Visual Programming of Location-Based Services

A. Bottaro[2], E. Marino[1], F. Milicchio[1], A. Paoluzzi[1], M. Rosina[2], and F. Spini[1]

[1] CAD Laboratory, Dept of Computer Science and Aut., Roma Tre University, Via della Vasca navale 79, 00146 Roma, Italy
[2] SOGEI S.p.a., Via M. Carucci 99, 00143 Roma, Italy

Abstract. In this paper we discuss a visual programming environment for design and rapid prototyping of web applications, securely connected to remote Location-Based Services. This visual programming approach is based on computation as data transformation within a dataflow, and on visual composition of web services. The VisPro environment uses a very simple approach to service composition: (a) the developer takes a set of web widgets from a library, (b) builds a user interface by drag and drop, (c) builds the application logic of the web service by drawing the connections between boxes (standing for suitable data transformations) and widgets (standing for user interaction). The development session produces, in presentation mode, a web page where the user may trigger, and interact with, the resulting compound service and related computations. A successful GUI (and logic) is abstracted as a new service, characterized as a widget, and stored in the widget library.

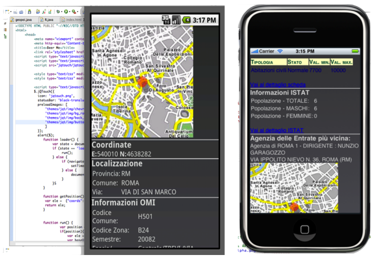
Keywords: Location-Based Services, visual programming, web service

1 Introduction

Cartographic Applications, traditionally concerning projects related to digital cartography, like Geographic Information Systems, Digital Elevation Models, Terrain Analysis and Remote Sensing, have always required a great amount of vertical specialization, with reference to the used technologies, and to the professional skills needed to implement the applications.

The new world of web changed this scenario radically; cartographic applications moved quickly from GIS technology (Geographic Information Systems), currently downgraded to fill useful but peculiar application niches, towards *Geographic Information Services.* In few easy words: from GIS to *ubiquitous GIS*. Such novel ubiquitous GIS is thoroughly oriented to, and supported by, information services connected to geolocalization — i.e. Location Based Service. LBSs provide advanced cartographical interfaces directly at final user's disposal, and may convey easy customization of information services that, hinged around localization attributes, become flexible, interactive, and strongly integrable.

At the present time, web techology is going to make available software frameworks usable as SaaS (Software as a Service) components, that put the user in control of programming—through suitable libraries (API/SDK)—highly specialized services, which are instantly deployable within the departments of a public institution or private company. The predictable future of this scenario, looking beyond the infrastructural components and the evolution of technological platforms, requires the adoption /development of novel visual programming environments, that must enforce the easiest (i.e. strongly based on drag-and-drop) composition of elementary programmable components, in order to build highly specialized applications and location-based services.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HT
<html>
    <head>
        <meta name="viewport" conte
        <meta http-equiv="Content-t
        <title>Beer Me</title>
        <link rel="stylesheet" href
        <script type="text/javascri
        <script type="text/javascri
        <script src="jqtouch/jqtouc

        <style type="text/css" medi
        <style type="text/css" medi

        <script type="text/javascri
        $.jQTouch({
          icon: 'jqtouch.png',
          statusBar: 'black-translu
          preloadImages: [
            'themes/jqt/img/chevr
            'themes/jqt/img/bg_ro
            'themes/jqt/img/back_
            'themes/jqt/img/butto
          ]
        });
        alert($);
          function loader() {
            var state = docum
            if (state == 'loa
              run();
            } else {
              if (navigato
                setTime
              } else {
                documen
              }
            }S
          }

          function getPosition()
            var ele = {"coords":
            return ele;
          }

