Semantic Automation from Screen Capture

Omer Barkol, Ruth Bergman, Ayelet Pnueli, Sagi Schein

HP Laboratories
HPL-2009-161

Keyword(s):
Automation technology, Image analysis, Visual parsing

Abstract:
IT organizations want efficiency and are constantly attempting to reduce or eliminate manual work. Automation technology facilitates the transition from manual to automatic. However, until now, automation technology has been applied in limited settings in IT organizations. For example, automated software testing uses automation technology, but only about 5-10% of software testing is automatic. One of the reasons automation technology is not more widely adopted, is that there is a very high barrier to using it. The objects and constructs used for today's automation originate from the underlying application or computing environment, so that using automation technology is as complex as programming. We propose a new automation technology for applications with Graphical User Interfaces (GUI). Unlike existing automation technology, which uses the objects defined by the technology that was used to build the GUI, the technology we propose uses screen capture. The system described in this paper infers GUI semantics from screen images. The system is modelled after the notion of compilers for programming languages. It has a structural analysis module, which corresponds to the lexical analyzer (Lex), and a semantic analysis module, which corresponds to the parser (Yacc). Structural analysis uses image analysis techniques including segmentation and object recognition, and semantic analysis uses graph grammars and visual parsing.

External Posting Date: July 21, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: July 21, 2009 [Fulltext]

Semantic Automation from Screen Capture

Omer Barkol
HP Labs
Haifa, Israel
omer.barkol@hp.com

Ruth Bergman
HP Labs
Haifa, Israel
ruth.bergman@hp.com

Ayelet Pnueli
HP Labs
Haifa, Israel
ayelet.pnueli@hp.com

Sagi Schein
HP Labs
Haifa, Israel
sagi.schein@hp.com

ABSTRACT

IT organizations want efficiency and are constantly attempting to reduce or eliminate manual work. Automation technology facilitates the transition from manual to automatic. However, until now, automation technology has been applied in limited settings in IT organizations. For example, automated software testing uses automation technology, but only about 5-10% of software testing is automatic. One of the reasons automation technology is not more widely adopted, is that there is a very high barrier to using it. The objects and constructs used for today's automation originate from the underlying application or computing environment, so that using automation technology is as complex as programming. We propose a new automation technology for applications with Graphical User Interfaces (GUI). Unlike existing automation technology, which uses the objects defined by the technology that was used to build the GUI, the technology we propose uses screen capture. The system described in this paper infers GUI semantics from screen images. The system is modelled after the notion of compilers for programming languages. It has a structural analysis module, which corresponds to the lexical analyzer (Lex), and a semantic analysis module, which corresponds to the parser (Yacc). Structural analysis uses image analysis techniques including segmentation and object recognition, and semantic analysis uses graph grammars and visual parsing.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation—*Workflow management*; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems—*Grammar types, Parsing*; I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Color, Shape, Object recognition,Time-varying imagery*

Keywords

Automation technology, Image analysis, Visual parsing

1. INTRODUCTION

Consider the value of software that can observe a user of a computer and understand the user's interaction in much the same way as a person standing behind the user would understand. Such software would be able to summarize, replicate and analyze the user's activity. Such automation technology has the potential to streamline workflows, improve productivity and provide administrative information throughout IT management. One could improve UI usability based on analysis of user steps. One could improve product support based on capturing user interactions that can recreate a product issue. We could learn the steps necessary to fix a problem by capturing service engineer's interactions during problem resolution. Today, automation technology is used in a small number of use cases, such as, automatically testing IT applications and automatically repairing some well understood operational problems. Today's automation technology depends on a detailed model of the computing and development environments used by the organization. Thus, the capability of each automation tool is limited to a restricted set of environments, e.g., .NET and HTML, but not Java. In addition, this automation technology uses information that is intended for object rendering. Thus, the semantic information describing the user's intent is not available. The approach proposed in this paper aims to overcome both problems. Our automation technology uses screen capture, which is available for all GUI computing environments and applications, to record and replay user activity. Semantic information is found in the image using image analysis and visual language techniques.

It may seem that by resorting to screen capture, we have defined an unnecessarily difficult problem. We foresee several trends in IT management, that create a need for such a generic approach to automation.

Web 2.0 For automation, Web 2.0 is a disruptive technology for several reasons, including the large number of UI technologies, UI technologies with proprietary APIs such as Adob Flash®, Microsoft Silverlight®, and the dynamic nature of the applications. In such applications the semantics are encoded in JavaScript, and not available in the HTML Document Object Model (DOM).

Software As A Service (SAAS) The trend toward SAAS implies that IT organizations will have less control over

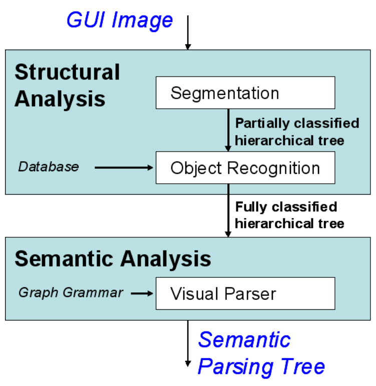


Figure 1: Block diagram of a system for inferring a semantic hierarchy from a screen image.

applications, and that the mix of UI technologies in a mash-like application will be larger.

Virtualization The trend away from a single user on a single machine toward desktop virtualization implies that a significant fraction of user interaction will only be available via image-based protocols, such as the Remote Desktop Protocol (RDP).

While analyzing the screen image is a difficult problem, we believe it is actually the best approach to understanding what the user is doing. Unlike the internal objects of the application, which were intended to be manipulated by the operating system, the UI was intended to be viewed by a person. It was designed to be easy to understand and easy to use. Furthermore, GUIs conform to rules that are familiar to anyone who has some experience using a computer. For example, a link is typically underlined, and search boxes commonly appear on the top right of a Web page. These common structures and semantics lead us to view GUIs as visual languages.

When we analyze an instance of a GUI, we want to construct a hierarchy that represents the objects on the page and their layout. This hierarchical structure should also be annotated with the semantics that the user attaches to the interface. For example, the flight reservation form within the web page shown in Figure 9. This semantic hierarchy enables us to record what the user does on the page in a meaningful way. This paper describes algorithms that, given a screen capture, computes a semantic hierarchy of the objects in the image. The approach may be divided into two major steps, which correspond to the process of compiling programming languages. The structural analysis step identifies UI layout and UI objects in the image. This step is analogous to lexical analysis which identifies tokens in the text stream along with the hierarchy of the text (paragraphs, sentences). The semantic analysis step recognizes language structures and assigns meaning to them. This step is analogous to parsing, and produces a parse tree which is annotated with semantics. The structure of the system is shown in Figure 1.

