be deployed to a Web site and served from there to a client machine, where it is executed and displayed either as an applet embedded in an HTML page or directly as the top-level contents in a browser window. In either case, the content is processed by a Curl runtime system (including a browser plug-in) installed on the client machine. Like HTML content, Curl content can be transmitted from the server to the client as source code. A just-in-time compiler in the Curl runtime compiles this source code to the native machine language of the client machine, which can then be executed with the full performance of the client hardware. (In addition to Curl source code format, a compressed and preprocessed, but not yet compiled, format called "pcurl" is supported for users who want to deploy a more compact file to a Web server or do not want to reveal the original source code.)

2.1 First-Class Treatment of Content

The basic philosophy behind the "content language" concept is to elevate *content for presentation* to an equally first-class status as the classical data structures and abstractions of contemporary programming languages such as Java, C++, or Lisp. The implications of this philosophy are seen even at the level of the language syntax, which unlike the syntax of other familiar languages must be equally as friendly for HTML-like markup as for conventional programming. The infix syntax of Java, C++ [22], and related languages is not well suited to this task, so Curl adopts as its basic syntactic model the prefix syntax originally pioneered by Lisp [17]. In Lisp, each prefix expression begins and ends with a parenthesis, but parentheses are frequently useful in textual content, so Curl chooses the less frequently used "curly braces" `{}` instead. (The use of these braces, in turn, gives the Curl language its name.) Thus, the basic syntax of a Curl expression is just

{*operator operands…*}

where the operator controls the interpretation and processing of the operands. Curl takes this concept considerably beyond traditional language models based on prefix syntax, however. When the operator is an identifier, Curl allows all of the following possibilities:

- Procedure and class names. In this case the operands are interpreted as Curl expressions to be evaluated. The resulting values are supplied as arguments to the procedure or the class constructor. This category of operators corresponds to the subroutines and functions provided by most programming languages.
- *Text procedures*. In this case the operands are treated as text to be formatted, but if an unescaped `{` character appears in the operands, the material from that character to the next matching `}` character is treated as an expression and the result of evaluating that expression is substituted into the text at that point. Curl includes predefined text procedures such as `bold` and `italic` that provide markup functionality similar to that provided by HTML tags. User applications can define and use additional text procedures.
- Macros. In this case the entire operand text, up to the closing `}` character, is passed unmodified to the macro definition, which acts like a rewrite rule and provides a new expression to be used in place of the macro expression. Macros are procedural and hygienic [24], in contrast to the simple declarative rewrite rules offered in languages like C [13], and therefore are very powerful. Many built-in Curl operators such as `if` are actually defined as macros. User applications can also define and use additional macros.
- Primitives. A small number of fundamental Curl features are provided by specially defined primitive operators. This set of operators cannot be extended by user applications.

Despite using the prefix syntax described above as its primary syntactic model, Curl allows the more familiar infix syntax for common arithmetic and relational operators such as + and >. This concession to centuries of mathematical practice helps Curl programming feel familiar even to new developers.

The top-level source code of a Curl applet is treated somewhat like the operand of a text procedure, so the "Hello World" program in Curl can be written as just

```
{curl 3.0 applet}
Hello World!
```

The `{curl 3.0 applet}` herald identifies this file as an applet that uses version 3.0 of the Curl API, and the remaining text is treated simply as content to be displayed, leading to the legend

Hello World!

appearing in the content area of the user's browser window.

This next applet illustrates some of the different kinds of Curl operators:

```
{curl 3.0 applet}
{let a:int = 3}
The square root of {italic a+1} is
{sqrt a+1}.
```

leading to the displayed result

The square root of *a+1* is 2.

This example shows the difference between the text procedure `italic`, used here for markup purposes, and the ordinary procedural operator `sqrt`, used to embed the results of a computation in the presented output. It's notable that even the decision as to whether `a+1` is text to be formatted and displayed, or an expression to be evaluated, must be postponed until after determining the type of the operator in each expression where `a+1` appears as an operand. At the level of syntax, this feature is the most significant difference between the Curl content language and a conventional programming language.

Of course it is a straightforward extension of the above principles to write nested markup expressions such as

```
{italic My {bold big} friend.}
```

or nest computations inside markup as in

```
{bold {sqrt 9} is bigger than {sqrt 4}.}
```

or even nest markup inside computations as in

```
{CommandButton label = {bold Push me!}}
```

And of course all of the above examples become more compelling when simple computational expressions such as `{sqrt 4}` are replaced with computations that exercise the functionality of whole application modules and return complex, interactive graphical objects (possibly containing marked-up text as well as the results of other subcomputations) for display.