          function run() {
            var position
            if(position){
              var ele =
              var hasell
```

Coordinate
E:540010 N:4638282

Localizzazione
Provincia: RM
Comune: ROMA
Via: VIA DI SAN MARCO

Informazioni OMI
Codice Comune: H501
Codice Zona: B24
Semestre: 20082

Tipologia	Stato	Val. min.	Val. max.
Abitazioni civili	Normale	7700	10000

Informazioni ISTAT
Popolazione - TOTALE: 6
Popolazione - MASCHI: 6
Popolazione - FEMMINE: 0

Agenzia delle Entrate più vicina:
Agenzia di ROMA 1 - DIRIGENTE : NUNZIO GARAGOZZO
VIA IPPOLITO NIEVO N. 36, ROMA (RM)

Fig. 1. Example of LBS (Location-Based Service) developed by SOGEI, a company fully owned by Italian Ministry of Economy and Finance, for a related Agency.

This predictable trend is especially desirable when dealing with applications of location intelligence, where sophisticated techniques of business intelligence require a real-time visualization of discrete events or diffuse situations that may evolve dynamically at various geographical levels, i.e. either countrywide or within specified regional districts. Such analyses may concern also investigation activities, in fraud prevention and hampering, that may require autonomous elaboration capability with high levels of secrecy.

2 Background

The visual programming approach of this research is based on a few simple-but-pervasive concepts namely (a) computation as data transformation within a dataflow, and (b) visual composition of web services that are going to shape the rapid application development in the next years, where on-demand enterprise IT services cry for a dynamically configurable architecture of the service process engine [1]. Whereas other approches [5, 1] use either web reasoning or planning [1] or Petri nets [7] for service composition, as in Ref. [3] we use a simple but effective dataflow model. We briefly describe in the following the two fundamental abstractions that are used extensively in the project. Let us just recall that abstraction is the process of ignoring the details, and to focus on the overall design (the big picture).

2.1 Composition of web services

As defined by the W3C, a web service is a software system designed to support interoperability between different computers on the same network. The main characteristic of a web service is to provide a software interface that other systems can interact with. The interface is operated through messages included in an envelope. These messages are usually transported via the HTTP protocol and formatted according to the standard XML, using some lightweight protocol for exchange of messages between software components. Software applications, possibly written in different programming languages, and deployed on different hardware platforms, operate data exchange and execution of complex operations either on corporate networks or the Internet, via the interfaces they expose publicly and through the use of operations they make available. Web service-based architectures generally, but not necessarily, make use of XML data representations. More efficient implementations, like the one we discuss in this paper, use JSON. In particular, we discuss a novel approach that establishes an initial library of interoperable software components, including location-based services and related interfaces, as template GUI components, and accumulates successful business processes as templates for new service classes.

2.2 Computations as data transformations

The abstract concept of function provides an important computational abstraction, since it encapsulates the type of computation and hides the details of the calculation to the user. This one only needs to know the mechanism of the function call, and not how the function works. A single function superbly represents the essential characteristics of a computation. Graphically, a function may be represented by a rectangle (box) and by its function name. In abstract terms, it is a transformation (mapping) between the set of possible data (input) and the set of possible outcomes (output). In our visual programming approach, we only use two symbols, possibly instantiated in a set of specialized icons, to represent (at variable levels of detail) programs and data, respectively. Our computing environment supports higher-level functions, i.e. services (programs) that accept other services in input and/or that produce other services. All functions in this environment are unary, i.e. accept only one input and provide only one output. Were this condition is not satisfied, either data containers or curried higher-order functions are used. The first case is resolved visually with a single rounded rectangle that hides the detail, the second one with a composition of partial functions that are associated in a bijective manner to data components [9]. The whole service can be viewed as a stateless data-flow graph with pipelined data exchanges. In particular, we produce computable diagrams, which may be directly executed for debugging, or exported as concurrent processes, or encapsulated and abstracted as a new higher-level service component. Each message in each executable diagram can be stepwise graphically visualized on request.