The structural analysis step has, as input, a screen image. Image analysis techniques are used to identify the UI layout and UI objects in the image. This is done using segmentation techniques, described in Section 2.1. There are many good algorithms for segmentation of natural images, including watershed [21], mean-shift [10], region growing methods [23], spectral graph methods, such as normalized cuts [20], and self-similarity methods [6]. Since GUI images are very different from natural images the segmentation approaches should be adapted. Prior work is found on layout segmentation for Web applications. In this work, the HTML DOM is assumed to be available, and there are several good solutions for extracting the layout, most notable the VIPS algorithm [9]. Our algorithm, in contrast, uses image analysis, and detects the layout of the page recursively from the outside in. The output of the segmentation stage in fed into a classification module where image segments are tagged, e.g., button, text box, radio button, etc. The first step in most classification algorithms is to represent each object in a compact manner that is also robust to certain image transformations. One effective approach, Scale Invariant Feature Transform (SIFT) [14, 15, 16] detects features that are invariant to uniform image scaling and can be generated efficiently. A different approach, shape contexts [7], tries to capture the intrinsic shape of an object. More recently, [19] proposed local descriptor which analyzes the self similarity structure of the input image. Once object representation is decided, there are many available approaches for object classification. Non-parametric methods such as nearest-neighbor algorithms utilize a database of tagged examples. [8] suggests that nearest-neighbor classifiers can be effective for visual object recognition. Unfortunately, when large databases are used, they could become too slow. In such cases, parametric approaches (e.g. Neural-Networks and support vector machines [11]) may yield superior performance. Again, the recognition problem for GUI objects is significantly different from recognition of natural objects. The objects are categorized to a small number of well known groups, which reoccur in all GUIs. On the other hand, the variability within a group is large. For examples, buttons can come in any color combination, whereas zebras are always black and white. Section 2.2 describes the algorithm for GUI object recognition.

Structural analysis results in a collection of objects which are tagged with some known attributes, such as location, type, and text value. It also gives the overall layout of the UI which is used to construct a hierarchy in a top-down fashion, as described in Section 2.1. The semantic analysis step uses this information and the spatial relationships between objects to construct the semantic hierarchy. UI constructs are identified in a bottom up fashion using Graph Grammar and parsing. Graph grammars [18] are a two-dimension (2D) extension of formal grammars [17]. One dimensional grammars describe legal strings (words) in a language, and have been used to capture semantics of programming language as well as natural language. The two dimensional extension of grammars describes diagrams, rather than strings. The current theoretic understanding about the power of graph grammars and parsing complexity is less developed compared with the understanding of string grammars. It seems obvious, in the spectrum of visual languages, that GUIs are a relatively simple diagrams. In analogy to parsing standard (1D) languages, we believe that the task of describing GUIs with graph grammars is more complex than describing strictly formal languages as programming languages, but less complex than describing loosely formal languages as natural languages. To best exploit our merit of looking at the image, we adopted the direction of Spatial Graph Grammars (SGG) [12]. For these graph grammars, the arcs hold the spatial relation between pairs of adjacent nodes. In addition, since web pages present high degree of variability, we adopt a combination of a top-down hierarchy construction based on layout and a bottom-up hierarch construction which is based on a *best-effort* parsing strategy [22]. Graph Grammars have been suggested as a mean to extract the semantics from HTML forms in the presence of a Document Object Model (DOM) [12, 22]. In that case, the layout and DOM hierarchy were used to understand the actual semantic hierarchy. However, the reconstruction of such hierarchy from the images is completely novel. Section 3 describes the visual parser.

The first target application for this work is automated software testing. This domain is a natural one for new automation technology, because there are already a number of widely used automated software testing tools on the market [1, 2, 4] which use automation technology. While the benefit of this automation technology may be even greater in other aspects of IT managements, it is more difficult to realize this benefit because those applications do not exist today. In automated testing, a semantic automation engine using screen capture is a departure from the existing technology. It can enable software testing tools that can be used to test every GUI application. This is in contrast with existing automated software testing tools that use the run-time object hierarchy of the application and, therefore, depend on a deep model of the technology used to build the test application. In fact, a number of automated testing tools already use screen capture for synchronization, but this capability is limited to very simple pattern matching of a fixed template [**?**, 3]. The proposed image analysis technique is independent of the technology used to build the test application. It improves the reliability of tests for technologies where objects are not labeled accurately (Web 2.0), or object hierarchy is not available (e.g., Adobe Flash®), or testing protocol does not enable transfer of run-time information (e.g., RDP). We point out that, in a final application, if there is partial information available about objects, e.g., from the DOM, it makes sense to combine the image analysis with this information. This paper, however, focuses on image analysis.

2. STRUCTURAL ANALYSIS

The structural analysis phase starts with a single image, I, depicting a screen capture, where $I(row, col)$ is the value of the pixel at the row row and column col. The output of this phase is a classified hierarchical tree, T_I, which includes layout hierarchy segments, $\mathcal{L} = \{L_i\}$, in its internal nodes and a set of classified objects, $\mathcal{O} = \{O_i\}$, with attributes in its leafs. Each object is defined as a triple $O_i = (R_i, C, V)$, where R_i is a rectangle containing the location of the object, class $C \in \mathcal{C}$, (the set of classes $\mathcal{C}$ includes, e.g., *button*, *text box*, *list box*, *text*, etc.), and V is a vector of auxiliary information whose contents depend on the class (e.g., object color, text value and size or other attributes). The construction of this hierarchy relies on spatial inclusion relations only — in a sense, this is analogous to the division of natural language text into paragraphs and sentences based on spaces and full stops.

