A Context-Aware Decision Engine for Content Adaptation

Building a good content adaptation service for mobile devices poses many challenges. To meet these challenges, this quality-of-service–aware decision engine automatically negotiates for the appropriate adaptation decision for synthesizing an optimal content version.

Wai Yip Lum and Francis C.M. Lau
University of Hong Kong

Mobile-device users often wish that they could access the variety of rich hypermedia content that exists or will exist on the Web. Given, however, these devices' constrained computational and rendering power and cellular networks' limited bandwidth, effective Web content presentation will require new computation patterns. The mismatch between rich multimedia content and constrained client capability presents a research challenge.

Mobile devices' variety also increases the difficulty of accessing content. For example, mobile devices are conveniently sized to fit in a pocket, but this size constrains their display area.[1] Creating trimmed versions of content could get around this constraint, but differences in display capabilities would easily make a device-specific authoring approach too costly to be practical.[2] Examples of device differences include screen sizes ranging from 20×5 characters to thousands of pixels, and color depths ranging from two-line black-and-white display to full-color display.[3] Web content can also be encoded in many different modes, such as JPEG, which best suits PCs, and the 1-bit WBMP format for Wireless Application Protocol (WAP) devices. Furthermore, mobile devices support many different markup languages, including HTML, HDML (Handheld Device Markup Language), and WML (Wireless Markup Language). If the client device subscribes to a Web site that uses a presentational mode that the device cannot render, information loss might result.

While users complain about the "World Wide Wait" problem, owing partly to slow last-mile speeds, cellular networks work at far lower data rates, which work fine for plain text but are far from adequate for Web pages.[4] Much Internet content and various other types of multimedia information that mobile applications running on PDAs or notebooks use, such as for identifying locations (for example, the nearest restaurant) or describing product displays (for stock inventory), are unsuitable for smaller devices such as WAP phones.

To tackle these problems, we propose a *content adaptation system*. Such a system decides the optimal content version for presentation and the best strategy for deriving that version, and then generates that version. The system's most crucial component is the *decision engine*, which negotiates for that strategy. The engine takes into account the entire computing context (see the "Context Awareness" sidebar), focusing particularly on the user's preferences. A prototype PDF document adaptation system demonstrates our approach's viability.

Content adaptation

The process begins when someone uses a mobile device to submit a request to the system—that is, to the content provider via an intermediary proxy server (see Figure 1). After the system identifies the user, it inputs context information to the decision engine, which resides on the proxy server. On the

1536-1268/02/$17.00 © 2002 IEEE

Context awareness

A client device's characteristics and capabilities are part of the context of a client environment where Web content rendering occurs. Context includes any information that can characterize an entity's situation.[1] An entity could be a person, place, or object that is relevant to interaction between a user and an application. The user and the application themselves are such entities.

This definition makes designing the relevant context easier because we don't have to examine the context's implicit and explicit nature.[2] Unlike human–human interaction, the distinction between implicit and explicit context information (for example, nodding the head versus saying "Yes, I will drive you to the bank") is blurred or irrelevant for human–machine interaction because of the semantic gap between machines and humans. Instead, the concepts of qualitative and quantitative context information are more applicable, as we discuss in the main article.

A system is context aware if it uses contexts to provide relevant information or services to the user, where relevancy depends on the user's task.[1] This definition is more general than the ones that Richard Hull and his colleagues[3] and Jason Pascoe and his colleagues[4] provided, whereby context-aware applications must detect, interpret, and respond to contexts. Here, we consider only the interpretation of and response to contexts. We leave the detection mechanism's sensor design to other discovery systems, which we can cleanly separate from the main context-aware process.

REFERENCES

1. G.D. Abowd and A.K. Dey, "Towards a Better Understanding of Context and Context-Awareness," *Proc. 1st Int'l Symp. Handheld and Ubiquitous Computing* (HUC 99), Lecture Notes in Computer Science, no. 1707, Springer-Verlag, Heidelberg, Germany, 1999, pp. 304–307.
2. A. Schmidt, "Implicit Human Computer Interaction through Context," *Personal Technologies*, vol. 4, nos. 2–3, June 2000, pp. 191–199.
3. R. Hull, P. Neaves, and J. Bedford-Roberts, "Towards Situated Computing," *Proc. 1st Int'l Symp. Wearable Computers*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 56–63.
4. J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," *Proc. 2nd Int'l Symp. Wearable Computers*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 92–99.

basis of some scoring scheme applied to different content versions, the decision engine then executes an algorithm that computes the optimal version that is renderable with the current client device and network characteristics. "Version" at this stage means a set of desired settings such as color depth, scaling factor, and presentation format. The decision engine sends the results to the *transcoder*, which generates the desired content version. The intermediate proxy then sends the adapted content to the target device for rendering.

Contextualization

To design a good adaptation service, we must understand the environment sufficiently. We can gain such an understanding through a contextualization framework that facilitates the expression and capturing of context information.

One such framework is the Resource Description Framework–based *Composite Capability/Preference Profile* (www.w3.org/Mobile/CCPP). The CC/PP can describe the capabilities of a client device, called a *user agent*, and the user's specified preferences within the user agent's set of options. It offers a mechanism for retrieving capability and preference profiles via the Web, from hardware or software vendors. This approach reduces the amount of information that the user agent must directly send through a limited-bandwidth communication channel.

