Πανεπιστήμιο Πατρών
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

ΜΕΛΕΤΗ ΚΑΙ ΑΝΑΠΤΥΞΗ ΤΕΧΝΙΚΩΝ ΒΕΛΤΙΩΣΗΣ ΤΗΣ ΠΛΟΗΓΗΣΗΣ ΤΩΝ ΧΡΗΣΤΗ ΣΤΟ ΔΙΑΔΙΚΤΥΟ

(WEB COMPONENTS)

ΚΩΝΣΤΑΝΤΙΝΟΣ Δ. ΣΥΡΕΓΓΕΛΑΣ
Α.Μ. 2189

Διπλωματική Εργασία

υποβληθείσα για την επί μέρους πλήρωση των προϋποθέσεων του Διπλώματος του Μηχανικού Ηλεκτρονικών Υπολογιστών και Πληροφορικής

Σεπτέμβριος 2005

Επιβλέπων Καθηγητής: Χρήστος Μπούρας

Πάτρα 2005

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΡΟΛΟΓΟΣ

Για τη δημιουργία της παρούσας διπλωματικής θα ήθελα να ευχαριστήσω τον μεταπτυχιακό φοιτητή του πολυτεχνικού τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής του Πανεπιστημίου Πατρών Μισεδάκη Ιωάννη καθώς και τον καθηγητή του ίδιου τμήματος Μπούρα Χρήστο για την πολύτιμη βοήθειά τους χωρίς την οποία θα ήταν αδύνατη η συγγραφή της παρούσας διπλωματικής..

1 ΕΙΣΑΓΩΓΗ

Οι σύγχρονες ανάγκες των χρηστών του Δικτύου απαιτούν την πλοήγησή τους σε πλήθος σελίδων με ποικίλης ύλης πληροφορίες. Το μεγάλο πλήθος των σελίδων όμως απαιτεί και αρκετό χρόνο και μεγάλο όγκο δεδομένων που "κατεβαίνουν" τα οποία πιθανό να μην έχουν κάποια ιδιαίτερη σημασία για το χρήστη. Σε αυτή τη λογική λοιπόν θα προσπαθήσουμε να φτιάξουμε μία σελίδα στο δίκτυο, η οποία θα είναι προσωποποιημένη για τον κάθε χρήστη και θα περιέχει τμήματα άλλων σελίδων που περιέχουν την "καθαρή" πληροφορία . Με αυτή τη λύση ο χρήστης κυρίως για τις καθημερινές τους πλοηγήσεις θα χρειάζεται να ανοίγουν μόνο μία σελίδα και όχι πολλές. Έτσι κερδίζεται αρκετός χρόνος και περιορίζεται σημαντικά ο όγκος πληροφορίας που "κατεβαίνει" στον υπολογιστή του χρήστη.

Πάνω σε αυτή την ιδέα έχουν γίνει ήδη κάποιες άλλες προσπάθειες στο παρελθόν. Αυτό αποδεικνύεται από διάφορα paper σχετικά που υπάρχουν στο Δίκτυο και από διάφορα εργαλεία για αυτή τη δουλειά. Όπως είναι το JTidy που είναι ένα εργαλείο για να διαχωρίζει τα Components των σελίδων και έχει σχεδιαστεί κατά τέτοιο τρόπο ώστε να χρησιμοποιηθεί στο μέλλον για τη δημιουργία τροποποιημένων σελίδων. Σε αυτή την κατεύθυνση σημαντική είναι και η διπλωματική μεταπτυχιακή εργασία του Ιωάννη Μισεδάκη, ο οποίος δημιούργησε την προαναφερόμενη σελίδα της οποίας προσπάθειες για τη βελτίωση του αλγορίθμου της παρουσιάζονται στην παρούσα εργασία.. Επίσης το γενικότερο ενδιαφέρον προς αυτή την κατεύθυνση αποδεικνύεται και από το πλήθος των συνεδρίων που γίνονται διεθνώς για την εξοικονόμηση κυρίως χρόνου απο το χρήστη για την πρόσβαση στις επιθυμητές πληροφορίες.