2.2 Declarative Content Specification

Since it lacks procedural programming constructs, HTML necessarily provides for describing content in a declarative style. Most widely used programming languages, on the other hand, practically force the use of an imperative approach when building

up data structures. Many pages have been written advocating declarative approaches to programming in general, but the declarative approach is especially well suited to describing graphical presentations of information across the whole spectrum including graphical user interfaces as well as text-heavy presentations like memos and reports. Indeed, the availability of a declarative specification style in HTML and the comparative unavailability of such a style in languages like Java is a major reason why one could write a memo in HTML but it would be very unnatural to write a memo in Java.

To be a good tool for expressing the whole range of content types, then, Curl must provide powerful tools for describing content declaratively. Generally, the content that must be described can be viewed as a tree of nodes, where some nodes may be leaf nodes but others are "container nodes" that have varying numbers of child nodes. Both leaf nodes and container nodes can have various *properties* such as color, font size, etc. Thus, there are two fundamental requirements for a language that supports declarative specification of content:

- Since many kinds of container nodes, such as tables or itemized lists, can have variable numbers of child nodes, it must be possible to define an operator that constructs a container node so that the operator can accept any number of operands. The traditional Lisp [10] concept of "rest arguments" is ideal for this purpose and has been incorporated into Curl.
- Declarative specification of node properties requires some sort of keyword argument syntax. Keyword arguments are well established in the Lisp world and in some scripting languages and are also used for specifying attributes of HTML tags, but they are missing from the C++/Java family of languages. They have been incorporated into Curl.

Combining these features, it becomes possible to write expressions such as

```
{VBox
    background = "yellow",
    {bold font-size = 24pt, Big Title},
    {text font-size = 8pt, Fine print.}
}
```

which specifies a considerable amount of formatting information in a very concise and readable fashion. Here `VBox` is a constructor for a class that stacks graphical objects vertically and `text` is a text procedure that applies no formatting of its own, but serves as an anchor for attaching keyword arguments. `24pt` and `8pt` are examples of Curl's *quantities* syntax to specify font sizes. Quantities are discussed further below.

Graphical user interfaces often need to specify *event handlers* for various objects. This too can be done declaratively, as in the following example:

```
{VBox
    background = "yellow",
    {on PointerEnter at v:VBox do
        set v.background = "orange"
    },
    {on PointerLeave at v:VBox do
        set v.background = "yellow"
    },
    {bold font-size = 24pt, Big Title},
    {text font-size = 8pt, Fine print.}
}
```

This example uses event handlers defined by `on` expressions to change the background color of the `VBox` from yellow to orange whenever the mouse pointer is inside the bounds of the `VBox`. Specifying event handlers is the first step down the road from static content to scripted, interactive content, and it is possible to take this step without departing from the overall declarative style with which static content can be specified.

`background` and `font-size` in the above code are examples of *options*. Specifying attributes of graphical objects is so important in a content language that Curl includes an options mechanism solely for this purpose. Options can be set on a graphical object at creation time using keyword syntax such as the `background = "yellow"` in the above code. Option values can also be read or set imperatively using field-reference syntax like the `v.background` in the above code.

Options come in two varieties. *Local* options such as `width` apply only to the object on which they are set. A binding of a *nonlocal* option such as `font-size` applies to the object on which it is set and also to all graphical descendants of that object, unless the option binding is shadowed by another binding of the same option. Nonlocal options allow a style attribute to be applied across an entire graphical hierarchy by a single operation on the root node of that hierarchy. The options mechanism includes provisions for notifying objects affected by changes in an option binding and for compile-time type checking of option operations.

2.3 Security

Security is an essential concern for any platform that can execute mobile code downloaded from the Internet. The security challenges facing a content language don't really differ from those that face other mobile code platforms, so the starting point for Curl, as with other platforms, is to distinguish *privileged* applets (which come from trusted sources) from *unprivileged* applets (all others).

Unprivileged applets are executed in a sandbox that prohibits operations that could cause damage or access data without authorization. The sandbox guarantees type-safety during execution of unprivileged applets. As is now widely recognized, garbage-collected storage management helps greatly in this process, and the Curl platform uses that approach.

A sandbox that can safely execute unprivileged applets is of little use unless unprivileged applets can actually perform useful functions. Accordingly, the Curl design team has focused considerable energy on expanding the power of unprivileged applets without enabling them to do anything that would actually be unsafe. The Curl platform has some unique features that allow unprivileged applets to have controlled access to third-party server machines, save information persistently on client machines, and access files on the client machine when explicitly authorized by a user. These features reduce the number of situations where privileged applets are required, but their detailed discussion is beyond the scope of this paper.