For J2ME (Java 2 Platform, Micro Edition; http://java.sun.com/j2me) developers, the *Connected Limited Device Configuration* defines a standard Java platform for small, resource-constrained, connected devices and enables the dynamic delivery of Java applications and content to those devices. The only profile currently developed for the CLDC configuration is the *Mobile Information Device Profile*.[5]

Both the CC/PP and MIDP can provide the declarative semantics needed to determine the adaptation settings for transcoding to produce the optimized content for a specified user agent. So, this article is concerned mainly with the procedural semantics.

Transcoding techniques

On a related front, considerable research has addressed techniques for different transcoding methods. Little research, however, has addressed having the decision engine provide quality of service-sensitive decisions to compensate for or minimize the losses due to transcoding. A gap seems to exist between the declarative specification of the client characteristics (such as the CC/PP) and what the various transcoding techniques can achieve. We propose a negotiation model that the decision engine would use to bridge this gap. Negotiation happens between a representation of the user preferences and a decision function based on real-time parameters.

In existing transcoding methods, operations such as compression, color depth reduction, image scaling, and so on are lossy[6] and cannot be blindly applied in all application scenarios. These operations should be managed according to strategies synthesized from all related contextual information sources.

Device- and user-specific preferences

Timothy Bickmore and Bill Schilit's research on *device-independent access* offers good insight on handling client device variability by staying away as much as possible from creating content versions specif-

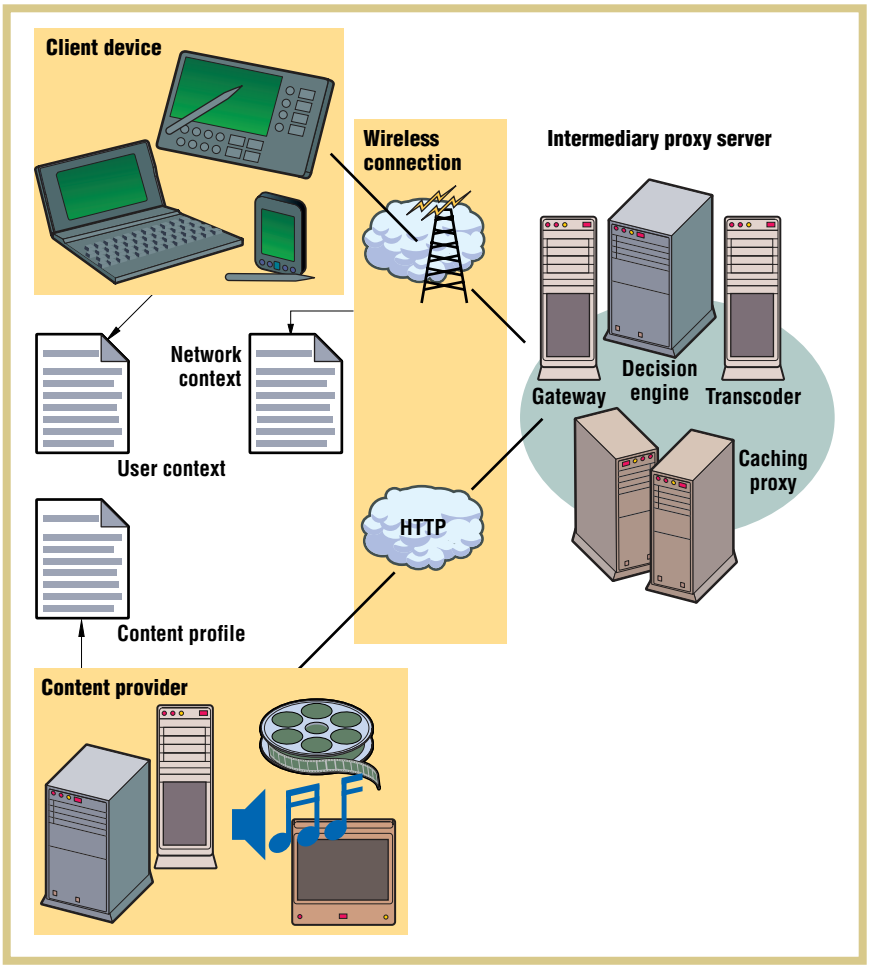

Figure 1. Content adaptation's overall structure.

ically for individual device types.[2] Armando Fox and Eric Brewer's research is in the same vein and suggests that clients generally vary along three important dimensions: network, hardware, and software.[7] The practical approach in response to such research assumes that the application scenario's basis is the client device's possible variations. In this article, we also examine client variability along the line of user perception.

Richard Han, Veronique Perret, and Mahmoud Naghshineh have discussed the concept of multiuser and multidevice browsing, focusing on the specific application scenario of content browsing in a lecture.[8] This scenario required creating two different content views because the presenter might wish to view his or her notes for each slide but prevent the audience from viewing them. Also, forward-and-backward navigation access should be limited to the presenter only. These separate views point to the need for user-specific requirements in addition to device-specific ones. Applying this to content adaptation, we can imagine creating different views of the same content for different users according to their preferences. Such a user-centric approach and the resulting content should increase user satisfaction.

Qualitative user preference and quantitative content value

Content adaptation systems' decisions can take into account numeric values associated with different content versions or transcoding strategies. Not all user preferences, however, are easily expressible in terms of numeric values in a certain context—for example, a user's perception of color. On the other hand, providing qualitative information on a user's preference for one quality domain over another would be trivial. The user merely needs to specify the preference without any exact quantification. A preference relation (for example, ranking) that connects different quality domains can describe the qualitative information.