Στην παρούσα διπλωματική εργασία αρχικά θα μελετήσουμε τον αλγόριθμο τμηματοποίησης των σελίδων, οι οποίες θα δημιουργηθούν σαν πηγή για την προσωποποιημένη σελίδα. Στη συνέχεια θα μελετήσουμε τη διάφορα θέματα όπως είναι η δημιουργία της προσωποποιημένης σελίδας, η σύνθεσή της και δίαφορα θέματα που μας απασχόλησαν στην υπολποίηση αυτής της ιδέας. Στη συν΄χεια θα αναφερθούμε στον αναλυτή των Web Components στην μηχανή προσωποποίησης των σελίδων και μια σχετική αναφορά θα γίνει στη βάση δεδομένων της μηχανής μας. Στη συνέχεια στο επόμενο κεφάλαιο αναφέρονται γενικά πράγματα για τις κλάσεις που χρησιμοποιούνται καθως και η λειτουργία της μηχανής. Τέλος μελετάται η προοπτική για μελλοντική αξιοποιήση της εργασίας καθώς και κάποια συμπεράσματα που προκύπτουν από αυτή τη διπλωματική.

2.1 ΕΙΣΑΓΩΓΙΚΟ ΣΗΜΕΙΩΜΑ

Κάπου στις αρχές της δεκαετίας του 90 και έχοντας γίνει το internet σχετικά διαδεδομένο οι σελίδες ήταν απλές περιέχοντας κατ' εξοχήν μόνο κείμενο. Με την πάροδο των χρόνων όμως και χάρη στην εξέλιξη των web browsers το κείμενο πλαισιώθηκε και από άλλες εφαρμογές όπως εικόνες, ήχους, πίνακες και άλλα (java applets και flash movies). Τα τελευταία χρόνια δε οι browsers άρχισαν να υποστηρίζουν και javascript με αποτέλεσμα πολλές σύνθετες σελίδες να είναι κανονικές εφαρμογές οι οποίες εκτελούνται μέσω του Web (web applications).

Βασικό στοιχείο αυτής της εξέλιξης ήταν και οι δημιουργία κάποιων sites που σκοπό έχουν την καθοδήγηση των χρηστών στο internet ώστε να βρίσκουν το site με το περιεχόμενο που τους ενδιαφέρει. Κοινώς αυτά τα sites δεν αφορούσαν σε κάποιο συγκεκριμένο θέμα απλά ήταν κατάλογοι άλλων sites. Στη συνέχεια αυτά τα sites βελτιώθηκαν προσφέροντας και άλλες υπηρεσίες στους επισκέπτες τους όπως ειδήσεις, μηχανή αναζήτησης ακόμα και e-mail. Έτσι αυτά τα sites έγιναν πολύ δημοφιλή στο χώρο του internet και έμειναν γνωστά στους χρήστες του web ως "portal" (δικτυακές πύλες).

Ανάλογη με την εξέλιξη των portals ήταν και αυτή των υπόλοιπων σελίδων και ο τρόπος συγγραφής αυτών. Σε αντίθεση με τα πρώτα χρόνια διάδοσης του web όπου η δημιουργία σελίδων ήταν μία απλή διαδικασία, η δημιουργία ενός portal ή μιας σελίδας με πλούσιο περιεχόμενο και με ευχάριστη διεπαφή για το χρήστη (User Interface) είναι μία διαδικασία πολύ πιο πολύπλοκη. Οι πληροφορίες, οι εικόνες, το κείμενο και οι σύνδεσμοι πλοήγησης κάνουν τη δημιουργία της σελίδας αρκετά πιο δύσκολη υπόθεση. Τα επιπρόσθετα εκτός του κειμένου στοιχεία τα οποία συνθέτουν μία σελίδα καθιστούν απαγορευτική την προσθήκη του περιεχομένου με τον τρόπο που γίνεται σε κάποιο αρχείο κειμένου (top to bottom). Ευτυχώς υπάρχουν αρκετές τεχνικές πλέον για τη διαμόρφωση του περιεχομένου ενός site (layout) που το κάνουν να φαίνεται με ένα πιο ελκυστικό τρόπο στο χρήστη και τονίζοντας περισσότερο τη σημασία και το σκοπό που επιτελεί. Μία από τις πρώτες τεχνικές που χρησιμοποιήθηκαν στο WWW διαφοροποιώντας μία σελίδα από ένα αρχείο κειμένου ήταν η μπάρα πλοήγησης στην επικεφαλίδα (header) ή στο πάνω αριστερό μέρος της σελίδας. Κατ' αυτόν τον τρόπο οι εικόνες που χρησιμοποιούνταν στο site για την πλοήγηση σε αυτό είχαν διαφορετική εμφάνιση και σημασία από το υπόλοιπο κείμενο που στην ουσία αποτελούσε και το πραγματικό περιεχόμενο της σελίδας. Σήμερα για τη δημιουργία ενός site χρησιμοποιούνται αρκετά πιο πολύπλοκες τεχνικές και πιο σύνθετος κώδικας html έτσι ώστε να προσφέρουν ένα ανάλογο layout του περιεχομένου τους.