2.3 A righteous mixup of web technologies

In this section we shortly discuss the selected set of web technologies that our visual programming approach is based upon. They include prototype-based functional programming with Javascript, non-blocking Ajax methods, and Json-based data representations.

Client-side scripting via Javascript The JavaScript language is used to gain access to programmable objects in a client computational environment, typically a web browser. It is characterized as an object-oriented scripting language, dynamically typed and prototype-based. Javascript is also considered a first-class functional language, since it contains both closures and higher-order functions. From this viewpoint, it is quite similar to Scheme, even if using a C-like systax. Javascript is maily used as a client-side language implemented within a web browser, in order to offer powered user interfaces and access to dynamic web sites. The main principles of the language design are derived from Scheme, like lexical closures, i.e. namespaces of local symbols, and lambda functions, i.e. anonymous functions, to be used as input/output values to/from other functions, as well as from the Self language (prototype-based programming).

Prototype-based programming Prototype-based programming is an object-oriented style without classes, where the reuse of behaviors—known as inheritance in object-orientation based on classes—is obtained through a cloning process of existing objects, that behave like prototypes, i.e. as archetypal examples. This programming model is also known as class-less or prototype-oriented, and also as instance-based programming. The model contains two methods to build new objects: a literal form for “ex nihilo” creation, and another one via the cloning of existing objects, allowing for the addition of new properties non present in the prototype. This programming paradigm looks as the best one in order to accommodate the development of user interfaces that are both flexible and sophisticated.

AJAX methods AJAX (standing for Asynchronous JavaScript and XML) is a group of technologies for web development, used client-side to create interactive web applications through non-blocking server calls using JavaScript. By using AJAX, web applications asynchronously receive data that are sent in background without interfering with the display and the behavior of the active pages. Furthermore, AJAX uses a combination of HTML and CSS to define the presentation style of the information. The DOM model of the web page is accessed via JavaScript to dynamically display the page, and to allow the user to interact with the information thereof. For this purpose, JavaScript and XMLHttpRequest supply asynchronous data exchange methods with the server, to avoid the browser to refresh the whole page. Recently, JSON (JavaScript Object Notation) and JavaScript are more and more utilized as an alternate efficient format for data exchange and as data manipulation language, respectively. In particular, the jQuery library supports a complete suite of AJAX methods. Objects and methods contained thereof may open the browser multiple asynchronous data channels to the server without requiring a refreshing of the current page.

JSON data representation and exchange JSON is a lightweight text format for data exchange. Since it is text-based, the data format is easy to read and to write for humans, and possesses a regular syntax that is easy to parse automatically. Even more, JSON is exactly a subset of JavaScript, whose parsing is even easier than the parsing of XML. JSON code is a literal representation of Javascript arrays and objects, that can be nested at will. In particular, JSON is built over efficient access to two universal data structures that are virtually supported, in one form or in another, by all modern programming languages:

1. *Collection of name/value pairs*: in other languages, it is implemented as an object, record, struct, dictionary, hash table, keyed list, or associative array.
2. *Ordered list of values*: in most languages it is implemented as array, vector, list, or sequence.

Such two fundamental structures can be mutually nested in any way, therefore allowing to easily represent any type of data structure.

HTML5 Canvas The HTML5 standard provides advanced functionalities demanded by novel Rich Internet Applications (RIA). In particular, the *canvas* object yields raster visualization of any complexity and precision, that previously was provided only by vector web graphics based on SVG (Scalable Vector Graphics), that needed a suitable plugin in the browser. The HTML5 document defines the <canvas> element as "a resolution-dependent bitmap canvas which can be used for rendering graphs, game graphics, or other visual images on the fly." In other words, a canvas is a page rectangle were the application can draw by using JavaScript. HTML5 provides a set of functions (canvas API) to draw geometric shapes, define open or closed vectorial paths, create color gradients, and apply geometric transformations and image filters.