To reduce the user's workload, assigning a numeric score to a particular content version should be automatic. To automate this assignment, researchers have proposed *resource-based content value*.[9,10] This value depends on the client device's resources that can be used to render the content. In this article, however, we follow a different direction; we base the content value on user preferences. Such a content score should lead to the best user satisfaction, because quality of service (QoS) is a user-oriented property.[11]

The decision engine

Our decision engine aims to increase users' satisfaction in their subscribing to Internet contents in a constrained mobile computing environment. The engine automatically negotiates for the appropriate content adaptation decisions that the transcoder will use to generate the optimal content version. A separate paper describes the transcoding part of the system.[12]

Because our QoS-sensitive approach compensates for transcoding's lossy nature,[6] it reduces the chance of serious loss of quality in various domains. The decision engine tries to arrive at the best trade-off for content adaptation while minimizing content degradation due to lossy transcoding. It is aware of different types of context information such as the user's preferences, the device's rendering capability, and the network connection's characteristics (see Figure 2).

The engine relies on a user's indication of preferences according to his or her perception in different quality domains. On the basis of these preferences, the engine can devise

- A method to express quantitatively any given content's quality along various quality axes
- Algorithms to negotiate for an optimal content version

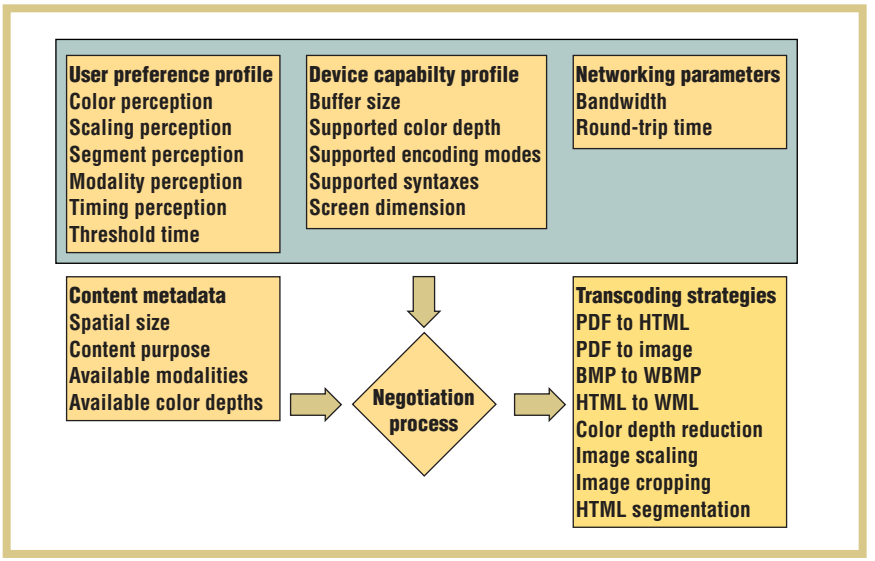

Figure 2. From contextual information to transcoding strategies.

with some guarantee on the returned objects' QoS.

The critical issue in designing a content adaptation system is how to determine a trade-off that guarantees the desired QoS by interpolating from context information, without modifying the underlying system, including the client device and the content provider. That is, we desire "zero administration"[6] on the client side. In our decision engine, *content negotiation* performs this trade-off. This process takes into account factors such as processing overhead, optimization accuracy, and context awareness.

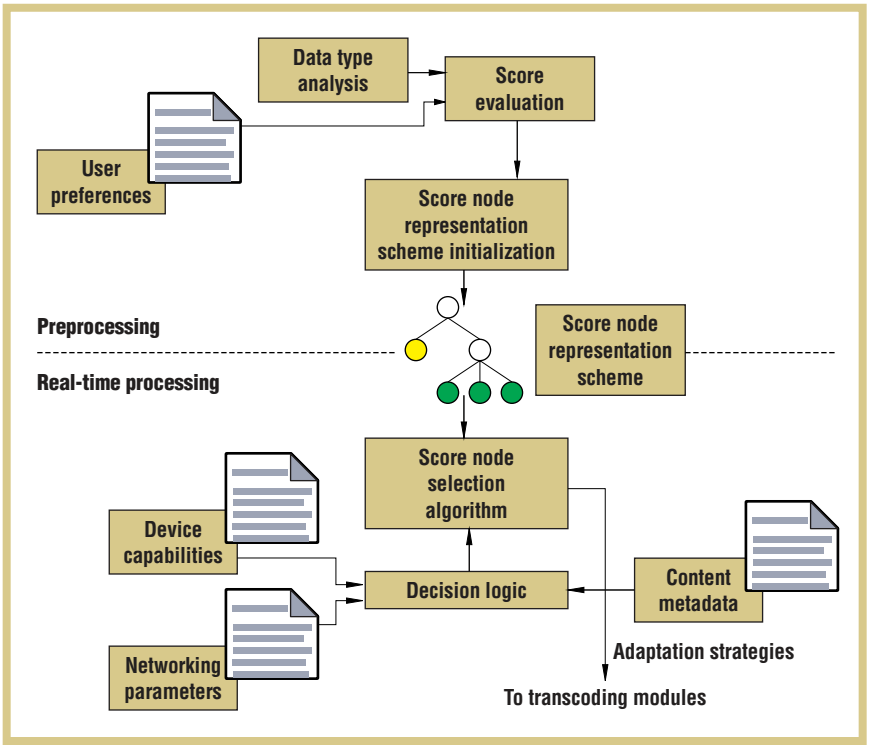

Figure 3. Content negotiation's two stages: preprocessing and real-time processing.

Content negotiation

Content negotiation has two stages: *preprocessing* and *real-time processing* (see Figure 3).

Preprocessing

This stage occurs before the user request arrives.