2.4 Large-Scale Application Development

When an application becomes large, reliable software development requires language features that promote modularity

and enable compile-time detection of program inconsistencies such as type errors. Many large applications also need to interface with external components implemented using different programming languages. Given Curl's focus on removing obstacles that impede the construction of Web applications, providing effective support for large-scale programming has been important.

Virtually the entire implementation of the Curl runtime, the Curl documentation set, and the Curl development tools — over a million lines of code including the Curl compiler itself — has been written in Curl. Thus, we have developed a deep, first-hand appreciation of the challenges of large-scale application development and how the Curl feature set can help.

2.4.1 Type System

A type system — along with a compiler that processes type information — aids the development of large-scale applications in two ways. First, type information provided by a programmer can be used to generate fast application code. Second, compile-time type-consistency checking can uncover many common programming errors, particularly in the use of interfaces between modules of a large application. This checking is particularly valuable during the maintenance or improvement of an already large application, where the original application developers are no longer involved or have already begun to forget details of the application architecture. The Curl team's own experience is that this is a tremendous advantage for reliable software development.

Accordingly, all Curl variables (and similar entities such as class fields, procedure arguments, and procedure return values) are typed. Curl types include the usual primitive types such as `int`, `double`, and `bool`. Class and enumeration types are also supported. The Curl runtime system defines numerous class and enumeration types, and other types of both kinds can be defined in application programs.

2.4.1.1 `any` Types

Although compile-time type declarations do have great software-engineering benefits, certain subsystem interfaces (such as C's `printf` function) are inherently generic and become awkward to define and use if there is no escape from the requirement to define specific types for all values. Languages like Java have a type `Object` that is a supertype of all class types, so interfaces that are generic across class types can be declared using `Object`. However, there is no convenient way to declare an interface whose domain also includes types (such as primitives or enumeration types) that are not subtypes of `Object`. In Java, for example, programmers are forced to explicitly create `Integer` objects in order to pass integer values to generic interfaces. (The C# language [5] recognizes this problem and addresses it by having all types — including `int` — inherit from `Object`.)

To support the graceful construction and use of such generic interfaces, the Curl type system includes an `any` type. A variable declared as `any` can contain any Curl value. An `any` value is tagged so that its type can be determined at run time. Also, unlike a Java `Integer`, a Curl `any` can be used directly as an operand of any operation that is valid for its run-time type, including arithmetic operations and even method calls. This valuable concept is taken directly from the Lisp/Scheme family of languages.

The Curl type system includes several other features designed to support the development of interfaces that are strongly typed without requiring sacrifices in usability. They are discussed in the following subsections.

2.4.1.2 Non-null Types

Many languages include the special value `null` as a member of every class type. While `null` can be useful as a sentinel value, it is often a source of type errors. In Curl, a class type such as `Graphic` does not contain the null value. To support situations where a null sentinel value is desired, the syntax `#Graphic` denotes the class type `Graphic` with the null value adjoined. This feature promotes greater precision in type declarations and prevents null-dereference errors from occurring deep within a subsystem that did not expect to see a null value. Detecting inappropriate uses of `null` at compile time makes them much easier to find and fix.

2.4.1.3 Quantities

Presentation-oriented applications frequently deal with measurements of space and time. Curl accordingly includes "quantity types," which are numeric data types with units, such as `Distance` and `Time`. These types are distinct from the conventional dimensionless numeric types such as `float` and `double`. There are also quantity literals, such as `2.54cm`, `72pt`, `50ms`, `1week`, and even `800dpi` and `32(ft/(s^2))`. By using quantities, programmers can let Curl handle conversions between units and can write more descriptive (and checkable) interfaces to graphical APIs. There is also a pixel-denominated unit `px`, which can be converted to a `Distance` by dividing by the spatial resolution of a display device. Quantity types are not found in other languages for creating Web content, but they are used pervasively in the Curl GUI APIs and are tremendously useful.

2.4.1.4 Unicode Characters and Strings

Unicode [23] has become the standard character encoding for multilingual applications and is adopted by Curl. Special optimizations in the implementation of strings and string buffers mitigate the excess storage consumption that would result from the simple strategy of allocating 4 bytes for each character in these structures.

2.4.1.5 Multiple Inheritance

Class definitions in Curl can inherit from multiple superclasses, not just from multiple interface definitions as in Java. As others have learned in the past, multiple inheritance is tremendously useful in building object-oriented GUI toolkits, where it is common to encounter situations like "a TabContainer is a Control and is also a Box."

