TIBET(tm): Enabling Cost-Effective & Scalable Web Services/Applications

Scott Shattuck

Technical Pursuit Inc.

11268 Grove St. #G

Westminster, CO 80031

ss@technicalpursuit.com

William J. Edney

Technical Pursuit Inc.

1597 Milbridge Drive

Chesterfield, MO 63017

bedney@technicalpursuit.com

ABSTRACT

Client-side processing can reduce server/network loads and user wait times for typical web applications by 90% or more. Leveraging JavaScript for these tasks rather than applets or plug-ins provides barrier-free deployment, seamless integration with server-side tools, and W3C standards-compliance, without browser upgrades. This paper introduces the technology underlying TIBET™, a JavaScript framework that achieves these goals in 120K, small enough for the EPROMS of embedded web servers but powerful enough to support advanced web applications.

Keywords

Web Services, Embedded Devices, Scalability, XML, SAX2, XForms, SVG, Client-Side, JavaScript

1.0 INTRODUCTION

While server-centric designs might imply otherwise, the user's browser is where web applications run. Server requests are essentially remote calls whose performance is affected by network congestion and server load. On the other hand, JavaScript runs locally. C/C++ or Java may be "faster" but they're not faster after a server round trip -- and the user is waiting. Minimizing that wait means using JavaScript.

2.0 FOUNDATIONS

2.1 "Repairing" JavaScript

Inheritance keeps client applications small, promotes quality via reuse, and allows subtyping to solve browser-specific issues. JavaScript inheritance is broken however. The "fundamental guarantee of object-oriented systems" is that instances of a type behave as proper instances of their supertypes[1]. JavaScript types are instances of `Function` implying a flat type hierarchy that violates this guarantee when using type methods. TIBET's solution is shown in Figure 1:

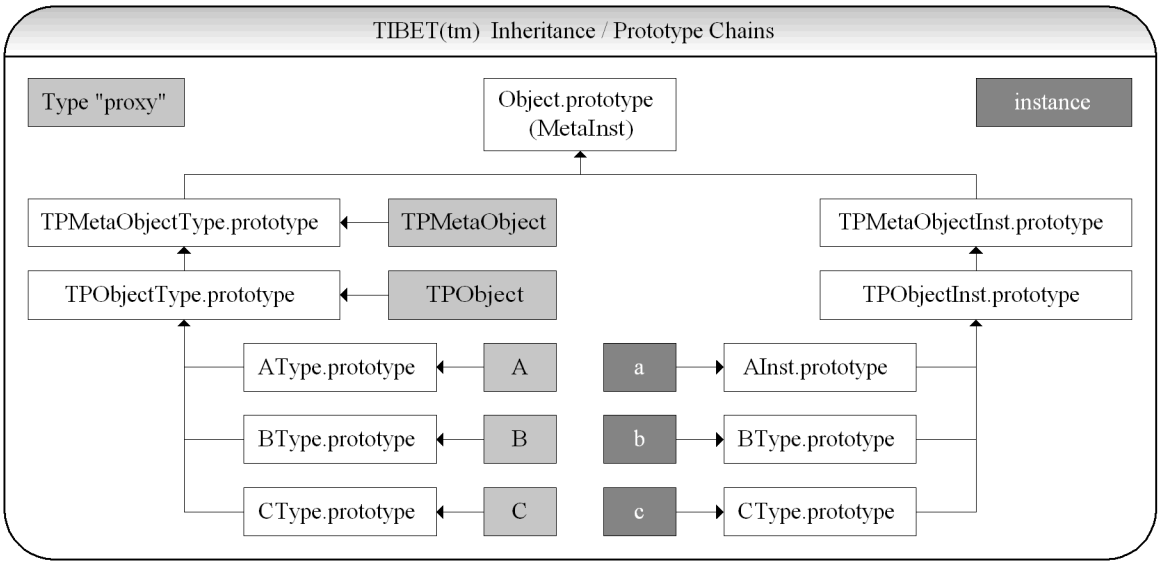

Figure 1: TIBET Inheritance Model

TIBET type proxies are not Functions. The meta-object system manages inheritance by leveraging `create()` rather than `new`, enabling type "spoofing" , single and multiple inheritance, type clusters [2], and instance pooling.