Data-type analysis. The set of quality axes for different types of multimedia content forms the working properties that precisely define the QoS for any multimedia content. We view the quality of a specific version of an object as a point in an *n*-dimensional space, where *n* is the number of different qualities.

For example, take the QoS of a PDF document:

$$QoS_{pdf-document} = (\text{color, scaling, segment, downloading-time, modality})$$

Modality addresses the change in the content's presentation scheme to render the content in diverse devices (see Figure 4). For example, a handheld device could display a PDF document in its original PDF format. However, considering the cellular network's constrained bandwidth and the handheld device's limited resources, it is advisable to convert the document to a format that the device can comfortably render, such as WML, which many WAP devices support.

Quantitative analysis of quality. To facilitate the expression and automatic processing of QoS parameters, we need a quantitative approach to characterizing QoS in any axis. We define a metric based on *quality value*. Take the color quality axis as an example. We assign an 8-bit color image a larger *qv* than that of a 1-bit black-and-white image. However, expressing or measuring quantitatively the loss of *qv* in terms of colors is not as straightforward. Also, different quality axes will have different QoS characteristics; such diversity makes capturing all the relevant characteristics quantitatively a nontrivial task.

We use *first order* or *second order* modeling to monitor the change of *qv* against the quantization step *qs*. A quantization

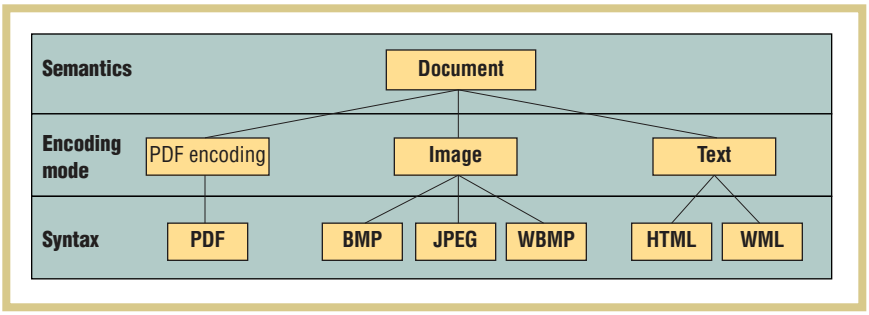

Figure 4. The concept of modality.

step is a step in a scale covering the possible settings of a particular quality axis—for example, 2, 16, 256, and so on for color.

For first-order modeling, *qv* increases or decreases linearly as *qs* increases or decreases. Second-order modeling characterizes the modeling curve as a second-order equation. This modeling applies saturation whereby *qv* will attain saturation near the far end of *qs*. At or near saturation, *qv* will insignificantly increase or decrease when *qs* changes. Consider again the color quality axis. When *qs* is at 16-bit colors (25,536 colors), adding more colors will insignificantly affect *qv*. In contrast, increments of colors near the value of 2 colors will more greatly affect the perceived quality. We expect most user preferences will exhibit such a saturation pattern. To model quality domains that do not have such a saturation behavior—for example, the modality quality axis—we can use first-order equations.

Together, these two models can capture most if not all of the behavior of the most common quality axes. If users have a strong sensitivity to a quality axis that these two models do not cover, they can input their desired modeling curves in the system.

Score evaluation and representation. Using quality axes, users can easily indicate their preferences. For example, a user might have a weak sensitivity to color but a strong sensitivity to difference in dimensional size. So, the user will rank the color quality axis lower than the scaling axis. Users can input their specific preference (through ranking) for each quality axis, and each axis's relative weight can then be calculated. An aggregate score can then be computed based on these weights.

Having the user manually assign the score to each possible version of content would be difficult and impractical. Also, using only resource utilization measures to determine the score is unreasonable.[9,10] We offer a mechanism that automatically assigns a score to a content version, taking into account user perceptions in different quality domains. The set of aggregate scores serves as the main input for the decision engine to determine the optimal version.

Scores corresponding to different versions must be stored in some organized structure to facilitate efficient searching. This structure is either a linked list or a tree where each node represents a content version and stores the corresponding score. These *score nodes* also contain the adaptation settings (the *qs*'s) for possible subsequent generation of this content version. A score node is not tied to any specific Web content; neither does it replicate any actual content from the content provider. It is generic and applicable to any content. The actual content comes into the picture only during transcoding. The same is true of the global structure containing all the score nodes.

At initialization time, the decision engine creates a search space consisting of all possible score nodes, which covers all the possible adaptation decisions that the decision engine can make. The engine computes each node's score during preprocessing when the user registers and specifies his or her preferences. This computation can occur during preprocessing because of the score–version association's static nature. That is, we expect that users will rarely change their preferences for the various quality domains. The decision engine preprocesses the established score node space into a suitable data structure such that when the adaptation system receives a request, the real-time heuristic search will be effective and efficient. The construction and initialization of this structure represents the end of preprocessing.

Real-time processing

This stage processes the user's request.

Decision logic and score node selection. A user's score nodes capture all the possible combinations of preference values in various quality domains. They do not include values of real-time parameters such as the characteristics (metadata) of the Web object being requested, the network connection's characteristics, and the device's capability. The decision logic aims to find the best scoring node corresponding to a version of the content that is renderable given those parameters. This is a negotiation process between the data structure containing the user's preference information and a decision engine's decision function (see Figure 5).