2.4.1.6 Types as Values

Curl programs can create values and variables of type `Type`. For pragmatic and even semantic reasons, any type variable or expression that is used in a type declaration must be a constant whose value can be determined at compile time, but `Type` variables can be used freely to access the Curl reflection API and even for constructing an instance of a class whose identity is indicated by the current value of a `Type` variable.

2.4.1.7 Parameterized Classes

The Curl language provides generic classes such as `{Array-of` *type*`}` or `{HashTable-of` *type1*`,` *type2*`}` which can be parameterized using type-valued expressions. Being able to write `{Array-of int}` or `{Array-of String}` adds greatly to the precision with which interfaces can be declared. Storage use can also be optimized: an `{Array-of int}` avoids allocating heap storage for each integer in the array and is only half the size of an `{Array-of int64}`. Finally, `{Array-of any}` can be used, without restriction, for heterogeneous collections of data. User applications can define and use their own parameterized classes.

Parameterized classes are available in some object-oriented languages, such as C++, but were initially omitted from others, such as Java, perhaps because simple implementations of parameterized classes are prone to code bloat caused by generating many slightly modified copies of the same class implementation. The Curl implementation incorporates special optimizations including lazy expansion of class templates to prevent parameterized classes from bloating the generated code or slowing down the JIT compiler [12].

2.4.1.8 Value Classes

Typical object-oriented languages offer primitive types such as `int` that are not heap-allocated, but use heap allocation for all instances of application-defined types. It is occasionally very useful to define a group of values that travel together, along with some behaviors, without invoking heap allocation. For example, in graphics programming it is useful to have a type `Distance2d` that contains x and y coordinates of an object and can be manipulated without heap allocation. A type such as `Distance2d` is much like a class type, except for its storage management discipline. Curl provides for such types as *value classes*, which are built into the type system so that, for example, an `{Array-of Distance2d}` can represent the coordinates of consecutive points in-line in the array, without allocating a heap object for each point. Application-defined value classes are not yet supported, but such support is envisioned in the future.

2.4.1.9 Procedure Types

As in Scheme [14] and Common Lisp [10], procedures are first-class objects in Curl. Procedures also participate in the strong typing system, so the type of a procedure can be described with considerable precision. For example, the expression

```
{proc-type {String}:int}
```

describes the type of a procedure that reads a character string, parses it as an integer literal, and returns the corresponding integer value. The `proc-type` syntax also has notations for keyword and rest arguments. Finally, Curl also borrows from the Lisp family of languages the concept of declaring procedures with multiple return values, which is very useful as a way of returning multiple results from a procedure or method without allocation or side effects.

2.4.2 Modularity and Namespace Control

Like other modern object-oriented programming languages, Curl supports several levels of name scoping to help developers control the visibility of names and build modular applications. Name scoping control begins with the ability to create nested code blocks, where names defined within a block are visible only within that block. The Curl language also allows *anonymous procedures* to be defined within code blocks. A procedure can access variables whose names are visible at the point where the procedure is defined, but can be called from places where those names are not visible. Thus, the classic `twice` function is easily defined in Curl as

```
{define-proc {twice f:any}:any
    {return
        {proc {x:any}:any
            {return {f {f x}}}
        }
    }
}
```

Using `twice`, the expression

```
{value
    let square:any =
        {proc {a:double}:double
            {return a * a}
        }
    let quad:any = {twice square}
    {quad 3.0}
}
```

will calculate the value $3^4 = 81$. (The use of `any` types in the definition of `twice` allows it to accept any function `f` that will not cause a run-time error when used in this way. Type-specialized versions of `twice` could also be defined.)

Anonymous procedures are an important building block for applications. For example, the `on` syntax for event handlers is actually implemented as a macro that creates an anonymous procedure containing the body of the event handler.

Anonymous procedures are also present in (and inspired by) Scheme and Common Lisp. They are not present in fully general form in the Java/C++ family of languages, though the Java "inner class" mechanism provides a workable (albeit more verbose) substitute.

Curl also provides namespace control at the level of class definitions. Names declared as `private` are not visible outside the class definition where they are declared. Modularity on a larger scale is supported by *packages*, each of which functions as a separate naming environment. The visibility of names outside of a package is controlled by `public` and `package` declarations. In this respect Curl's approach is similar to that of Java.

A Web site will often have various different pages or applets that share some functionality such as navigation bars or other user-interface elements. Putting such functions into shared packages promotes code reuse and facilitates giving the entire site a consistent look and feel. Additionally, the Curl runtime has a *package caching* mechanism that caches the compiled code for recently accessed packages on the client computer. When shared functionality is implemented in a package, the package will be downloaded and compiled just the first time a user accesses the package's Web site. As the user navigates from one Curl page on the site to another, the package's functionality will be already available, already compiled, on the client machine, accelerating the startup time for displaying each new page.