Σε κάθε site πλούσιο σε περιεχόμενο ή σε κάθε δικτυακή πύλη εύκολα εντοπίζονται διακριτές περιοχές περιεχομένου που περιλαμβάνουν κείμενο, συνδέσμους ή εικόνες που αφορούν στο ίδιο θέμα. Τέτοιες περιοχές μπορεί να είναι για παράδειγμα νέα σχετικά με αθλητικά, οικονομικά, πολιτικά και άλλα. Αυτές τις περιοχές τις αποκαλούμε "Web Components" (ή "Web Fragments"). Κάθε σελίδα διαχωρίζεται σε τέτοιες περιοχές που το πλήθος τους εξαρτάται από την πολυπλοκότητα της διαμόρφωσης και της διαφοροποίησης (ως προς τις θεματικές περιοχές)του περιεχομένου. Συνήθως η κεντρική σελίδα των περισσότερων portal από 15-25 Web Components.

Τα Web Components που αποτελούν μία σελίδα είναι εντελώς ανεξάρτητα μεταξύ τους, πράγμα που συνεπάγεται ότι με κάποια κατάλληλη τεχνική θα μπορούσαν να απομονωθούν και να εξαχθούν από τη σελίδα και να χρησιμοποιηθούν για να συντεθεί μία άλλη σελίδα. Η τεχνικά αυτή έχει αναπτυχθεί ώστε Web Components άλλων σελίδων να μπορούν να χρησιμοποιηθούν από τους χρήστες σε μία άλλη σελίδα η οποία θα περιέχει μόνο αυτά τα Web Components που τους ενδιαφέρουν. Έτσι έχουμε τη δημιουργία προσωποποιημένων σελίδων. Αυτή η τεχνική βοηθά στο στη μείωση της αντιληπτής από το χρήστη καθυστέρησης (AXK), εφόσον λιγότερα δεδομένα κατεβαίνουν τοπικά στον υπολογιστή του και άρα συνεισφέρει και στη μείωση του κόστους πλοήγησης. Επίσης έχουμε και την ευκολία ο χρήστης να έχει όλο το περιεχόμενο που τον ενδιαφέρει σε μία σελίδα, αντί να επισκέπτεται πιο πολλές.

Τα Web Components εξάγονται από τις σελίδες με ανάλυση του HTML κώδικα που τις συνθέτουν, την αναγνώριση του κώδικα του κάθε Component και την ανάκτηση του κώδικα αυτού ως μία ξεχωριστή οντότητα. Στον κώδικα αυτό γίνονται κάποιες αλλαγές έτσι ώστε να χρησιμοποιηθεί σαν ως ανεξάρτητη οντότητα. Αυτό σημαίνει ότι μπορεί να χρησιμοποιηθεί μαζί με άλλα Web Components από την ίδια ή διαφορετική σελίδα χωρίς να δημιουργούνται προβλήματα στην εμφάνιση. Τα εξωτερικά αρχεία που συνθέτουν τη σελίδα, όπως αρχεία εικόνων, stylesheets, κώδικας java αποθηκεύονται στον server της υπηρεσίας.