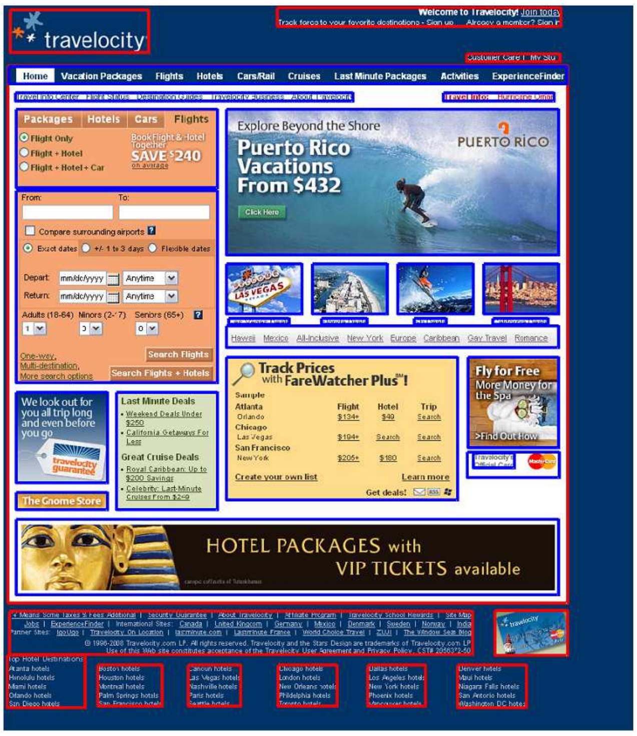
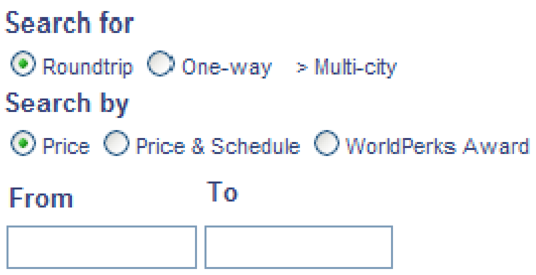


Figure 2: First (red) and second (blue) levels of the hierarchical layout.

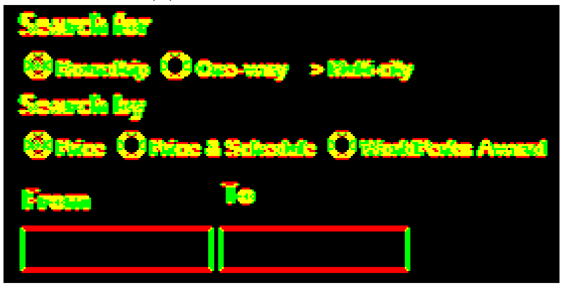
2.1 GUI Segmentation

The first task toward semantic analysis of the input GUI image I is to divide it to its components. Because GUIs are designed to make it easy to comprehend by a user, objects are easily detectable, e.g. objects have sharp edges or color transitions with limited variability. These are the visual cues used by the algorithm for segmentation approaches. We start by inferring the layout of the GUI and continue recursively level by level up to the point GUI elements are segmented out.

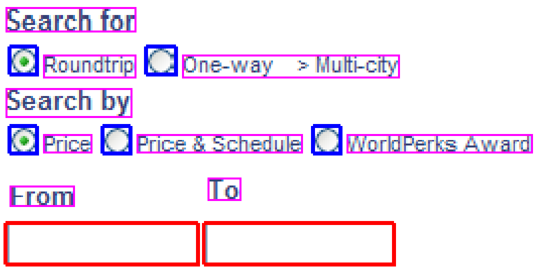
To infer the top level of the layout, the algorithm uses edge analysis on the GUI image. Edge points are detected by applying a linear line-detector filter. Then, long edges directed in the horizontal and vertical directions are considered. Pairs of parallel edges are aggregated to identify rectangles. Finally, rectangles that do not lie within any other rectangle in the GUI image are selected as candidates for the top-most level. These are added to the tree and marked with their locations $L_i \in \mathcal{L}$. Following the above procedure all the framed rectangles on the page have been identified. In Figure 2, for example, the only framed layout object at the highest level is the main content area. The remaining components at the top and the bottom of the page are grouped by a second processing step which seeks areas that contain dense visual information, and groups them into distinct layout elements. This step identifies, in Figure 2 the title and bottom navigation area. By recursively applying the above procedure, the algorithm finds the next levels of the layout. This recursion terminates when it gets to the individual GUI objects $\mathcal{O} = \{O_i\}$.



(a) Section of GUI image



(b) Filtered image



(c) Segmented image

Figure 3: (a) is a section from the GUI image, and (b) shows the directional edges: vertical edges (green), horizontal edge (red), multi-directional edge (yellow), and no edge areas (black). Image (c) shows the complete segmentation marking radio buttons (blue), text (magenta), and rectangles (red).

Edge detection effectively identifies both frames for layout analysis and object boundaries. Figure 3 illustrates the use of edge detection to detect rectangles. To infer vertical lines with one pixel width, for example, a kernel $K = (-1, 2, -1)$ is convolved with the image $(I * K)\,(row, col) = \sum_{i=1}^{3} I(row, col+i-2) \cdot K(i)$. Additional kernels are used to detect horizontal lines, and thicker lines. Figure 3(b) shows vertical and horizontal lines detected for the image in Figure 3(a) by applying linear line-detector filter, and combining the output of the filter into horizontal and vertical line segments. Based on the typical sizes of GUI objects, thresholds on the length of line segments can be reliably set and eliminate small edges. In Figure 3(b) we see pairs of vertical and horizontal line segments that are then used to define rectangular shapes in the image which often correspond to real GUI objects, shown in Figure 3(c).

There are some GUI components with a very distinct appearance, e.g, radio buttons and check buttons. For this kind of components, we apply custom detectors. The detector analyze the morphological edges in the image (a morphological edge is the difference between the maximal and the minimal pixel values in a given neighborhood, usually 3x3 or 5x5 pixels). It looks for symmetrical and small edge blobs. Circular blobs are detected as radio buttons, and square blobs are detected as check boxes.

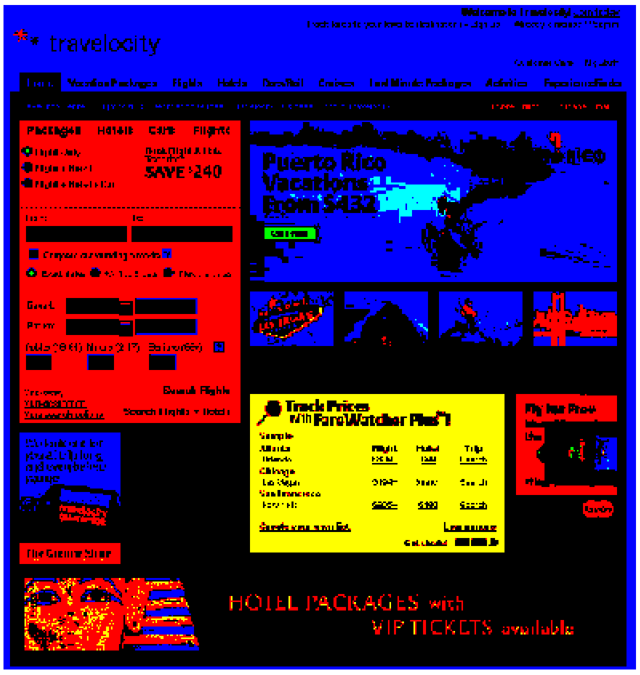

travelocity
Puerto Rico Vacations From $432
Track Prices with FareWatcher Plus
HOTEL PACKAGES with VIP TICKETS available

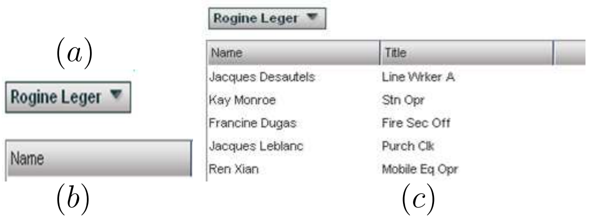
Figure 4: Primary color components of the web page.