During negotiation, the *negotiation algorithm* systematically traverses the score nodes. This iterative heuristic search tries to find the most optimal score node. To locate the optimal node, the negotiation algorithm examines each score node and generates a binary decision (True or False) based on the client device's capability, the network parameters, the adaptation settings in the score node, and the content itself:

```
T || F ⟵ decision(score-node, content-metadata,
network-parameters, device-capability)
```

`Score-node` provides the settings of the quality axes. The binary decision True indicates that if the adaptation system transcodes the content according to the score node's adaptation settings, the target device will be able to render it in the current network environment. The binary decision False indicates otherwise. The decision function, `decision()`, negotiates iteratively with the score node data structure until it finds a satisfactory score node with a True decision.

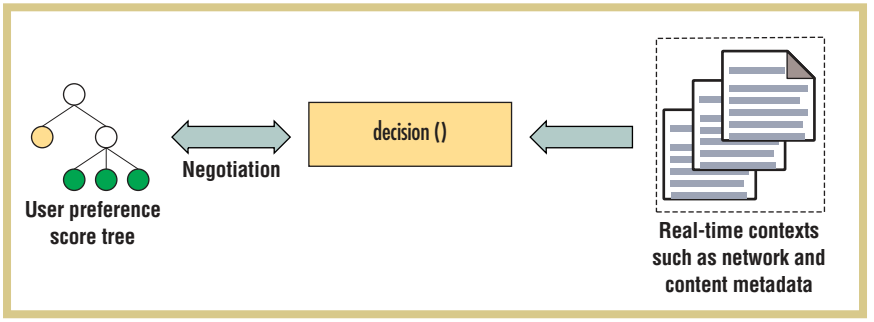

Figure 5. The negotiation process.

Ordered relations. The decision logic might operate with different strategies in different application scenarios. In some scenarios, we can identify a metric with the *ordered relation* property, on the basis of which we can make binary decisions. For example, the resource metric is based on an ordered relation. We can save some effort during searching for the optimal score node by using binary search or a binary search tree. To have the ordered-relation property, a metric must satisfy three criteria: a binary relation R on a set A is an ordered relation if and only if it is reflexive, asymmetric, and transitive. In some cases, however, identifying a metric with the ordered-relation property is difficult. We'll examine this situation in more detail in the next section.

Choosing a negotiation algorithm

To traverse the score nodes, the decision engine can use one of four negotiation algorithms.

SLL

The obvious choice of data structure for score nodes is a linked list where the elements are in descending order of scores. To determine the optimal version of content with the highest score, the *score linked list* negotiation algorithm applies the simple search where **decision()** yields either True or False at a particular node. The resultant score node is optimal in that it has the highest score among all that are feasible to render in the present context. The SLL algorithm is easy to implement and requires little housekeeping. In the worst case, however, the number of nodes to traverse is equal to the total number of nodes in the list. So, we have an $O(n)$ algorithm, where n is the number of nodes in the linked list.

Score tree

The next two negotiation algorithms use a balanced binary tree (for example, a red-black tree) to reduce the worst-case search complexity to $O(\lg n)$ or $O(\textit{tree height})$. The score tree's main advantage over the score list is reduced real-time processing overhead. Deploying the adaptation service involves frequent accesses of the data structure, so the fewer score nodes the decision engine must traverse, the better its performance. But the tree structure requires considerable housekeeping for deleting and adding score nodes. Fortunately, in adaptation applications, once the tree is initialized, it seldom requires modification. The initialization might incur some overhead, but this should not be too significant because initialization occurs only once.

ORST

The *ordered-relation score tree* negotiation algorithm works when we can identify an ordered-relation property for the decision logic. It works similarly to the classic binary tree search but with the decision function dictating tree traversal. During preprocessing, this algorithm marks each subtree with a value indicating the minimum resource that a node in that subtree requires. The decision function can decide whether to visit a subtree by comparing this value with the client device's acceptable resource level. This can lead to savings from not having to visit all the nodes.

NORST

When determining whether a metric has the ordered-relation property is difficult or impossible, we cannot model the decision logic with a comparison relation. Without the comparison logic, we must visit every node in the linked list in the worst case. Using the linked-list data structure and the linked-list traversal in this case would guarantee the best score for the adaptation, but the traversal overhead would be $O(n)$.

To achieve better efficiency, we allow a trade-off between the traversal overhead and the optimization's accuracy. The idea is a probabilistic score tree that incurs less overhead and can likely return a good, if not the best, score node for content adaptation. During preprocessing, the decision engine computes the probability (p_t) of finding a node in a subtree that would return True at the decision logic. It applies this value to every subtree (the probability multiplied by the number of nodes) so that the negotiation algorithm will know whether it should visit a subtree during runtime.

This probabilistic score tree aims to find a subtree that can most likely return True at the decision logic while giving a good score in the tree walk. Not all the nodes returned will have the highest acceptable score, but most nodes returned will have the best score.

This *nonordered relation score tree* negotiation algorithm can reduce the traversal overhead to $O(\lg n)$. However, its major shortcoming is nonoptimal accuracy. We can derive its optimization accuracy (of finding the optimal score node) and a bound on the probability that the optimal node is not returned (1 – optimization accuracy). On the basis of this derivation, we find that the optimization accuracy increases with the value of p_t. But even when p_t is small (for example, 0.3), the optimization accuracy still falls in the neighborhood of 70 percent, which is probably acceptable for many applications.

SLL-NORST

If we need guaranteed higher accuracy, we can use a mixed algorithm: the *score linked list–nonordered relation score tree* negotiation algorithm. SLL-NORST can preserve SLL's optimization accuracy while exploiting NORST's reduced overhead over a series of requests.