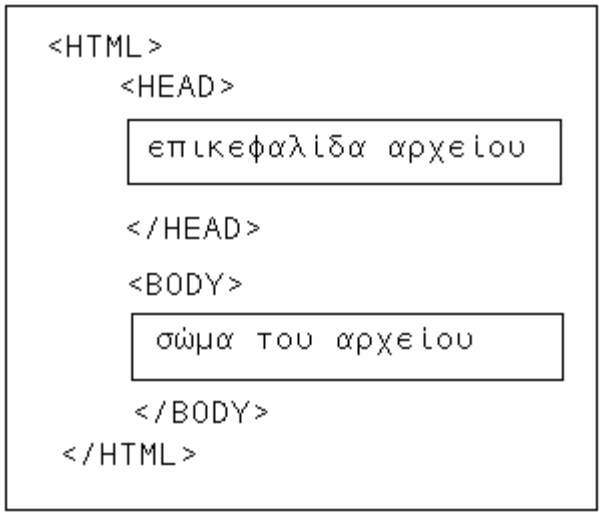
Ας μελετήσουμε λοιπόν τον αλγόριθμο που χρησιμοποιείται για την τμηματοποίηση της σελίδας στα Web Components που τη συνθέτουν και την εξαγωγή αυτών.

2.2 ΑΛΓΟΡΙΘΜΟΣ ΤΜΗΜΑΤΟΠΟΙΗΣΗΣ

Όπως είναι γνωστό οι browsers δημιουργούν τις σελίδες που βλέπει ο χρήστης βασιζόμενοι σε έναν HTML κώδικα. Το αρχείο του HTML κώδικα είναι το βασικό για την αναπαραγωγή της σελίδας, αλλά υπάρχουν και άλλα αρχεία που επιτελούν κάποιο συγκεκριμένο ρόλο στην αναπαράσταση της σελίδας (για παράδειγμα αρχεία για τα stylesheets, για τον κώδικα javascript, οι εικόνες, java applets, flash movies και άλλα). Το αρχείο με τον κώδικα HTML είναι ένα απλό αρχείο κειμένου, το οποίο

περιέχει markup tags που οδηγούν τον browser στην αναπαράσταση της σελίδας. Ο αλγόριθμος τμηματοποίησης που έχει ήδη αναπτυχθεί αλλά θα βελτιώσουμε χρησιμοποιεί το αρχείο από τον κώδικα HTML (που στη συνέχεια θα καλείται HTML file) για να αναγνωριστούν και να εξαχθούν τα αρχεία που συνιστούν τη σελίδα. Όμως όλα τα υπόλοιπα αρχεία που συνιστούν τη σελίδα είναι απαραίτητα για την παρουσίαση ενός Web Component.

Τα tags μέσα στο HTML file είναι εμφωλευμένα το ένα μέσα στο άλλο (nested). Ένα HTML file έχει τη δομή που φαίνεται στην παρακάτω φωτογραφεία. Το γεγονός ότι τα HTML tags είναι εμφωλευμένα μας δίνει τη δυνατότητα να αναπαραστήσουμε τον HTML κώδικα μιας σελίδας με μία δενδρική δομή (HTML tree). Οι κόμβοι αυτού του δέντρου που δεν είναι φύλλα αναπαριστούν τα tags, ενώ τα φύλλα αναπαριστούν το κείμενο της σελίδας. Η ρίζα αυτού του δέντρου είναι πάντα το HTML tag, το οποίο έχει δύο παιδιά (HEAD και BODY). Ο κώδικας που είναι υπεύθυνος για τη σχεδίαση της σελίδας βρίσκεται κάτω από τον κόμβο BODY, ενώ κάτω από τον κόμβο HEAD βρίσκονται τα tags με πληροφορίες επικεφαλίδας όπως (όπως ο τίτλος της σελίδας ή meta-tags). Η δενδρική αυτή αναπαράσταση του κώδικα HTML σημαίνει ότι μπορούμε να εξάγουμε τα Web Components που απαρτίζουν τη σελίδα εξάγοντας απλώς κάποιους κόμβους από το HTML tree.