Types are defined via `addSubtype()`, "supertypes" via `addGuardian()`,attributes via `add*Attribute()`, and methods via `add*Method()`, where * represents "Type", "Inst", or "Local". Each function updates reflection metadata. As shown, JavaScript's native lookup mechanism is leveraged while a `callNextMethod()` method supports overriding.

Each `add*Method()` call places a `doesNotUnderstand()`"backstop" on `Object.prototype` that, when triggered, checks "guardians" for an implementation, providing multiple inheritance. If no guardian implementation is found but the object can be converted to a type implementing the method, a method to convert the receiver and invoke the requested method is defined and invoked, resolving the request while avoiding future inferencing overhead.

A `set('attribute',value)/get('attribute')` API supports encapsulation while reducing access methods.

Via inheritance, inferencing, and set/get, TIBET is able to supply 350 types and 2500 methods – in 120K. Small enough to be served from the EPROMS of embedded devices that can't leverage today's heavyweight server-centric tools.

TIBET is heavily event-based. An `observe()/ignore()` API unifies handling of UI events, exceptions, and other signal types while remaining open to distributed event processing. State-change signals support Model/View/Controller (MVC) [3].

2.2 Page/Data Capture

Full support for client-side MVC requires that page templates (views) and data (models) be captured and encached for reuse.

Page templates can be captured by writing an opening TEXTAREA tag in the document head and a closing tag at the end of the document via <SCRIPT> blocks. In newer browsers an XML/HTTP request can be used.

Web-service data capture is possible via SOAP interfaces in newer browsers. Invoking CGI requests that return JavaScript can also capture data. For "stateful" or non-HTTP connections the Flash5 plug-in's socket or a minimal socket applet may be used. Signing the applet enables peer-to-peer web applications where distributed event processing can be leveraged.

TIBET uses request signals to acquire data or page templates. TIBET cache objects observe request signals, allowing them to return cached results, or defer to the server. Requestors observe response signals, allowing them to handle asynchronous results in a consistent fashion.

Separate frames, iframes, or layers are used for caching to avoid releasing encached code or data `onload`.

The value of client-side MVC is clear from analysis of an Amazon.com book search using "JavaScript". Viewing all books via a 56K modem requires 12 server hits, 1080K, and as much as 4 minutes. The same 12 pages rendered via TIBET require 1 server hit, 120K, and 40 seconds [4]. Individual page draws via TIBET are sub-second and require no server interaction.

While broadband-based clients might narrow the time gap, the server hit and bandwidth reductions don't change.

2.3 Inline Scripting

Web development relies heavily on inline scripting technologies such as PHP and JavaServer Pages (JSP). TIBET's Active Client Pages™ (ACP) supports client-side inline scripting. Here's "Hello World":

```
&{return 'Hello World'}&.
```

Rather than a new language or templating paradigm, ACP is simply JavaScript delimited by &{ and }&, a nestable variant of Netscape's JavaScript entities. In the following example ACP is used within an <input> tag where <SCRIPT> tags aren't valid:

```
Date: <input type="text" value="&{return Date.create().asString()}&"
      onchange="TIBET.getObjectWithID("currentEmp").set('hireDate',this.value)" />
```

As with JSP, you can inline ACP anywhere in the page. ACP expressions are converted to functions by removing &'s and adding the function keyword and final ()'s and may be nested. Since ACP expressions aren't compliant XML, they're transformed into entity references prior to parsing. ACP can alternately be placed in an inline DTD before transmission.

Parsing pages allows complex ACP constructs to be reused via a custom tag API mirroring the JSP Tag Extension API [5].

2.4 SAX2 Parsing

Combining SAX2 parsing with custom types is the foundation of TIBET's Active Client Tags™ technology.