Packages can be imported statically, while an application is being compiled, or dynamically, while the application is running. The latter capability can be exploited to improve performance when an application has many features but a typical user will not use all the features in one session. If a feature is put into a dynamically imported package and the feature is never used during a particular user's session with the application, the user will never have to wait for the package that implements that feature to be downloaded and compiled. For example, if a data-mining application offers a large number of chart types and each chart type is implemented in a separate, dynamically imported package, a user will have to experience only the delay of importing the chart types that are actually used.

2.4.3 Extensibility

The development of a large application is often aided by developing a special "vocabulary" of concepts suited to the task at hand. Accordingly, Curl applications can define new Curl operators of every kind except primitives. The provisions for defining new procedures and classes have already been discussed. The Curl language also has constructs for defining new text procedures and macros.

New text procedures can be defined from whole cloth using `define-text-proc`, or can be derived from existing text procedures using `define-text-format` in the following style:

```
{define-text-format warning as text with
    font-size = 24pt,
    font-weight = "bold",
    color = "red"
}
```

Defining a package that exports several public text formats is not so different from defining a style sheet for a text document.

Curl applications can also define and use their own macros. Using macros, application builders can extend the Curl language in almost limitless ways, even embedding whole sublanguages within it. For example, macros have even been used to embed a rule-based logic-programming language within Curl (briefly mentioned in [24]). The macro feature is described in much more depth in [24].

2.4.4 Interfacing to External Components

No application platform can afford to be an "island universe," no matter how good its implementation or feature set. Therefore, it is important that Curl code can interface with components not built using Curl. In the case of Web-deployed applications, there are two ways for this to happen: client-side Curl code interacting with non-Curl code on a server, and client-side Curl code interacting with other client-side code not built using Curl.

The first of these scenarios is the common one, and it is supported by many features of the Curl platform including the ability for client-side applets to communicate with servers using the HTTP protocol and open TCP and UDP sockets. The Curl platform also contains built-in support for processing data in XML format and using Web services via the SOAP protocol [19]. Layered on top of these capabilities are features for communicating with server-side databases and building database-driven applications.

Developers of Curl applications may also want to interface to non-Curl components on client machines, for example, to use an image-processing library or embed a media player in a Curl applet. Such capabilities are clearly incompatible with the security restrictions on unprivileged applets, but they are available to privileged applets, which can link to dynamically loaded libraries on the client machine and (on Microsoft Windows) can interface to COM and ActiveX objects. Application developers do need to think carefully before using these capabilities, because an applet that uses them will only work correctly on a client machine where the needed external components are available. Developers of a locally installed application can just add the needed components to their install kits, but the advantages of a Web-deploying an application are reduced if custom client-side software needs to be installed for it.

2.5 "Programming" vs. "Scripting"

Programmers often distinguish between "scripting" languages and full-blown application development languages. Scripting languages are thought to promote faster prototyping and development for small projects because their test-edit-run cycle is very quick and programmers don't have to spend much effort declaring types or otherwise setting up an overall architecture for an application before starting implementation.

In trade for these benefits, typical scripting languages have much poorer run-time performance because they use interpretation rather than compilation. Also, without programmer-supplied type declarations or other architectural information (such as declarations specifying the visibility of class definitions and class members), a scripting language implementation typically can't do much checking for programmer errors. This can be an especially big problem for maintainers who are not the original application authors.

Unfortunately it is fairly common for a project that started out as a small scripting project to end up as a large application. Thus, the scripting vs. serious programming distinction should be viewed not as a dichotomy but as a spectrum of development situations. The Curl approach to this spectrum is to identify the needs of developers all along the spectrum and address those needs within the one unified Curl framework. This approach gives the Curl technology a "gentle slope" property [11]: As developers' needs evolve, they can evolve their approach without ever having to junk their work and start over with a different technology.

Actually, it makes a lot of sense to think of the yearning for "scripting" as just a desire for simplicity: The test-edit-run cycle should be fast and uncomplicated and the programming notation itself should be clean and free of repetitive "boilerplate" code. But these attributes are desirable at any stage of the programming process! Accordingly, Curl is oriented toward making this simple life possible for all programmers. The use of a JIT compiler already streamlines the test-edit-run cycle. Various features discussed above, such as declarative content specification, options, the `on` operator, and anonymous procedures, already streamline the coding process and lead to largely boilerplate-free code. The following additional features further help streamline Curl application code:

- Class types can be defined with *implicit constructors* that automatically cast values from specified other types to the

class type. For example, the `background` option on graphical objects is declared to be of type `Background`, but the `Background` class is defined to implicitly map objects of type `String` (and also several other types) to `Background`. The upshot is that a programmer can write

```
set v.background = "orange"
```

and the character string `"orange"` will automatically be converted to the needed `Background` object.