Text segments can be highly beneficial for object classification and for the semantic grouping phase. For example, the meaning of an edit control can be more easily understood by interpreting the text to its left. Moreover, text segments might also have an adverse effect on the performance of our line detection algorithms since text segments produce many short edges with high contrast, (as can be seen in Figure 3(b)). To detect and extract text we use known Optical Character Recognition (OCR) methods.

Additionally, many objects have uniform color and high contrast with their background. Figure 4 shows the web page from Figure 2 transformed to the space of **primary colors** (red, blue, green, magenta, yellow and black). Using this observation, even in the absence of sharp edges, basic objects within the screen image can be identified. The current algorithm relies mainly on edge information, and uses the primary colors to enhance the segmentation when needed. Future work will use color information more extensively.

2.2 GUI Object Recognition

The output of the segmentation is a partially classified hierarchical tree T_I representing the GUI image, I, by a tree with non-overlapped image segments as its leafs, we call objects $\{O_i\}$, such that each segment contains at most a single GUI element. Note that part of the recognition has taken place as part of the segmentation and thus some objects, such as text, are classified. The task, in this section, is to classify each segment, O_i, into either one of the known classes, $C \in \mathcal{C}$, of GUI objects or decide that it is not a GUI object. The main challenge here is that a pair of GUI objects might look very similar but play very different roles. Figure 5 shows two GUI objects that appear very similar. Their shading seems identical and both contain text at about the same position. The only differentiating visual cue is a small triangle in the right side of (a) which identifies it as a list-box. Note that a human observer might classify object (b) as a push-button, which would be wrong in this case since it is a grid header, as seen in (c).



Name	Title
Jacques Desautels	Line Wrker A
Kay Monroe	Stn Opr
Francine Dugas	Fire Sec Off
Jacques Leblanc	Purch Clk
Ren Xian	Mobile Eq Opr

Figure 5: Two objects (a) and (b) are extracted from a complete image GUI. (c) shows a small cutout from the same GUI.

We employ a three-step approach for GUI element recognition,

1. Represent each element as a vector in a high dimensional feature space.
2. Define a metric to compare the distance between pairs of elements.
3. Search a database of tagged examples for the nearest-neighbor to classify a query object into a specific class.

Feature Vector

Selecting the proper object representation is crucial in any object recognition system. Features should be discriminative, two different objects should be represented as two different feature vectors, but they should also be robust to uninformative differences. For example, in the case of GUI objects color is usually regarded as an uninformative trait. We represent objects using three types of features which are concatenated into a single vector: (1) basic geometric data related, (2) projection related, and (3) shape related. The first part represent the geometric shape of the object, e.g. width and height. The second part of the feature vector is based on the notion of projection. This feature is illustrated in Figure 6. We start by computing the Laplacian of Gaussian $LoG(I) = \nabla G_\sigma \circ I$, where G is a zero-mean Gaussian kernel with variance σ and ∇ is the Laplacian operator. By representing the image in the LoG domain linear color changes across each GUI object are eliminated. For each object, O_i of size $n \times m$, we project the value of its pixels along the rows and columns to produce two object signatures $S_{vert}(O_i) = \left(\frac{1}{n}\sum_{y=1}^{n} LoG(I)(row, y)\right)_{row \in r(O_i)}$ and $S_{horiz}(O_i) = \left(\frac{1}{m}\sum_{x=1}^{m} LoG(I)(x, col)\right)_{col \in c(O_i)}$, where

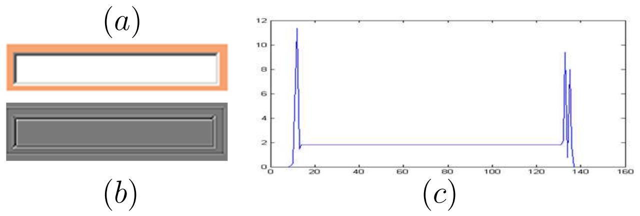


Figure 6: A one-dimensional projection-based feature vector. (a) is the original GUI element I, (b) shows $LoG(I)$, (c) shows the feature vector signature $S_{horiz}(O_i)$ which results from averaging along the columns.

$r(O_i)$ and $c(O_i)$ are the set of rows and columns O_i resides in, respectively. Figure 6 demonstrates the construction of the vertical projection signature.

It is important to note the significance of text to successful object recognition. In some cases, classification of GUI objects based on words or text patterns can be highly effective feature in others text can be misleading. If the word "OK" is found inside an object, this object is most likely an OK button. On the other hand, projection-based features tend to blur small details, such as text, from the signature, which can be beneficial when the text is less informative. The explicit handling of text-based detection is part of our future plans.

Since projection-based features can not always distinguish between certain GUI elements, we propose a third part for the feature vector, which is based on the notion of Shape Context [7]. Shape context operates on a binary edge image, currently the zero-crossings of $LoG(I)$. Given an object O_i, let $\{P_j\}$ be the set of edge points considered. We compute the set of vectors emanating from it to all other edge points, $\vec{V_j}(k) = P_k - P_j$. Finally, these vectors are represented in a log-polar domain and their histogram $h_i(j)$ is computed (two-dimensional histogram based on direction and log-partitioned distance). This histogram is denoted the shape context of object O_i relative to edge point j. $H_i = \{h_i(j)\}$ is added to the feature vector.