Ο αλγόριθμος τεμαχισμού της σελίδας θα πρέπει να αναγνωρίζει κόμβους του HTML tree που αναπαριστούν web components. Το βασικότερο πρόβλημα σε αυτή τη διαδικασία είναι η 'υποκειμενική' φύση της διαδικασίας αναγνώρισης ενός Web Component. Μία περιοχή που από κάποιον θεωρείται περιοχή που περιέχει περιεχόμενο της ίδιας θεματικής ενότητας για κάποιον άλλο χρήστη μπορεί να μην είναι και απλά να πιστεύει ότι θα έπρεπε να αναλυθεί σε περισσότερα Web Compnents. Αυτό φαίνεται από τη σελίδα του BBC όπου θα μπορούσε να τεμαχιστεί σε διαφορετικά Web Components. Για να λυθεί αυτό το πρόβλημα θεωρούμε κάθε κόμβο του HTML tree ως Web Component. Για παράδειγμα μία εικόνα που περιέχεται σε ένα IMG tag ή μία παράγραφος του κειμένου που περιέχεται σε ένα P tag θα μπορούσαν να θεωρηθούν ως διαφορετικά Web Components. Ο χρήστης θα μπορούσε να συνδυάσει κάποια από αυτά τα Web Components για να κατασκευάσει τα δικά του Web Components. Αυτό όμως θα έκανε αρκετά περίπλοκη τη διαδικασία την υπηρεσία που παρέχεται στους χρήστες και θα οδηγούσε και σε μεγάλα

προβλήματα όταν το περιεχόμενο του site αλλάζει και ο μηχανισμός του τεμαχισμού (τμηματοποίησης) θα έπρεπε να αναγνωρίσει τα νέα στιγμιότυπα (εκδόσεις) των Web Components.

Για να ξεπεραστούν όλα αυτά τα προβλήματα, ο αλγόριθμος που σχεδιάστηκε και υλοποιήθηκε βασίστηκε μόνο στη δομή της σελίδας την οποία έχει κατασκευάσει ο συγγραφέας της σελίδας. Η πλειοψηφία των σελίδων στο web χρησιμοποιούν πίνακες για τη διαμόρφωση του layout. Αυτό μας οδήγησε στην απόφαση να χρησιμοποιηθεί η ιεραρχία των πινάκων (που καθορίζεται με βάση τα table tags) σαν το βασικό κριτήριο για την τμηματοποίηση της σελίδας σε διακριτά Web Components.Αν αγνοήσουμε όλα τα tags (κόμβους) ενός HTML tree, εκτός από τους κόμβους που αντιστοιχούν στα tags «TABLE», το βάθος και η πολυπλοκότητα του HTML tree μειώνονται σημαντικά σε μέγεθος. Κάποια από τα TABLE tags του δέντρου χρησιμοποιούνται μόνο για τη διαμόρφωση της σελίδας (δηλαδή είναι κενά περιεχομένου), ενώ κάποια άλλα περιέχουν περιεχόμενο. Βασιζόμενοι στην ποσότητα του περιεχομένου (μέγεθος του κειμένου) που αντιστοιχεί σε κάθε κόμβο, ο αλγόριθμος επιλέγει ποιοι κόμβοι πρέπει να θεωρηθούν ως Web Components που συνθέτουν τη σελίδα. Κάθε κόμβος σε αυτό το μειωμένο σε μέγεθος δέντρο (το οποίο ονομάζεται index tree-δέντρο ευρετηρίου) περιέχει ένα σύνδεσμο στον αντίστοιχο κόμβο του HTML tree. Ο αλγόριθμος τμηματοποίησης χρησιμοποιεί το index tree για να κάνει υπολογισμούς σχετικά με την τμηματοποίηση της σελίδας και στη συνέχεια ανακτά το περιεχόμενο των Web Components ακολουθώντας τους συνδέσμους στο HTML tree. Το index tree για το site του BBC φαίνεται από κάτω. Κάθε κόμβος με κόκκινο περίγραμμα αναπαριστά έναν πίνακα που εκλέχθηκε ως Web Component.