- Classes can have *getters* and *setters* (collectively known as *accessors*) that are invoked using the same syntax as field references but have implementations that are like methods. For example, the parent of a graphical object `g` can be obtained by the getter invocation `g.parent` instead of requiring a method call like `{g.get-parent}`. Similarly, if an object `obj` has a `label` setter, then a programmer can write

  ```
  set obj.label = "Hello!"
  ```

  instead of `{obj.set-label "Hello!"}`. The impact of this feature in any one place is minor, but the aggregate simplification across a whole program is significant.
- Similarly `a[x]` is syntactic sugar for `{a.get x}` (no matter what type of object `a` is) and

  ```
  set a[x] = something
  ```

  is equivalent to `{a.set x,` *something*`}`. This mechanism generalizes to multidimensional subscripting syntax such as `a[x, y, z]`.
- Finally, programmers need not explicitly declare the type of all variables. Variables whose types are not declared are assumed to be of type `any`. Scripting-oriented developers can exploit this feature to program with fewer keystrokes, but the advantages of strong data types are compelling enough that most Curl programmers soon train themselves to include type declarations even when in a scripting frame of mind. For serious application development, compiler directives are available to trigger error messages whenever a type declaration is omitted.

2.6 Performance

From the beginning, a major Curl goal has been to provide high enough performance that application developers will feel safe in aggressively moving as much of a Web application's computing load as possible from the server to the client machine. As experienced by a user, performance really has two components: application startup time and the "zippiness" of running the application itself. Application startup time, in turn, has two components: the time to download the application bytes from the server and the time for the JIT compiler to prepare the executable native code.

Since the format of the downloaded application is source code (or the preprocessed "pcurl" form), all the measures discussed above for enabling compact application code also reduce download time. This is a major advantage of the Curl approach over the approach of loading already expanded and compiled bytecode files. The speed of the JIT compiler and the "zippiness" of applications are both kept high by designing the Curl language so it can be compiled into competent machine code without heroic compile-time processing.

Startup time for document-like applications is also enhanced by *incremental evaluation* of applet contents [24], so displayed content begins to appear even while the applet file is still being read from the server. This technique is a generalization of the strategy used by browsers that begin displaying the contents of an HTML file even while they continue receiving the remainder of the file from the server. Finally, the package caching technique discussed above allows the download and compilation steps to be completely bypassed for packages that have been used recently on the client machine.

2.7 Versioning

Web application authors face the problem that client machine configurations evolve over time and are largely beyond the authors' control. For today's mainstream Web applications, this means a quality engineering nightmare in which applications must be designed and tested for a wide range of browsers, browser versions, and operating systems. Even the smallest application change becomes expensive and time-consuming under these conditions.

The Curl versioning architecture sets the stage for the evolution of the Curl platform while providing a predictable environment for the execution of Curl applications. Different versions of the Curl API are identified by version numbers, and each application component such as an applet or package begins with a *herald* such as

```
{curl 3.0 applet}
```

indicating the API version or versions for which the component has been developed. The Curl platform is designed so that multiple versions of the Curl runtime can be installed simultaneously on the same client machine. An applet is automatically routed to an installed runtime version that supports one of the applet's specified API version numbers, or if no such runtime version is installed, the user will see a popup window with information about how to get an install kit for the needed runtime version. This versioning feature means that application developers can focus their testing efforts on just the runtime version or versions that they have specified in the application's heralds. The feature also protects applications against "bit rot," where future changes in a platform cause previously working applications to stop working.

2.8 Other Features

It takes many more features than just those listed above to make a complete application platform. There isn't space to list all of the Curl features here, but the following items deserve at least a brief mention:

- An exception and `try`/`catch` mechanism similar to those of Java and C++.
- An `evaluate` procedure similar to those found in Lisp and Scheme. Using `evaluate`, an application can create Curl source code at run time and cause it to be compiled and executed in a chosen environment (package). This feature adds an important kind of universality analogous in many ways to the benefits of having the `any` type.
- A sophisticated GUI toolkit featuring layout based on boxes and *elastics*, inspired by the "boxes and glue" layout model of TeX [15] and LaTeX [16]. A major goal of this toolkit is

that hierarchies of graphical objects just naturally assemble themselves into reasonable layouts within the available display space without requiring excessive coding to micromanage the layout manager, contributing to the overall goal of achieving a compact application coding style.