Parsing begins with assignment of a potentially URL-specific type as the event target. Mapping types to URLs supports page-specific optimizations. As each SAX2 event is thrown, the current target determines whether a custom type is mapped to the namespace or tag. If so, all SAX2 targets are updated and parsing continues. Type methods implement the SAX2 event interface and generate tag instance lifecycle code while instance methods implement lifecycle interfaces. Default lifecycles match the JSP Tag Extension API but can be extended via custom page, namespace, or tag types.

The current JavaScript SAX2 parser processes the 400K+ XML specification on a 1Ghz Pentium 4 in less than 4 seconds, a rough figure of 100K/second. Leveraging browser implementations of SAX2 will further improve performance.

2.5 Page Compilation

As templates are parsed their dynamic elements are transformed into functions, just as JSP processing transforms pages into servlets. In TIBET, each page becomes a function stored by the browser and ready for invocation/rendering. Compiled pages can be saved and used as an alternative to XML templates, skipping the parse/compile phases.

3.0 XML APPLICATIONS

3.1 Tag Libraries

TIBET's TPTag type and its subtypes support construction of powerful client-side tag libraries, including appropriate subsets of JSP, ColdFusion, or other existing tag sets. Such libraries can also include W3C tag sets such as SVG and XForms.

3.2 SVG & Inline Markup

Mapping types to namespaces enables inline SVG in browsers that don't natively support SVG. When the SVG namespace is encountered the SVG tag type simply writes out the plug-in references and targets the plug-in with captured SVG content.

SVG widget tags are also possible. These might include graphs and charts, or UI elements such as buttons, sliders, tabs, etc.

3.3 XForms In 4x+ Browsers

The TIBET Active Client Forms™ (ACF) project is already underway developing an XForms library that translates XForms markup into XHTML/JavaScript suitable for a 4X browser. An ACF version capable of generating SVG forms is in design. XForms and SVG support in the client take on new meaning when combined with client-side web services access.

3.4 Web Services

Providers offering a web services interface will likely phase out their existing interface to avoid maintaining multiple APIs. For server-centric applications this implies data will be wrapped in SOAP to provide a web service interface, then immediately unwrapped and reformatted for browser users. Shaded elements in Figures 2 & 3 carry load for the provider:

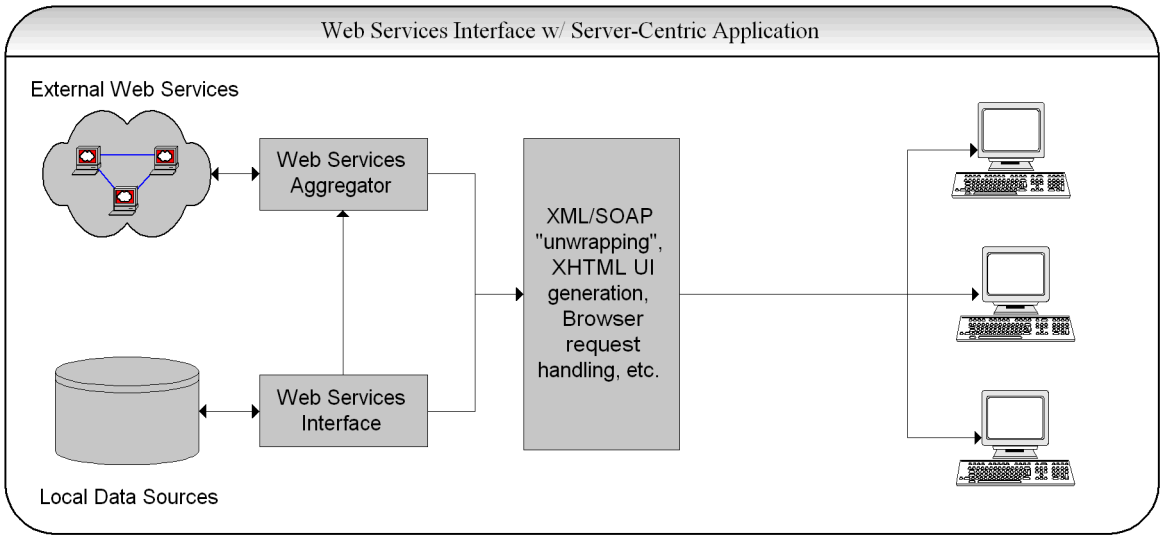

Figure 2: Server-Side Web Services Applications

Client-side systems distribute added service overhead. See Figure 3:

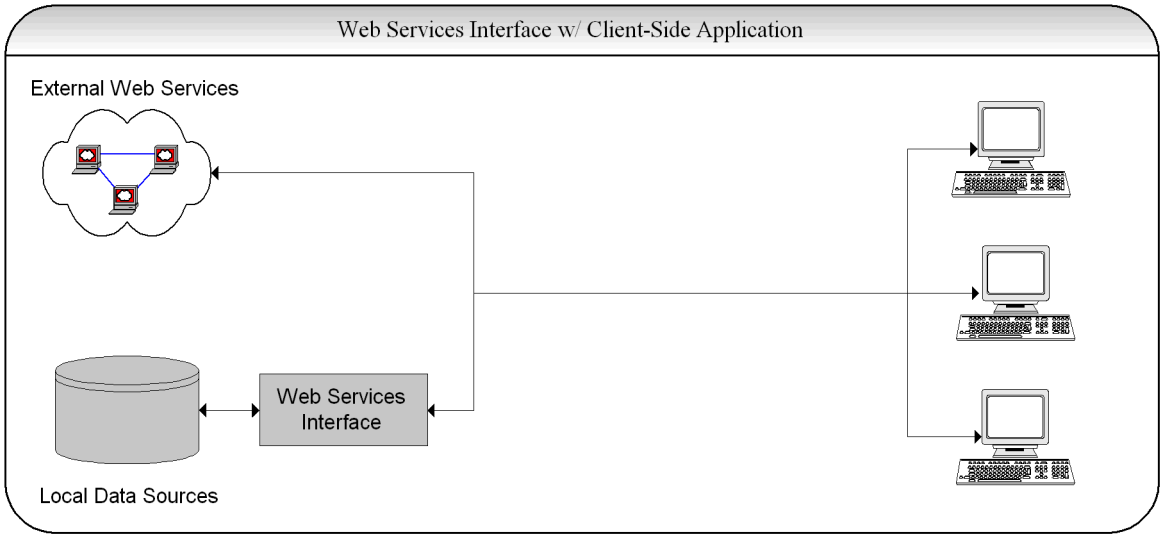

Figure 3: Client-Side Web Services Applications

The client/server web. Server and network loads shift to the user.

4.0 SUMMARY

Web services allow servers to be just that, servers. Clients receive data with no UI clutter. If those clients are web browsers the result is a client-server model with web services in the bottom tier. TIBET provides client-server's inherent scalability and cost-efficiency improvements while preserving page templating, custom tags, and other powerful web paradigms. Combining web services data with SVG-rendered XForms templates is the future. We just see it happening in the client.

TIBET is patent-pending, but is dual-licensed to support both closed and open source developers.

ACKNOWLEDGEMENTS

Scott Shattuck and William J. Edney created TIBET.

Jim Bowery inspired guardians and inferencing.

Scott Severtson developed the current SAX2 parser.

REFERENCES

1. Forman, I.R. and S.H. Danforth, Putting Metaclasses to Work, Addison-Wesley, 1999.
2. Gordon S. Novak Jr., Software Reuse by Specialization of Generic Procedures through Views. IEEE Transactions On Software Engineering, 23(7):401 -- 417, July 1997.
3. Steve Burbeck, Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC) 1987, 1992
4. http://www.technicalpursuit.com/Biz_valueprop.html
5. Sun Microsystems Inc., JavaServer Pages™ Specification 1.2, September 2001

APPENDIX

None