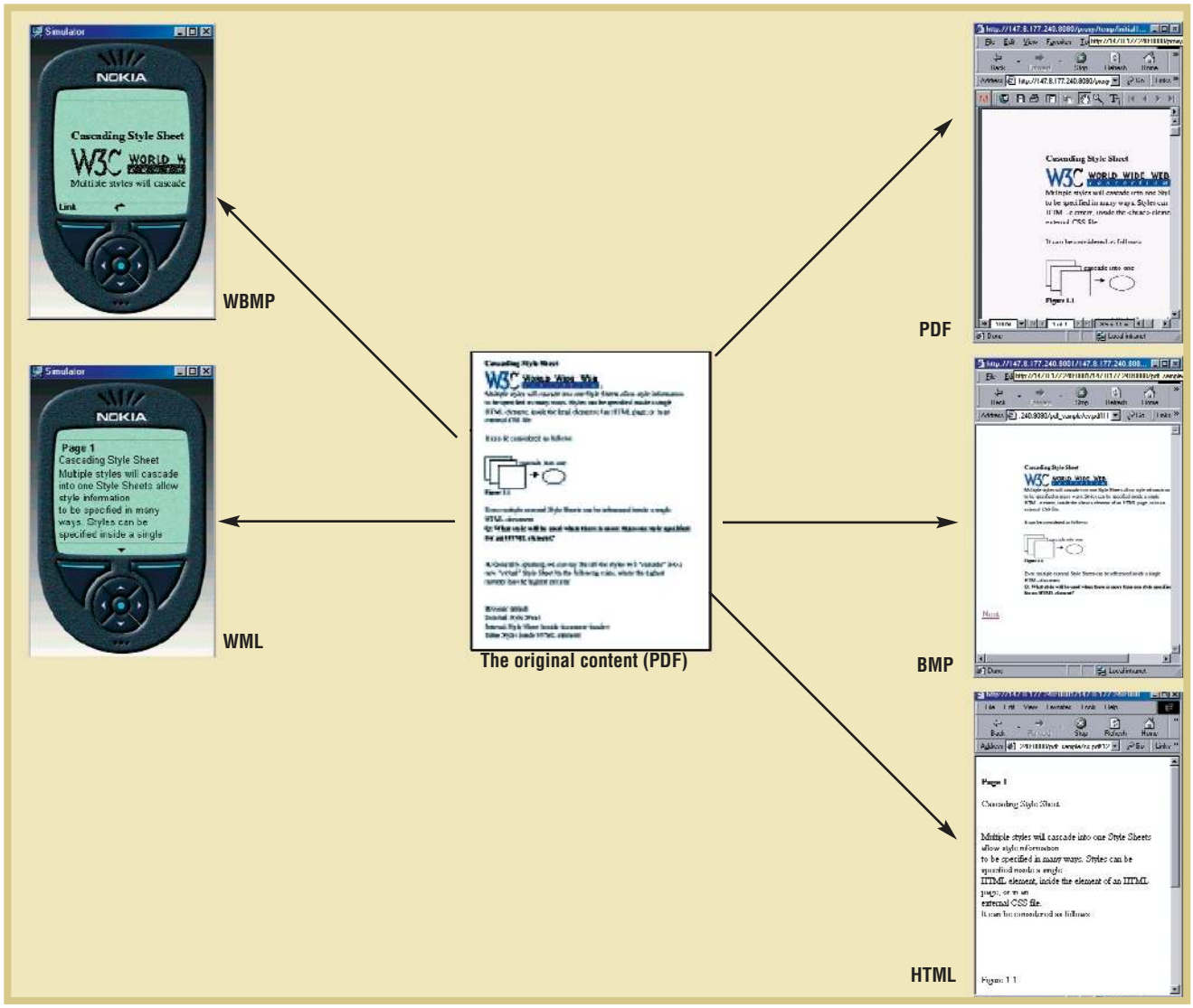

Figure 6. Delivering different modalities on the basis of device capability and user preference. The left side shows different presentations for a WAP device based on whether the user favors preserving the original modality (the upper left) or minimizing downloading time (the lower left). The right side shows presentations for a PDA, illustrating variations in trade-offs between preserving the modality (the upper right) and minimizing the downloading time (the lower right).

We can define a threshold for the optimization accuracy ($A_{threshold}$)—for example, 70 percent—such that when NORST's accuracy level falls below $A_{threshold}$, the system will automatically switch to SLL. This guarantees that the resulting optimization's accuracy will be bounded by this threshold value.

The PDF Document Content Adaptation System

Our PDF Document Content Adaptation System is aware of the user context in five quality domains: color, downloading time, scaling, modality, and segment. We previously discussed color preferences. The user can also specify the maximum downloading time he or she will tolerate for content delivery. The modality domain lets the user specify how he or she feels about preserving the mode versus transcoding the original content to a different mode. The scaling domain has four values corresponding to the output format: WML, HTML, bitmap, and PDF. The segment domain corresponds to how the user feels about cropping the content, with four different cropping ratios to choose from.

For the device capability context, we use information derived from some commercial portable-device models. The system takes into account the screen size, the supported color depths, supported media types, markup language, and the memory buffer size (for example, the maximum deck size acceptable for the WAP phone).