Ο αλγόριθμος που αναπτύχθηκε έχει το πλεονέκτημα της χρήσης της δομής της σελίδας που καθορίστηκε από το δημιουργό της σελίδας για την τμηματοποίηση της σελίδας στο web. Παρόλα ταύτα, υπάρχουν και κάποια μειονεκτήματα στον αλγόριθμο. Το βασικότερο και πιο εμφανές είναι ότι μπορεί να χρησιμοποιηθεί μόνο για σελίδες που χρησιμοποιούν την τεχνική των εμφωλευμένων πινάκων για τη διαμόρφωση του layout τους. Βέβαια αυτές οι σελίδες αποτελούν τη συντριπτική πλειοψηφία των σελίδων που συναντάμε στο δίκτυο και αυτών που έγιναν τα πειράματα για την ανάπτυξη του αλγορίθμου. Ένα άλλο πρόβλημα είναι ότι μερικές φορές η τμηματοποίηση της σελίδας μπορεί να οδηγήσει σε Web Components, τα οποία έχουν σημαντικές αποκλίσεις στο μέγεθός τους. Αυτά όμως τα προβλήματα αντιμετωπίζονται με κάποιες βελτιώσεις στον αλγόριθμο τμηματοποίησης, χωρίς να επηρεάζουν την υπόλοιπη πλατφόρμα. Στη συνέχεια της αναφοράς θα δοθούν κάποιες λύσεις σε αυτά τα προβλήματα εξελίσσοντας τον ήδη υπάρχον αλγόριθμο.

Ο αλγόριθμος τμηματοποίησης χρησιμοποιείται για την ανάλυση και τμηματοποίηση των σελίδων, η οποία περιλαμβάνει δύο φάσεις: τη φάση εκπαίδευσης (training phase) και τη φάση ανανέωσης (update phase). Τόσο στη φάση ανανέωσης, όσο και στη φάση εκπαίδευσης ένας μηχανισμός λογισμικού (ο οποίος υλοποιεί τον αλγόριθμο τμηματοποίησης) κατεβάζει τη σελίδα, την αναλύει (parsing) και την τεμαχίζει σε Web Components. Στη φάση εκπαίδευσης, ο αλγόριθμος τμηματοποίησης υπολογίζει τις περιοχές που θα είναι υποψήφιες για επιλογή ως Web Components, ενώ στη φάση ανανέωσης ο αλγόριθμος τμηματοποίησης ανανεώνει τις πληροφορίες που είναι αποθηκευμένες για κάθε Web Component (όπως για παράδειγμα τον HTML κώδικα). Αν ο αλγόριθμος τμηματοποίησης βρει αλλαγή στη δομή της σελίδας ή στον αριθμό των Web (κατά τη φάση της ανανέωσης) προσπαθεί να διορθώσει το πρόβλημα υπολογίζοντας ξανά τα web components της σελίδας.

Τα βήματα του αρχικού αλγόριθμου τμηματοποίησης παρουσιάζονται παρακάτω:

1. Κατέβασε το τελευταίο στιγμιότυπο (έκδοση) της σελίδας από το αντίστοιχο URL.
2. Ανέλυσε τη σελίδα κα φτιάξε το HTML tree.
3. Ανέλυσε το HTML tree και φτίαξε το index tree.
4. Ανέλυσε το index tree και υπολόγισε ποιοι κόμβοι πρέπει να επιλεγούν ως Web Components.
5. Έλενξε αν υπάρχουν διαφορές στη δομή του index tree από το index tree της φάσης εκπαίδευσης ή αν υπάρχουν διαφορές στον αριθμό των web components που έχουν επιλεγει. Αν υπάρχουν διαφορές, επαναυπολόγισε τα web components.
6. Εξήγαγε τα web components από το HTML tree και αποθήκευσέ τα.

Τα βήματα 1-4 χρησιμοποιούνται και για τη φάση εκπαίδευσης και για τη φάση ανανέωσης, ενώ τα 5,6 μόνο στη φάση ανανέωσης.

Το βήμα 1 είναι σχετικά απλό. Ο αλγόριθμος τμηματοποίησης ζητάει το html αρχείο της σελίδας από τον web server και το κατεβάζει τοπικά στον υπολογιστή του service provider που εκτελεί το ειδικό λογισμικό για την παροχή της υπηρεσίας.