- *Occasionally connected computing* (OCC) support. A major advantage of client-side execution over server-side execution is that client-side execution can continue even when the client machine is off-line. Capitalizing on this advantage, the Curl platform supports OCC by providing a way for a Web-deployed application to be copied to a reserved area on a client machine so that the application can be executed even when the machine is not connected to the network.
- A comprehensive library of other features, including scientific subroutines, number formatting and parsing, locale support, 3D graphics, date/time processing, timers, and many more.

2.9 Multiple Platform Support

Despite the current dominance of Microsoft Windows, any technology for building Web applications must be inherently platform-independent if it is to promote the goal of accessing any information or application from any networked computer. Being mostly implemented in Curl, the Curl runtime is designed to be inherently portable. Implementations are currently available on Windows and Linux, and a Macintosh port is planned. Interesting possibilities beyond that include mobile platforms such as cellular phones, which are beginning to boast enough RAM and enough processing power to make such a port a real possibility.

3. APPLICATIONS

Because of its support for the whole content spectrum, the range of applications for the Curl technology is inherently broad. To date, most commercial applications have been Web-deployed, data-driven enterprise applications: for example, tools for browsing and visualizing enterprise performance data such as sales figures. Some B2C applications have also been created and deployed: for example, a premium tool for bank customers to visualize the state of their investments and an interactive on-line tour of a museum. The same advantages that make Curl useful for these applications make it useful for many other enterprise applications, as well as other B2B and B2C applications including e-commerce, ticket sales, on-line tax forms, etc.

The increased responsiveness that results when presentation logic moves from the server side to the client side benefits the users of such applications, but the client's interaction is still exclusively with one server. Client-side execution enables more radical application architectures than this, however. A rich-client technology such as Curl fundamentally shifts the center of control of a Web application from the server machine to the client machine. Executing on the client machine, an application can act as the user's *agent* in accessing a broad range of services and information sources on the Internet, not just the server on which the application was deployed.

This model is a natural match for the "Web services" paradigm [1], and suggests a new Web application architecture in which services and information are unbundled, packaged as Web services, and distributed across the network, with a rich-client application running right on a user's personal computer reaching out across the network and directly accessing Web services on the user's behalf. For example, a portfolio management application could collect information about a user's holdings from one server, security prices from another, and research and news from yet other servers, bringing all this information together for the user. Such an architecture cuts out the middleman in many current Web services approaches, which involve application server machines communicating with Web services on other servers, then packaging up the results and sending them to the user's client machine.

"Web applications" are not the only domain where a content language can be useful. The easy fusion of text, graphics, and active content makes Curl an ideal platform for interactive textbooks, instruction manuals, or training aids. Indeed, the Curl documentation set is itself written entirely in Curl, making it easy to incorporate interactive examples right into the documentation, so readers can immediately test their understanding of the material presented.

4. STATUS OF THE CURL PROJECT

After its initial incubation at MIT, Curl development moved to Curl Corporation, a startup company. In 2004, Curl became a part of Sumisho Computer Systems, a Japanese software company. Several product versions of the Curl runtime as well as Curl development tools have been released and several Curl-based applications have been developed and deployed, chiefly in Japan. Current information can be found at http://www.curl.com.

5. CONCLUSION

It is time to finally realize the ten-year-old vision of Web-deployed rich-client applications. Actual experience has confirmed that the "content language" approach exemplified by the Curl language can successfully support such applications. In developing this approach, the Curl design team has admired and adopted many features of the Lisp family of languages, notably the following:

- Prefix syntax, which provides a much better basis for syntax extension than C++/Java syntax. Lisp's macros provide a powerful model of how syntax extensions can be defined.
- Keyword and rest arguments, which really help declarative content specification and code compactness. However, we do prefer the infix `x = y` syntax over Lisp's keyword argument syntax.
- Automatic garbage collection.
- Anonymous procedures.
- Dynamic type checking and the availability of the `any` type.

We thought it best to diverge from the Lisp approach in the following respects:

- We are confirmed believers in strong static typing as the fundamental typing model, for the reasons discussed earlier.
- Lisp implementations are optimized to operate on `any` objects as fast as possible. We are willing to accept slower performance for `any`s in return for achieving the full performance of platform-native data representations for values not declared as `any`.
- We find the performance advantages of C++-style static classes over Lisp-style dynamic classes to be compelling.

- We think the ability to define parameterized classes is important.
- Lisp is famous for its proliferation of parentheses. We saw this as a potential barrier to adoption and decided to support infix syntax for the most common operations. As a result, we believe the frequency of `{}` and `()` delimiters in Curl programs is no greater than in comparable C++ or Java programs.