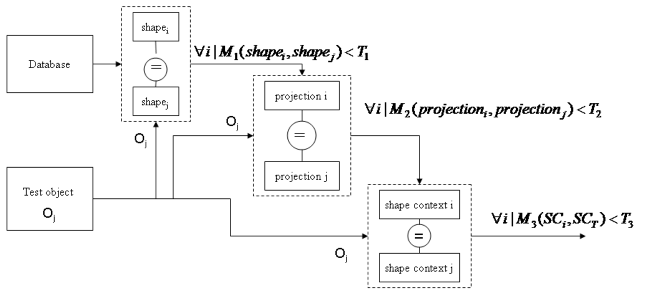
Distance Measure

In order to classify GUI objects, we need to be able to compare the distance between pairs objects. Since we have three different types of features in a single unified vector, we also need three different comparison measures. The dimension-based distance measure corresponds to the amount of deviation between the dimensions of the objects. Let O_i and O_j be two objects of dimensions $n_i \times m_i$ and $n_j \times m_j$ respectively. We define our geometric distance measure to be

$$M_1(O_i, O_j) = 1 - \frac{\min(m_i, m_j)}{\max(m_i, m_j)} \cdot \frac{\min(n_i, n_j)}{\max(n_i, n_j)},$$

note that $M_1(O_i, O_j) \in [0, 1]$.

To compare two projection-based feature vectors, $(S_{vert}(O_i), S_{horiz}(O_i))$ and $(S_{vert}(O_j), S_{horiz}(O_j))$ we compute their edit-distance [13] as the second measure $M_2(O_i, O_j)$. Edit distance is defined as the minimal number of insertions and deletions of characters that are needed to turn one string into another. We adapt a dynamic programming algorithm



Database
shape$_i$
=
shape$_j$
$\forall i \,|\, M_1(shape_i, shape_j) < T_1$
projection i
=
projection j
$\forall i \,|\, M_2(projection_i, projection_j) < T_2$
O_j
O_j
Test object
O_j
shape context i
=
shape context j
$\forall i \,|\, M_3(SC_i, SC_T) < T_3$
O_j

Figure 7: Cascade classifier for GUI object comparison. Each object O_j is yields three types of features which are compared against all the objects, O_i, in the database. The shape feature compares objects based on their dimensions.Projection-based features is computed for the surviving objects. Lastly, shape context, which is the most expensive feature, is considered to help discriminate between a small number of candidate objects.

for string alignment [13] for our case where values are not taken from a fixed alphabet but are floating point numbers. The main rational behind the use of this algorithm is that it enables this measure to be robust to non-uniform scaling of GUI objects (e.g. button can have different width and height).

For shape context, we first consider the χ^2 distance between histograms [7] (Given histograms h and h' on ℓ bins it is defined to be $\frac{1}{2}\sum_{k=1}^{\ell}\frac{[h(k)-h'(k)]^2}{h(k)+h'(k)}$). Now, for each two objects O_i and O_j we produce a best match algorithm between H_i and H_j, that is find the most similar histograms in the two sets, based on the above-mentioned distance. Finally, our shape distance function $M_3(O_i, O_j)$, is set to be the minimal distance in the best match found. (Note, this measure is different from the smallest distance between two histograms in the sets because we first apply best match).

Classification Process

To save execution time we employ a cascading approach where each distance function filters the database so that more expensive features are computed for smaller number of database objects. First the database is scanned to find all objects which are closer than a threshold T_1 relative to the shape measure M_1, then these objects are compared using the projection measure M_2 relative to a threshold T_2. Lastly, the surviving objects are compared using the shape context measure M_3 and those objects that are closer than T_3 are picked (see Figure 7 for an illustration of the process). The small set of candidates is compared to the test object, and the class $C \in \mathcal{C}$ of the database object which minimizes the weighted sum is selected as the nearest neighbor class.

Thus, finally, the hierarchical tree T_I is fully classified with each leaf labelled with a triplet $O_i = (R_i, C, V)$ where O_i is the object location, C is its class, and V includes auxiliary information, and each inner node includes a layout rectangle L_i.

3. SEMANTIC ANALYSIS

In the semantic analysis phase the classified objects, or tokens, and their layout, are used to identify the higher level constructs of the GUI visual language and to assign semantics to these constructs. The semantic analysis module, which is equivalent to the parser for standard languages, receives as inputs the classified hierarchical tree T_I, and a graph grammar $\mathcal{G}$. This section describes the three components of the parsing the input to be parsed, namely the host graph, the grammar, and the parsing algorithm.