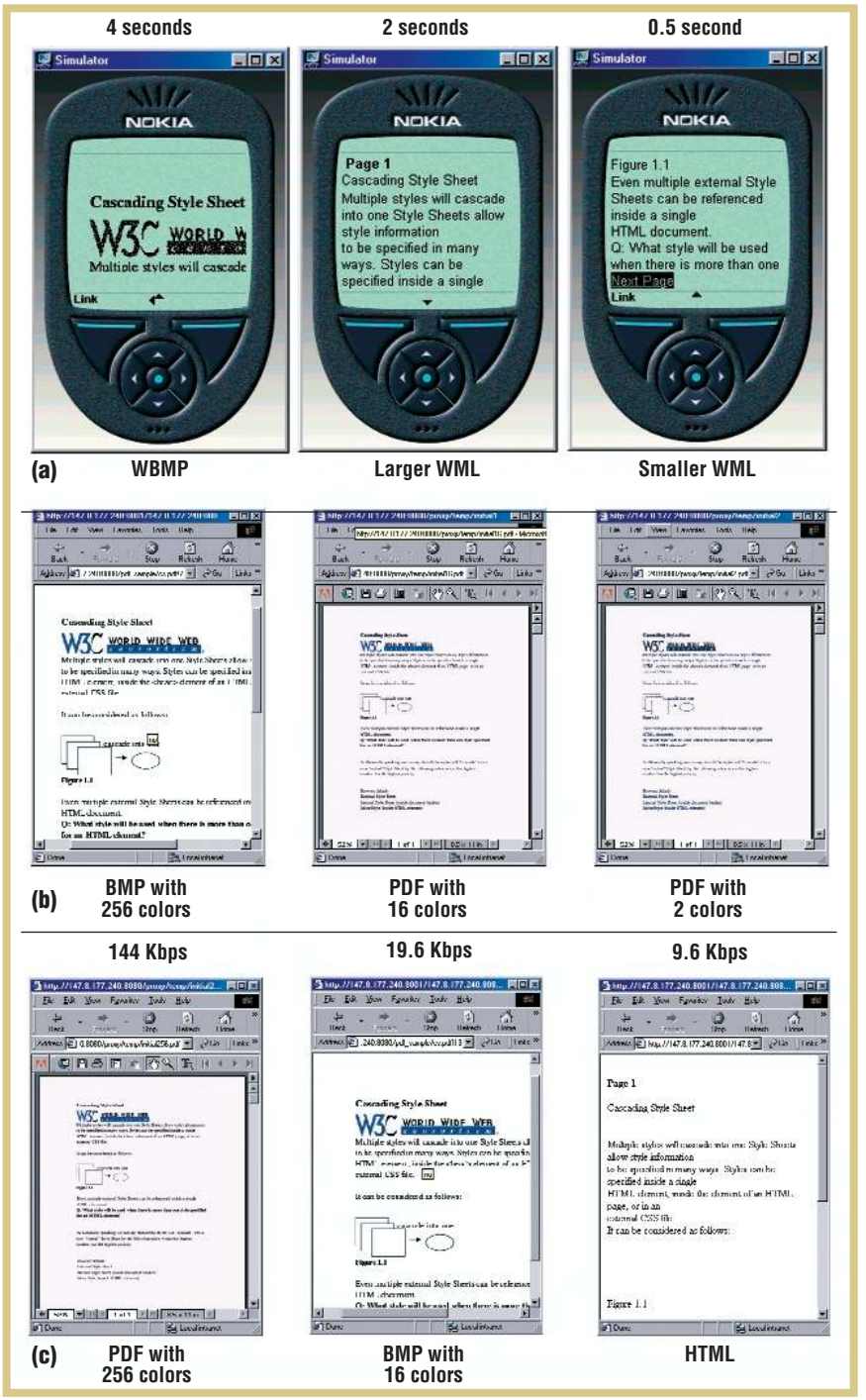

Figure 7. The content adaptation system adjusts the content version to meet changing requirements: (a) maximum tolerable downloading time; (b) color depth preferences; (c) available bandwidth.

For the network context, we use parameters such as bandwidth and round-trip time of some popular communication channels—for example, Code Division Multiple Access, General Packet Radio Service, and Cellular Digital Packet Data. Figure 2 shows an overview of the parameters.

By supplying values to these parameters, a Web browser or a WAP device simulator (we used the Nokia 6210 simulator; see www.nokia.com/phones/6210) can emulate the operation of any type of client device, even a fictitious one. In practice, techniques for automatically discovering the client device type (through, for example, HTTP headers), networking characteristics, and some means of client identification (explicit userid or cookies) would generate the necessary context information. Our system uses userid embedding in URLs to identify the end user.

We conducted several experiments to demonstrate both the prototype's usefulness in various practical application scenarios and our approach's viability. These experiments also show the negotiation model's sensitivity and awareness toward different situational or user-centric contexts. To perform graphical-manipulation operations such as image mode transcodings involving PDF-to-image conversion and color depth reduction, we used ImageMagick 5.3.8. To convert PDF to HTML, we used pdftohtml 0.21. We also implemented a simple HTML-to-WML transcoder. For all the cases, the time the negotiation algorithm (SLL-NORST) takes to arrive at a transcoding decision is small and is negligible compared to the transcoding and transmission times.

Modality preservation versus downloading time

Figure 6 shows sample results of our system delivering the same PDF document to a WAP device (on the figure's left) and a PDA (on the figure's right). The difference in presentation is due to the user perception on two conflicting factors: modality preservation and downloading time. For the WAP device, if the user prefers preserving the original modality, the system will produce an image (WBMP). If the user prefers minimizing downloading time, the system will produce a text version (WML). For the PDA, we tested the negotiation algorithm's sensitivity by varying the trade-off between these two factors while keeping the other factors constant. Correspondingly, our system presented the original content version in PDF, an image (BMP), or text (HTML), as the figure shows.