WebGL: advanced web graphics The embedded JavaScript code writes the canvas DOM elements with drawing functions that may produce both 2D and 3D interactive graphics, even generated via the hardware support of the GPU, if present, using the webGL framework. WebGL is a multi-platform and multi-vendor standard API for low-level 3D web graphics based on OpenGL ES 2.0 (the version for mobile devices), exposed through HTML5 Canvas as elements of the DOM inteface. Several libraries are currently being developed to allow an higher-level employment of the amazing graphics power exposed via WebGL canvas and the Javascript language directly in the web page by the browser, and without the use of any specialized plugin.

3 User Interaction Methods

We distinguish here between two main classes of user-interaction methods, namely the *development style*, committed to the interactive generation of executable diagrams and user interfaces, and the *navigation style*, used for navigation of information diagrams and for exploration of the hierarchical structure of basic and complex widgets, as well as for the interactive exploration of the logic of rich internet applications. The development style, or *2D visual programming style*,

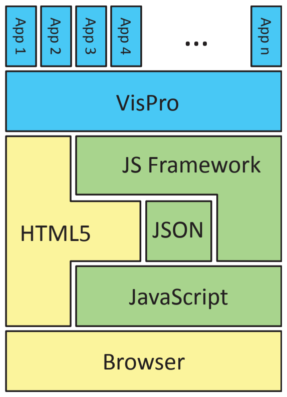


Fig. 2. A stack metamodel of the VisPro visual programming environment.

is based on drag-and-drop interaction, icon generation, input-output of widgets from a structured library of programmable objects. The navigation style, or *3D visual documentation style*, presents the user with a dynamic view of the hierarchical structure of a rich internet application. The 3D style can be used for purposes of interactive software documentation and for training of developers of RIAs and UIs.

3.1 Visual Programming (2D)

The widgets (or controls) of the GUI (Graphical User Interface) are defined as reusable elements of the graphical user interface. They offer a pleasant layout of the information and provide the user standardized techniques for data manipulation. In particular, a widget is a GUI element that produces some data input/output in a way more or less modifiable by the user; for example just consider a GUI window or a text box.

The main characteristic of a widget is to give the user a single interaction point for the direct manipulation of a specific type of datum. In other words, the widgets of the user interface are single visual building blocks that, when suitably combined into an application, may visualize both the data elaborated by the application and, in variable measure, the relations between them and the application logic. Of course, a *web widget* is a software control for the web. In practice, it is a small application that can be installed within a web page and can be executed directly by the final user. A web control is often a stand-alone application that can be embedded from third-party sites by a user with suitable write-access to somewhere (e.g. to a web page, a blog, the user profile in some multimedial social site, etc.).

The VisPro environment uses a very simple approach to service composition: (a) the developer takes a set of web widgets from a library, (b) builds interactively a user interface by drag-and-drop operations, (c) builds the application logic of the web service by drawing the connections between boxes (standing for suitable data transformations) and widgets (standing for user interaction). The development session produces, in presentation mode, a web page where the user may trigger, and interact with, the novel data mining and related computation. A successful interface (and logic) is abstracted as a new web service, characterized by a new widget, and stored in the widget library.

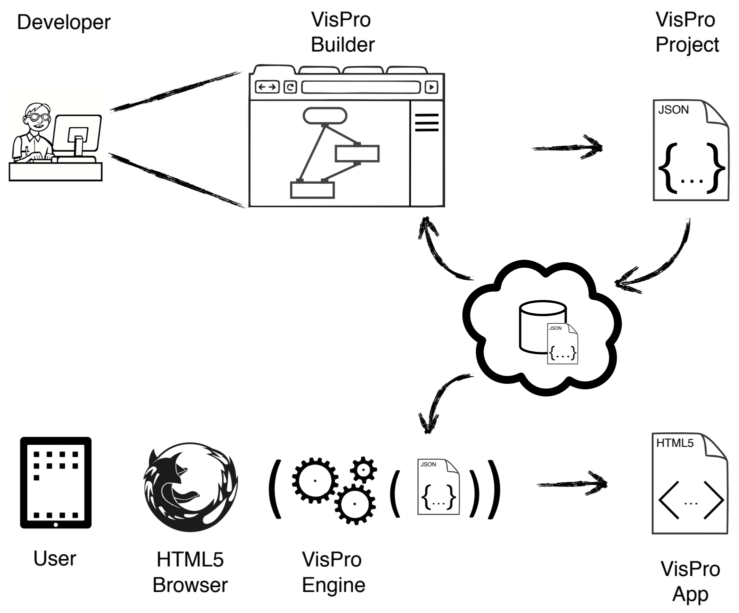


Fig. 3. The process of producing and using visual programming applications. The repository of widgets (one-to-one with data services) plays the central role.

4 The VisPro Architecture

4.1 Terminology

Some preliminary definitions may be useful to compactly set out the main characteristics of the visual programming environment VisPro discussed here.

1. *logical workspace (LW)*: work area where to define the application logic;
2. *visual normalized space (VN)*: work area where to define the UI;
3. *databox*: basic element of logical workspace that represents a data object;
4. *funcbox*: basic element of logical workspace that represents a function (service or data transformation);
5. *inlet*: connection area for input links to a box (databox or funcbox);
6. *outlet*: connection area for output links from a box (databox or funcbox);
7. *link*: connection element between the outlet of a box and the inlet of another.

4.2 Workspace layout