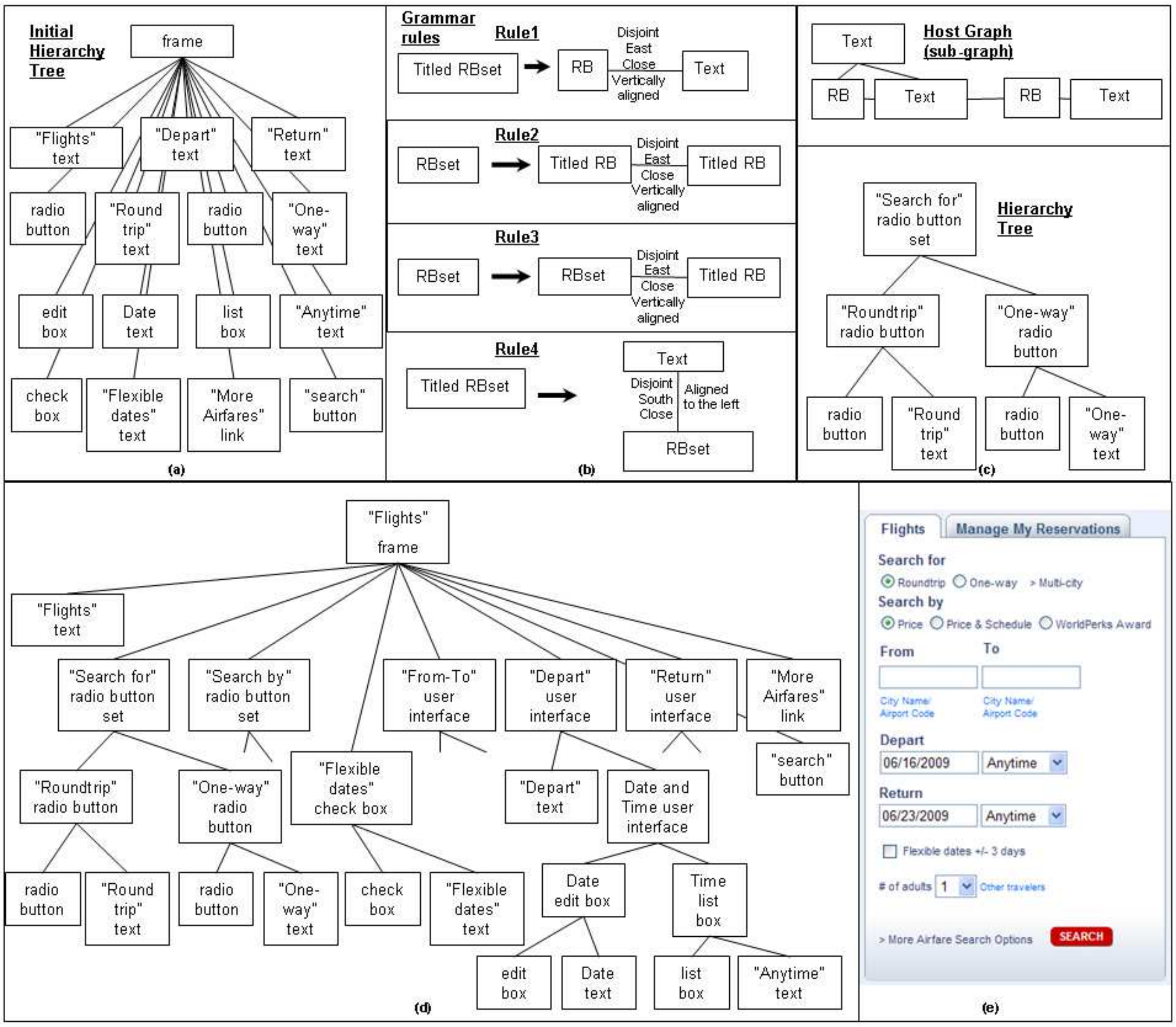
The Host Graph

The host graph is constructed from the layout components, $\{L_i\}$, and the GUI tokens $\{O_i\}$, in the input tree T_I using the spatial information. We define the host graph $\mathcal{H}_I(\mathcal{V}, \mathcal{E})$, with nodes $\mathcal{V} = \mathcal{O} \cup \mathcal{L}$, and labelled arcs. The arcs can represent the spatial relation, and other relations of interest, such as the relationship between the objects in the DOM. For example, two adjacent radio buttons will have an arc between them specifying their spatial relation in the labelling (close, North-South direction, aligned on the horizontal axes and with no other object in between them) and indicating there are at the same level in the DOM hierarchy, if applicable. In general, every two nodes $v_i, v_j \in \mathcal{V}$ have some spatial relation and thus have an arc between them. To reduce the complexity, we prune the graph to include only arcs that might be significant in the parsing stage. In particular, each node keeps an arc to its closest neighbor in each direction. For some applications, it may be beneficial to retain more of the arcs.

The Grammar

The graph grammar, unlike the tree input, can be designed off-line. Similar to standard grammars, it is defined as a tuple $\mathcal{G} = (\mathcal{C}, \mathcal{N}, N, \mathcal{R})$, where $\mathcal{C}$ is the set of object classes acting as the set of terminals, $\mathcal{N}$ is a set of non-terminals, $N \in \mathcal{N}$ is a starting non-terminal and $\mathcal{R}$ is a set of production rules. Such a grammar can be based on a general set of production rules for general classes of forms and can be manually augmented with per-application grammar rules. Figure 8(b) shows a simple example of grammar productions. A production rule consist of a left and a right side. Generally speaking, the right side specifies a sub-graph that should be found in the host graph, and the left side specifies the sub-graph that should replace it. In this work, we concentrated on rules where the left side has a single node. In addition to the sub-graph replacement, each rule might hold some action code. This allows maintaining the spatial information and also extracting the meaning of a form-object in terms of the business process. E.g., a title of a certain component of a form might indicate its purpose.

The construction of a set of grammar productions that captures the semantics of GUIs is a complicated task. There are many of reoccurring constructs in GUIs, e.g. sets of radio buttons, that can be described using pre-written set of production. On the other hand, many other constructs have too many variants to be predicted in advance. Yet, they may obey a more general form such as a list of complex elements. For example, for the interface in Figure 8(c), one could define titled date-and-time construct as a list element. In this case, the **Depart** and **Return** parts would be combined together as list of dates and times by a list template.



Search for: [✓ Roundtrip] One-way > Multi-city

Search by: [✓ Price] Price & Schedule WorldPerks Award

Depart: 06/16/2009 Anytime

Return: 06/23/2009 Anytime

Flexible dates +/- 3 days

# of adults: 1

Figure 8: Example of the parsing procedure on a frame in a web form (e). Part of the classified hierarchy tree T_I given as the initial input to the parser is shown in (a) where all tokens within the frame are siblings at first. (b) and (c) are concentrating on the parsing of the "Search for" radio button set. The relevant sub-graph of the host graph that is built out of the input is shown in (c) at the top. Then, Rule 1 of (b) is applied on the radio button and the "Roundtrip" text to its left based on their spatial relationship as implies from the host graph (c). Same is done with the radio button to their right. Later they are combined together using Rule 2, and finally combined with the "Search for" title using Rule 4 to get the hierarchy appears in (c) bellow (The recursive Rule 3 is required for larger radio button sets, such as the "Search by" radio button set). The final hierarchy tree for the whole frame with semantic diffused up appears in (d). This parsing required both standard production such as radio button sets, but also special rules such as for date-and-time recognition. Note that although the "search" button was not matched to any production rule, in this example, its semantics can be understood just by the fact it resides within the "Flight" frame.

To cover these cases, we introduce templates of productions. These templates allow the parsing of general constructs such as a list. A user can then enhance the existing grammar by specifying instances of the the template. Future work will generalize this and will allow to automatically discover such sets using preprocessing of the host graph and searching for similar sub-graphs. In general, induction of production rules should enable better parsing, see [5].

The Parsing Algorithm

Given the input grammar $\mathcal{G}$, the input classified hierarchical tree T_I, and the pre-processed host graph $\mathcal{H}_I$ the parsing algorithm produces a semantic hierarchy tree also called the parsing tree PT_I. At first $PT_I = T_I$. The algorithm then performs recursive parsing on the this tree PT_I, which is traversed in a post-order manner. At each stage only the sub-graph of the host graph which relates to a node and its children (in the hierarchy tree) are considered. This way, we divide parsing of the complete graph into several parsing tasks on the smaller sub-graphs. Since backtracking might be required to properly parse the graphs, this optimization greatly improves the running time of the algorithm.