Για την υλοποίηση του 2^{ου} βήματος του αλγόριθμου τμηματοποίησης δημιουργήθηκε ένας html parser (αναλύει τον html κώδικα). Παίρνει σαν είσοδο το κείμενο του html αρχείου που ανακτήθηκε στο βήμα 1 και κατασκευάζει το html tree. Σε αυτή τη δομή (html tree) αποθηκεύονται όλες οι πληροφορίες που είναι απαραίτητες για την ανακατασκευή του html αρχείου στην αρχική του μορφή. Ο σκοπός αυτού του μετασχηματισμού (απλό αρχείο κειμένου σε δενδρική δομή) είναι να βρίσκονται τα δεδομένα της html σελίδας σε μία μορφή που θα επιτρέπει την ευκολότερη, αποδοτικότερη και αποτελεσματικότερη διαχείρισή τους. Ο αλγόριθμος τμηματοποίησης ξεκινάει επίσης μία διαδικασία στο παρασκήνιο (κατά τη φάση της ανανέωσης μόνο), όπου κατεβαίνουν όλα τα επιπλέον αρχεία που χρειάζονται για την αναπαραγωγή της σελίδας (εικόνες, javascripts κ.α.).

Το βήμα 3 του αλγόριθμου τμηματοποίησης παίρνει σαν είσοδο το html tree και κατασκευάζει το index tree. Το index tree χρησιμοποιείται στο βήμα 4 για την αναγνώριση των κόμβων του html tree που θα εξαχθούν ως web components. Αυτό το δέντρο χρησιμοποιείται επίσης και σαν ευρετήριο για τους κόμβους στο html tree, η οποία είναι μια αρκετά μεγαλύτερη δομή σε μέγεθος που περιέχει και όλα τα δεδομένα του αρχείου html. Αυτός είναι και ο λόγος που ονομάζεται index tree. Το index tree είναι σαφέστατα πολύ μικρότερο από το html tree σε μέγεθος. Κατά τη φάση του σχεδιασμού αποφασίστηκε η χρήση αυτής της δομής κυρίως για δύο λόγους: οι απαραίτητοι υπολογισμοί γίνονται πολύ πιο γρήγορα και οι αλγόριθμοι που υλοποιήθηκαν ήταν πολύ πιο απλοί (με συνέπεια το όλο σύστημα να αναπτυχθεί πιο γρήγορα και με λιγότερα λάθη). Το index tree έχει τη δομή του html tree αν από αυτό αφαιρέσουμε όλους τους κόμβους εκτός από τα table tags. Ο αλγόριθμος τμηματοποίησης στο 3^{ο} βήμα ακολουθεί την εξής διαδικασία για τη δημιουργία του index tree: ο αλγόριθμος ξεκινάει από τη ρίζα του html tree και διασχίζει αναδρομικά όλη τη δενδρική δομή. Για κάθε table κόμβο που συναντά (δηλαδή κόμβους που αντιστοιχούν σε table tag), προσθέτει ένα νέο κόμβο στο index tree στην αντίστοιχη θέση. Έτσι το index tree που δημιουργείται κρατάει τη δομή των πινάκων στην html σελίδα, χωρίς όμως να κρατάει τα επιπλέον δεδομένα που δεν είναι απαραίτητα για