We are even willing to voice the undoubtedly controversial opinion that Lisp itself would have been improved by using Curl's approach in the above respects.

Finally, Curl diverges from Lisp in some ways that reflect Curl's special mission as a content language for mobile code:

- Sandbox and security features.
- Text procedures.
- Options.

Overall, the strongest technical influences on the Curl design project have been the Lisp languages, C++, HTML, and TeX. The project has focused on defining a single coherent framework that could accommodate the best ideas that each language brings to its area of strength. Taken as a whole, the architecture derived from this process comprises a "critical mass" that can finally get us over the hump and make Web-deployed applications as good as — or better than — the best non-Web applications of the past.

6. ACKNOWLEDGMENTS

The Curl language design and implementation are the result of many years' work by a "cast of thousands." Major technical contributions described in this paper came from Steve Ward, Chris Terman, and many other contributors, too numerous to list here, at MIT, at Curl Corporation, and at Curl, Incorporated.

7. REFERENCES

[1] Booth, D., *et al*. *Web Services Architecture*, W3C Working Group Note 11 February 2004. See http://www.w3.org/TR/ws-arch/.

[2] Byous, J. Java technology: the early years. Sun Developer Network, 1998. See http://java.sun.com/features/1998/05/birthday.html.

[3] Champeon, S. JavaScript: how did we get here?, April 2001. See http://www.oreillynet.com/pub/a/javascript/2001/04/06/js_history.html.

[4] Damle, N., *et al*. *Curl Programming Bible*, Wiley, 2002.

[5] ECMA TC39/TG2. *C# Language Specification*, October 2002.

[6] Flanagan, D. *JavaScript: The Definitive Guide*, 4th ed., O'Reilly and Associates, 2001.

[7] Ford, A. *Spinning the Web: How to Provide Information on the Internet*, ITPS, London, 1994. See http://andrew-ford.com/stw/stw.html.

[8] Gordon, M., *et al*. *Early Adopter Curl*, Wrox Press, Birmingham, England, 2001.

[9] Gosling, J., Joy, W., and Steele, G. *The Java Language Specification*, Addison-Wesley, 1997.

[10] Graham, P. *ANSI Common Lisp*, Prentice-Hall, 1995.

[11] Hostetter, M. *et al*. Curl: A gentle slope language for the web. *World Wide Web Journal*, Vol. 2, No. 2, Spring 1997. See http://www.w3journal.com/6/s3.kranz.html.

[12] Hostetter, M., and Kranz, D. Lazy compilation of template-generated classes in dynamic compilation execution environments, U.S. Patent 6,760,905, issued July 2004.

[13] ISO/IEC 9899:1999. *Programming Languages – C*, ISO/IEC JTC1/SC22/WG14 standard, 1999.

[14] Kelsey, R., Clinger, W., and Rees, J. (eds.) Revised^5 report on the algorithmic language scheme. *Higher-Order and Symbolic Computation*, Vol. 11, No. 1, August 1998. See http://www.swiss.ai.mit.edu/~jaffer/r5rs_toc.html.

[15] Knuth, D. *The TeXbook*, Addison-Wesley, 1984.

[16] Lamport, L. *LaTeX: A Document Preparation System*, Addison-Wesley, 1986.

[17] McCarthy, J., *et al*. *LISP 1.5 Programmer's Manual*, MIT Press, 1965.

[18] McGraw, G., and Felten, E. *Securing Java*, John Wiley & Sons, 1999. See http://www.securingjava.com.

[19] Mitra, N. (ed.) *SOAP Version 1.2 Part 0: Primer*, W3C Recommendation 24 June 2003. See http://www.w3.org/TR/2003/REC-soap12-part0-20030624/.

[20] Raggett, D. *et al. Raggett on HTML 4*, Addison Wesley Longman, 1998, Chapter 2. See http://www.w3.org/People/Raggett/book4/ch02.html.

[21] Raggett, D., Le Hors, A., and Jacobs, I. *HTML 4.01 Specification*, W3C Recommendation 24 December 1999. See http://www.w3.org/TR/html4/.

[22] Stroustrup, B. *The C++ Programming Language*, Addison-Wesley, 1986.

[23] The Unicode Consortium, *The Unicode Standard, Version 4.0*, Addison-Wesley, 2003.

[24] Ward, S., and Hostetter, M. Curl: a language for web content. *International J. of Web Eng. and Technology*, Vol. 1, No. 1, 2003, 41-62. See http://www.inderscience.com/.