At each parsing stage, the algorithm finds a production rule in $\mathcal{R}$ that is applicable and applies it to both the host graph $\mathcal{H}_I$ and the hierarchical parsing tree PT_I. The host graph is changed by replacing the sub-graph that relates to some right-hand-side of a production rule with the left hand side of the rule. The hierarchy tree is changed by adding another level in the tree.[1]

Another issue one should be aware of is the variability of GUIs and the fact that there is not likely to be a reasonably-sized grammar that could produce every GUI. Thus, within each layout section, parsing is performed in a best-effort manner. This means that we do not assume that each element can be matched to some production rule in the grammar. Rather parsing begins at the leaves and terminates when it cannot find another rule to apply. The visual analysis that was done in the first phase to produce the initial hierarchy ensures that the location of each layout component in the hierarchy tree provides semantics. The benefit of this approach is demonstrated by parsing result for the search button in the example in Figure 8(d).

To summarize, we produce a *semantic* parse tree PT_I that is based on the initial tree T_I. While the initial tree was built top to bottom, parsing starts from the leafs, and continues to the top-most layout object for which some grammar rule applies. The parsed elements are arranged in a hierarchical manner, diffusing up the semantic of their construct to the semantics of their ancestors. In particular, the role of each segment of the layout can be recognized according the semantic of its ingredients. We currently produce semantic understanding based on titles and of types of the constructs only. Based on specific constructs in specific topic, more refined methods can be applied, e.g., inferring the "flight locations" topic of a section that includes "from" and "to" edit boxes in a flight reservation form, as appears in Figure 8.

[1]In order not to create strings, in the case of recursive rules (like Rule 3 in Figure 8), the tree might be "flattened" by adopting a new sibling instead of adding another parent.

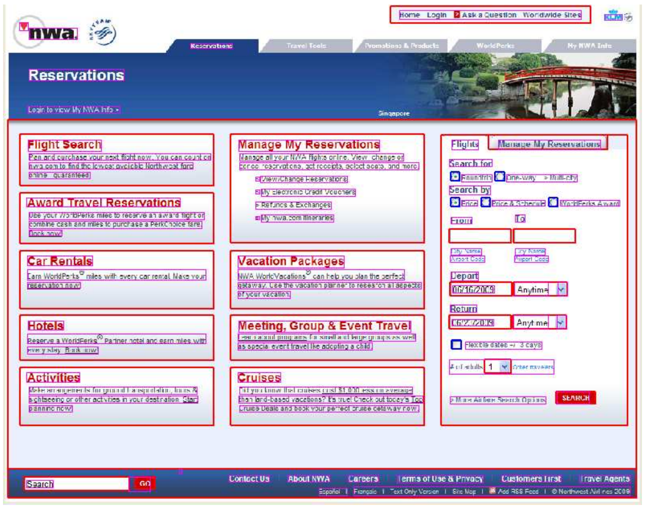


Figure 9: Results of segmentation on the Northwest Airlines Reservation Center page. Shown above are the results of the current suite of detection algorithms, including rectangles, radio buttons and text.

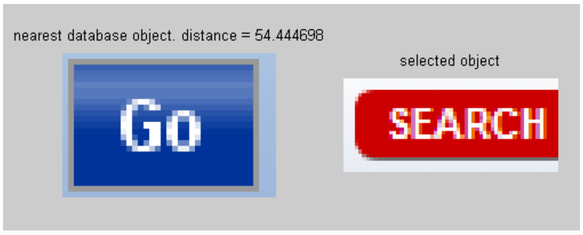


Figure 10: Results of object classification. The red "Search" button was matched to the database object on the left.

4. RESULTS

We demonstrate the feasibility of the proposed system for image-based GUI analysis with an end-to-end example. The system renders the Northwest Airlines Reservation Center page (*www.nwa.com/travel/reser/main.html*) and stores the rendered image. In the structure analysis phase, the image is first segmented using the suite of layout and object detection algorithms from Section 2.1. These include rectangle detection and radio button detection. Text detection and recognition complete this set of algorithms. At this point, we have a partially classified hierarchical tree, as depicted in Figure 9. In the figure, each object is displayed as a rectangle. Objects are coded by type, where blue indicates a *radio button* or *check box*, magenta indicates *text*, and red indicates a rectangle of *unknown* type.

Next, the GUI object recognition algorithm from Section 2.2 classifies each of the rectangles marked in red in Figure 9. We created a database of classified objects which included a variety of GUI objects, including GUI objects with a typical web look and feel, as well as some objects taken from other pages from the Northwest Airline web application. On this page all the text boxes, list boxes and buttons were classified correctly. Figure 10 shows the nearest match found for the "Search" button.

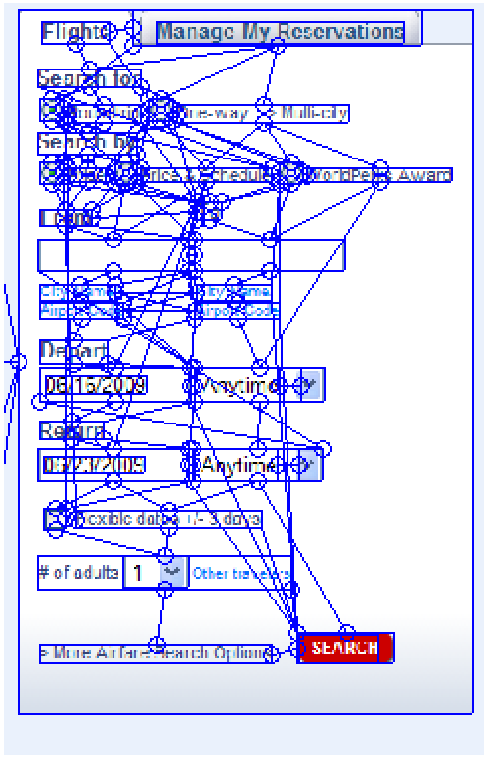


Figure 11: A section of the host graph for the objects identified on the Northwest Airlines Reservation Center page.