According with the well-known design rule of information systems, that requires a strong separation between the application logic and the application presentation, the whole VisPro workspace is split into two adjacent areas denoted as LW and VN, respectively, that become alternating when the tool will be ported to mobile devices. The LW is responsible for the application logic definition, whereas the VN is in charge for the visual definition of the user interface of the new service, including the layout of the presentation and the positioning and dimensioning of the component widgets.

A service layout typically possesses a hierarchical structure, being composed by widgets properly instanced and arranged by means of affine transformations (translations, rotations, scalings) within the Visual Normalized space. Since each widget may in turn be composed by other widgets, the hierarchical structure of the VN content is represented as a directed acyclic multigraph (we refer to it as the *presentation graph* of the service)

The VisPro interaction and building tools (single and multiple selection, transformation handles, etc.) allow the user to work in both the layout zones (LW and VN) at any chosen hierarchical levels, by selecting groups of widgets, as well as by using a stratified organization of elements by layers. Each object in the VN correspond to an object in the LW. The converse is not true. Some elements of the LW may have no mapping to elements in the user interface.

4.3 Computational model

As already discussed in the Introduction section, the computational model underlying VisPro is the data flow model. Some properties neeed to be introduced to correctly represent a data flow graph.

Assumptions In particular, we postulate the following assumptions:

1. a databox represents a datum and is provided with only one outlet;
2. a funcbox represents a function and is provided with one or more inlets and only one outlet;
3. to each inlet can be connected only one link;
4. multiple links can be conversely connected to the same outlet.

Our visual programming is based on two main interaction operations:

1. the *insertion* of a box (funcbox or databox);
2. the *linking* of the outlet of a box to the inlet of another one.

Computational network A funcbox produces an output dataflow by applying the function it represents to its input dataflows. The computation of a funcbox is only triggered when at least one datum is available for each funcbox inlet. The triggering of the function computation over a tuple of input data is sparked when the last needed datum is received by the funcbox. The triggering condition, together with the constraint concerning the existence of only one link for each inlet, allow to prove the nonexistence of flow loops in the computational network.

The set of links determines a direct acyclic graph of the computation. The whole computation is modeled by a modified DFS (Depth First Search) traversal that requires that all the inputs necessary for the trigging of each single computation are available. The traversal ordering is established by visiting, in connection order, the first node with all input available.

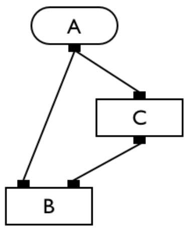


Fig. 4. Example of simple computational network

Example Let the output of node A in Figure 4 be connected with an input of B (requiring two inputs) and with C (single input). Let the C output linked with the other input of B. At first, when the computation is started from A, the triggering of the B computation is attempted. But the tentative cannot proceed because the second datum of B is missing. Therefore the A node starts the C computation. When halted, C will attempt the triggering of B. This time the try has success because the B inputs are all available.

4.4 Process communication

The linking of two boxes is always directed from an outlet towards an inlet, and no vice versa. In particular an inlet is connected to an outlet if interested to the flow of data passing through it. The action of connecting can be seen as a registration at the notification of an event:

1. the outlet is the event publisher;
2. the inlet is the event subscriber;
3. the release of a datum by the outlet is the triggering event;

when a databox releases an instance of its datum, or when a funcbox releases an instance of its output, the outlet generates the event associated to the release of the datum, and notifies all the inlets interested to such datum.

Process registration and notification Let us recall that within the JavaScript language the functions are first-class objects, and the scope is lexical, so that the functions have access to the context they are created within. This amazing language feature allow one to exploit the callback functions as connection apparatus.

In particular, the registration consists in passing a callback function from the inlet to the outlet. Conversely, the notification consists in the execution by the outlet of all the callback functions registered thereof. Also, the callback function employed in the registration maintains a full access to the context where it was created (i.e. to its closure) so transmitting its own knowledge to its publisher, that does not use a copy of it, but the function itself.