Maximum tolerable downloading time

If the user specifies a maximum tolerable downloading time, the system will negotiate the content version automatically to meet this specification. In this case, the content might require segmenting—for example, cropping the large HTML text and moving the residue to another page linked by the "next" anchor. In Figure 7a, the specified maximum tolerable downloading time decreases from left to right.

Color versus downloading time

Figure 7b demonstrates the effect of the negotiation algorithm trying to strike a balance between the conflicting factors of color and downloading time. With decreasing weights on the user's perception of color, the system responds with content versions of 256 colors, 16 colors, and 2 colors for the PDF document, to minimize downloading time over a slow network.

Network characteristics

We tested the system's adaptability to different network characteristics by varying bandwidth while keeping the other factors constant. As Figure 7c shows, the system switches modality to suit the connection's current bandwidth to keep downloading time within the allowed tolerance.

Device capability

To test the system's ability to handle heterogeneous devices, we adjusted the device's memory buffer size to see whether the system will automatically return the optimal content version. The results were similar to those for the network characteristics and closely agreed with our expectations.

We plan to extend the negotiation model to apply transcoding to the tasks a user wishes to perform on a device, rather than just to the delivered content. By "tasks," we mean programs or program functions. In the future, users will be able to download program versions on demand on the basis of their need and the context. For example, an e-mail program could have several versions: read only, read and compose, or text only. Such function adaptation is much more complex than content adaptation. We also plan to extend our prototype's document model to include images so that the quality domains could also include histograms on color use, segmentations, and so on.

In an ongoing project related to this one, we study the trade-off between dynamic and static adaptation.[12] To produce content versions, the decision engine can exploit static preadaptation (sacrificing I/O performance for CPU performance) or dynamic real-time adaptation (sacrificing CPU performance for I/O performance). A mixed approach can yield a productive balance between these two modes, leading to the most cost-effective methodology for content synthesis, with the decision engine's guidance. P

ACKNOWLEDGMENTS

We are grateful to the reviewers for their useful comments.

REFERENCES

1. T.L. Pham et al., "Composite Devices Computing Environment: A Framework for Situated Interaction Using Small Screen Devices," *Personal and Ubiquitous Computing*, vol. 5, no. 1, 2001, pp. 25–28.

2. T.W. Bickmore and B.N. Schilit, "Digestor: Device-Independent Access to the World Wide Web," *Proc. 6th World Wide Web Conf.* (WWW 6), 1997, pp. 655–663; www.scope.gmd.de/info/www6/technical/paper177/paper177.html.

3. S. Saha, M. Jamtgaard, and J. Villasenor, "Bringing the Wireless Internet to Mobile Devices," *Computer*, vol. 34, no. 6, June 2001, pp. 54–58.

4. R. Comerford, "Handhelds Duke It Out for the Internet," *IEEE Spectrum*, vol. 37, no. 8, Aug. 2000, pp. 35–41.

5. C.E. Ortiz and E. Giguere, *Mobile Information Device Profile for Java 2 Micro Edition (J2ME): Professional Developer's Guide*, John Wiley & Sons, New York, 2001.

6. J.C. Mogul, "Server-Directed Transcoding," *Computer Comm.*, vol. 24, no. 2, Feb. 2001, pp. 155–162.

7. A. Fox and E.A. Brewer, "Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation," *Proc. 5th Int'l World Wide Web Conf.* (WWW 5), 1996, pp. 1445–1456; http://www5conf.inria.fr/fich_html/papers/P48/Overview.html.

8. R. Han, V. Perret, and M. Naghshineh, "WebSplitter: A Unified XML Framework for Multi-Device Collaborative Web Browsing," *Proc. Computer Supported Cooperative Work 2000* (CSCW 2000), ACM Press, New York, 2000, pp. 221–230.

9. C.-S. Li, R. Mohan, and J.R. Smith, "Multimedia Content Description in the InfoPyramid," *Proc. IEEE Int'l Conf. Acoustics Speech and Signal Processing* (ICASSP 98), IEEE Press, Piscataway, N.J., 1998.

10. R. Mohan, J.R. Smith, and C.-S. Li, "Adapting Multimedia Internet Content for Universal Access," *IEEE Trans. Multimedia*, vol. 1, no. 1, Mar. 1999, pp. 104–114.

11. J.-P. Richter and H. de Meer, "Towards Formal Semantics for QoS Support," *Proc. 17th Ann. Joint Conf. IEEE Computer and Communications Societies* (INFOCOM 98), vol. 2, IEEE Press, Piscataway, N.J., pp. 472–479.

12. W.Y. Lum and F.C.M. Lau, "On Balancing between Transcoding Overhead and Spatial Consumption in Content Adaptation," to appear in *Proc. Mobicom 2002*, ACM Press, New York, 2002.

the AUTHORS

Wai Yip Lum is an M.Phil. candidate in the University of Hong Kong's Computer Science and Information Systems Department. His research interest is in the design of content adaptation services for mobile and pervasive computing. He received his B.Eng. in computer engineering from the University of Hong Kong. Contact him at the Univ. of Hong Kong, Pokfulam Rd., Hong Kong; wylum@csis.hku.hk.

Francis C.M. Lau is the head of the University of Hong Kong's Department of Computer Science and Information Systems. His research interests are in parallel and distributed computing, operating systems, and mobile computing. He received his B.Sc. from Acadia University and his M.Math. and PhD from the University of Waterloo, all in computer science. He is a member of the IEEE and served as a vice president of the IEEE Computer Society in 1999. Contact him at the Dept. of Computer Science and Information Systems, Univ. of Hong Kong, Pokfulam Rd., Hong Kong; fcmlau@csis.hku.hk; www.csis.hku.hk/~fcmlau.