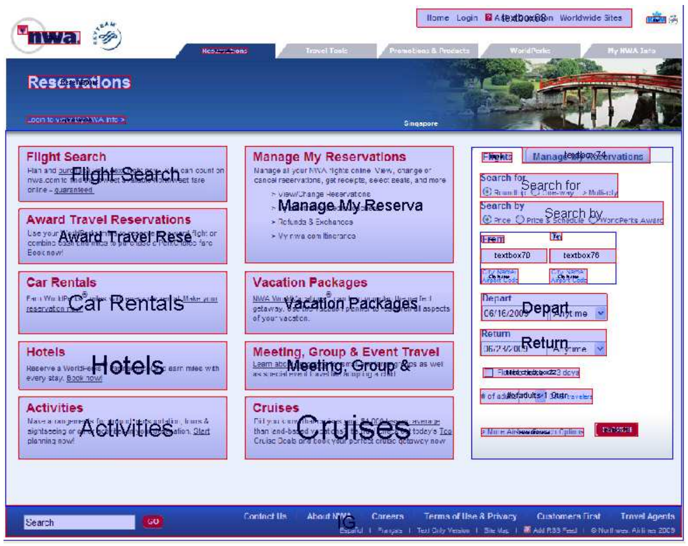


Figure 12: Results of parsing the Northwest Airlines Reservation Center page. This image shows a view of the parse tree. In this view some nodes are in the middle of the tree, e.g., the "Flight Search" and "Manage My Reservation" sections. Other nodes are leaves, such as the components making up the "From" and "To" user input area.

Structural analysis is now complete, and we have a fully classified hierarchical tree which includes set of layout segments and objects: their location on the page, with associated type, text value and other attributes. With this input we begin the semantic analysis phase, using the visual parser from Section 3. First the host graph is constructed based on the spatial relationships of the objects. Note that out of the potential complete graph, many arcs where eliminated, such as arcs between objects inside the frame to objects outside the frame.

The grammar used in this example includes about 20 production rules with 8 terminal objects and 10 non-terminals. Most of the rules are generic visual rules, like the rules described in Figure 8 for radio buttons or rules for templates such as vertical and horizontal sets of elements. We defined 2 template assignments for the Northwest Airlines web application, including a template assignment for the date-and-time elements which form a vertical set ("Depart" and "Return"). Figure 12 shows one view of the parse tree for this page. In the figure, nodes are shown as filled blue rectangles. Semantic information that was extracted from the image by the parsing process is shown as text on the node. Note that each of the navigation areas, such as the "Flight Search" and "Manage My Reservation" sections, are identified with an associated semantic tag. These nodes are internal nodes in the parse tree, meaning that we would have to traverse down the tree to get to UI component. Other nodes in the view shown in Figure 12 are leaf nodes, e.g., the components of the "from" and "to" user input area. These nodes correspond directly to a UI component. Overall, the grouping of objects in the parse tree, and the semantic tags assigned to the internal nodes correspond very well to the way people perceive this page. Thus, the parse tree constitutes a semantic hierarchy of the page.

5. SUMMARY AND FUTURE WORK

This paper presented a system for inferring GUI semantics from screen images. The system is modelled after the notion of programming language compilers. The structural analysis module corresponds to the lexical analyzer (Lex) and the semantic analysis module corresponds to the parser (Yacc). Structural analysis uses image analysis tools and techniques, whereas semantic analysis uses graph grammars and visual parsing. This paper describes a set of algorithms for each task, and demonstrates feasibility of the proposed algorithms for the task at hand.

We identify future work for each task. In the structural analysis module we are considering more robust segmentation schemes based on the notion of self-similarity and more powerful object classification algorithms, e.g., Support Vector Machines (SVM). In addition, we do not believe we can use a single database for all web applications. Rather, for each application, we can use some common components, and some components that have been modelled from the application under analysis. We are building a tool that assists a user of the system to efficiently and effectively create a classification database for any application.

We also do not believe that there is a single graph grammar for all web applications. We plan to explore several directions to simplify the creation of a grammar. We already use grammar templates, and we are building a tool that helps a user construct template rules. We plan to explore the possibility of inferring template rules from a single page based on repetition, alignment and rhythm. We also have a more ambitious goal to infer grammar rules from web crawling.

In addition to improving each component of the system, future work includes creating an automation engine that uses this vision system as the basis for recording and replaying automation scripts.

6. REFERENCES

[1] HP loadrunner and quick test professional. http://www.hp.com.

[2] Rational functional tester. http://www.ibm.com/us/en/.

[3] Tevron®citratest vu®. http://www.tevron.com/citratest-vu.asp.

[4] Tevron®citratest®. http://www.tevron.com/citratest.asp.

[5] K. Ates and K. Zhang. Constructing veggie: Machine learning for context-sensitive graph grammars. In *ICTAI '07: Proc. of the 19th IEEE International Conference on Tools with Artificial Intelligence*, pages 456–463, Washington, DC, USA, 2007. IEEE Computer Society.

[6] S. Bagon, O. Boiman, and M. Irani. What is a good image segment? a unified approach to segment extraction. In *ECCV '08: Proceedings of the 10th European Conference on Computer Vision*, pages 30–44, Berlin, Heidelberg, 2008. Springer-Verlag.

[7] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:509–522, 2002.

[8] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, Anchorage, Alaska, June 2008.

[9] D. Cai, S. Yu, J. rong Wen, and W. ying Ma. Extracting content structure for web pages based on visual representation. In *Proc.5 th Asia Pacific Web Conference*, pages 406–417, 2003.

[10] D. Comaniciu, P. Meer, and S. Member. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24:603–619, 2002.

[11] H. Drucker, C. J. Burges, L. Kaufman, C. J. C, B. L. Kaufman, A. Smola, and V. Vapnik. Support vector regression machines, 1996.

[12] J. Kong, K. Zhang, and X. Zeng. Spatial graph grammars for graphical user interfaces. *ACM Trans. Computer-Human Interaction*, 13(2):268–307, 2006.

[13] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845–848, 1965.

[14] D. G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, pages 1150–1157, 1999.

[15] D. G. Lowe. Distinctive image features from scale-invariant keypoints. In *International Journal of Computer Vision*, volume 60, pages 91–110, 2004.

[16] K. Mikolajczyk and C. Schmid. A performance evaluation of local discriptor. *IEEE transactions on PAMI*, 27(10):1615–1630, October 2005.

[17] G. Rozenberg, editor. *Handbook of graph grammars and computing by graph transformation.* World Scientific Publishing Co., Inc., NJ, USA, 1997.

[18] G. Rozenberg and A. Salomaa, editors. *Handbook of Formal Language Theory*, volume 1-3. Springer Verlag, Berlin, Germany, 1997.

[19] E. Shechtman and M. Irani. Matching local self-similarities across images and videos. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2007.

[20] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 2000.

[21] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE PAMI, 1991*, 13(6):583–598, 1991.

[22] Z. Zhang, B. He, and K. Chang. Understanding web query interfaces: best-effort parsing with hidden syntax. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 107–118, New York, NY, USA, 2004. ACM.

[23] S. Zhu, T. Lee, and A. Yuille. Region competition: unifying snakes, region growing and mdl for image segmentation. *ICCV*, 24:416–425, 1995.