Triggering the funcbox computation Each inlet has an associated callback function. The execution of the callback by the outlet provides the flow of the datum from the outlet to the corresponding inlet. Therefore, when a funcbox receives an input datum (via the inlet), verifies that all the other inputs are present and, if this predicate is true, applies to them the function associated to it (that we can call also box *behavior*). When the execution terminates, or when an item of the output stream has been generated, the funcbox will execute the callback functions registered on its outlet.

Data caching Let us consider a funcbox B with two inlets i_1 and i_2. Let a datum be arrived in i_1 but not in i_2. The B computation cannot start, of course. The arrival of a new datum in i_1 would imply the loss of the previous one. Therefore, each inlet is provided of a queue temporary storing the transient dataflow. When a computation is triggered the needed data are retrieved from the queues. No data loss results from this very simple caching approach.

4.5 Service Security

In order to provide the VisPro users with a secure and reliable development platform, the project implements a standard two-phases authentication facility [12]. Therefore, in the VisPro environment, each client ensures the authenticity of the server, and enforces a secure communication, by connecting through an SSL/TLS [13,2] channel. User authentication is performed comparing hashed versions of the secret passphrase entered by the user, with a calculated one on the server side. Hashing functions on both sides are chosen within the Secure Hashing Algorithm family [6,5], that provides a high security level with respect to current cryptanalysis evaluations [8,4].

An additional time-based control is performed, inspired by the Kerberos protocol [10], in order to ensure the authenticity of an authentication sequence within a man-in-the-middle attack scenario: in this case, a challenge-response sequence requiring a long timeframe is a strong indication that the original message has been diverted, and the communication between the server and the (supposed) client cannot be considered secure.

Once the identity of a user is established, it is associated with one or more group memberships, mimicking standard POSIX user-group links. Additionally, each object possesses an associated access control list (ACL), reflecting standard POSIX.1e capabilities [11], allowing the VisPro users to read, and modify, each object in the visual programming platform with an object-access granularity.

5 Conclusion

The VisPro programming environment is currently under development using webGL, HTML5 and a set of Javascript frameworks. We are going in few months to make the first experiments of localization intelligence according to the strong initial requirements of this project.

References

1. Agarwal, V., Dasgupta, K., Karnik, N., Kumar, A., Kundu, A., Mittal, S., and Srivastava, B. A service creation environment based on end to end composition of web services. In *WWW 05: Proceedings of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), ACM, pp. 128–137.
2. Allen, C. e. a. The transport layer security (TLS) protocol. RFC 5246, August 2008.
3. Curbera, F., Duftler, M., Khalaf, R., and Lovell, D. Bite: Workflow composition for the web. In *ICSOC-07: Proceedings of the 5th international conference on Service-Oriented Computing* ((Berlin, Berlin, Heidelberg, 2007, 2007), Springer-Verlag, pp. 94–106.
4. De Cannière, C., and Rechberger, C. Finding SHA-1 characteristics: General results and applications. In *Advances in Cryptology ASIACRYPT 2006*, X. Lai and K. Chen, Eds., vol. 4284 of *Lecture Notes in Computer Science*. Springer, 2006, pp. 1–20.
5. Eastlake, D. E., and Hansen, T. US secure hash algorithms (SHA and HMAC-SHA). RFC 4634, July 2006.
6. Eastlake, D. E., and Jones, P. E. US secure hash algorithm 1 (SHA1). RFC 3174, September 2001.
7. Gao, M., and Wu, Z. Epn-based web service composition approach. In *WISM-09: Proceedings of the International Conference on Web Information Systems and Mining* (Berlin, Heidelberg, 2009, 2009), Springer-Verlag, pp. 345–354.
8. Matusiewicz, K., Pieprzyk, J., Pramstaller, N., Rechberger, C., and Rijmen, V. Analysis of simplified variants of SHA-256. In *Western European Workshop on Research in Cryptology* (2005).
9. Milicchio, F., Bertoli, C., , and Paoluzzi, A. A visual approach to geometric programming. *Computer-Aided Design and Applications 2*, 1-4 (2005), 411–421.
10. Neuman, B. C., and Ts'o, T. Kerberos: An authentication service for computer networks. *IEEE Communications 32*, 9 (1994).
11. of Electrical, I., and Engineers, E. IEEE standards interpretations for IEEE standard portable operating system interface for computer environments. IEEE 1003.1-1988/INT, 1992.
12. Stallings, W. *Network Security Essentials, Applications and Standards.* Prentice-Hall, 2000.
13. Wagner, D., and Schneier, B. Analysis of the ssl 3.0 protocol. In *USENIX Workshop on Electronic Commerce* (1996).