τους υπολογισμούς που πρέπει να γίνουν. Το ID κάθε κόμβου εξαρτάται από τη θέση του κόμβου στο index tree. Πιο συγκεκριμένα, το ID είναι το μονοπάτι από τη ρίζα του δέντρου στον κόμβο. Κάθε παιδί ενός κόμβου έχει έναν αριθμό που δείχνει τη θέση του σε σχέση με τα υπόλοιπα παιδιά αυτού του κόμβου (π.χ. το δεύτερο παιδί έχει τον αριθμό 2). Ξεκινώντας από τη ρίζα και φτάνοντας στον κόμβο διανύουμε ένα μονοπάτι βάζοντας στη σειρά αυτούς τους αριθμούς (για τον κάθε κόμβο που διασχίζουμε). Το μονοπάτι αυτό και συνεπώς το ID καταγράφεται κατά τη δημιουργία του index tree. Για παράδειγμα το τρίτο παιδί του δεύτερου παιδιού της ρίζας έχει ID ΄2-3΄ (δε χρειάζεται να καταγράφεται η ρίζα του index tree επειδή ο κόμβος αυτός υπάρχει σε κάθε μονοπάτι). Σε κάθε κόμβο στο index tree υπάρχει ένας σύνδεσμος στον αντίστοιχο κόμβο (TABLE) στο HTML tree. Επίσης, καταγράφονται και κάποιες άλλες πληροφορίες που χρησιμοποιούνται για υπολογισμούς στο βήμα 4. Αυτές οι πληροφορίες περιλαμβάνουν το μέγεθος του κειμένου σε αυτόν τον κόμβο στο HTML αρχείο (με και χωρίς τα tags), το ID του κόμβου, τον αριθμό των εικόνων που βρίσκονται μέσα στον κόμβο αυτό και τέλος τον αριθμό των συνδέσμων που υπάρχουν μέσα σε αυτόν τον κόμβο. Αξίζει εδώ να σημειώσουμε ότι αυτές οι πληροφορίες εξάγονται από τον κόμβο και όλους τους απογόνους του στο HTML tree. Έτσι, ένας κόμβος του index tree αντιπροσωπεύει ένα ολόκληρο υποδέντρο του HTML tree. Επίσης, για την ανάκτηση του περιεχομένου ενός κόμβου από το index tree (του HTML κώδικα δηλαδή) πρέπει να ακολουθηθεί ο σύνδεσμος στον αντίστοιχο κόμβο στο HTML tree, μιας και δεν έχουμε καταγραφή αυτών των πληροφοριών για οικονομία μνήμης.

Οι αποφάσεις για το πώς θα τεμαχιστεί (τμηματοποιηθεί) η σελίδα λαμβάνονται στο $4^{ο}$ βήμα του αλγόριθμου τμηματοποίησης. Ο αλγόριθμος τμηματοποίησης χρησιμοποιεί το index tree που δημιουργήθηκε στο προηγούμενο βήμα της διαδικασίας τμηματοποίησης. Ξεκινάει διασχίζοντας και αναλύοντας το index tree ψάχνοντας να βρει κόμβους που ανταποκρίνονται σε κάποια συγκεκριμένα κριτήρια. Με το που βρεθεί ο κόμβος που πληρεί αυτά τα κριτήρια, ο αλγόριθμος σταματάει να διασχίζει τα παιδιά αυτού του κόμβου και ο κόμβος σημειώνεται σαν Web Component. Αυτό σημαίνει ότι ολόκληρο το υποδέντρο του index tree κάτω από αυτόν τον κόμβο θεωρείται σαν μία διακριτή οντότητα που μπορεί να χρησιμοποιηθεί από τους χρήστες της υπηρεσίας για τη δημιουργία των δικών τους σελίδων. Τα παιδιά αυτού του κόμβου είναι μέρος του component και δεν μπορούν να χρησιμοποιηθούν σύντομα, αφού σύμφωνα με υπολογισμούς του αλγόριθμου το περιεχόμενό τους είναι ελάχιστης σημασίας για να θεωρήσουμε τους κόμβους αυτούς Web Components. Όπως έχουμε ήδη αναφέρει για να λάβουμε τον κώδικα των κόμβων που έχει επιλεγεί ως Web Components, πρέπει να ακολουθηθεί ο σύνδεσμος στον αντίστοιχο κώδικα στο HTML tree και από εκεί να εξαχθεί το κείμενο του HTML κώδικα.
Τα κριτήρια που χρησιμοποιούνται για να αποφασιστεί αν ένας κόμβος του index tree (δηλαδή ένα τμήμα του HTML αρχείου) είναι κατάλληλο για να χρησιμοποιηθεί ως Web Component σχετίζονται με το μέγεθος του κόμβου και την εσωτερική του δομή (για παράδειγμα τον αριθμό των παιδιών και των απογόνων του). Στην παρούσα μορφή του, ο αλγόριθμος υπολογίζει το «μέγεθος του περιεχομένου» ενός κόμβου υπολογίζοντας το μέγεθος *καθαρού κειμένου* του κόμβου. Με τον όρο «καθαρό κείμενο» εννοούμε το κείμενο που περιέχεται στη σελίδα έχοντας αφαιρέσει τα HTML tags. Αν ο κόμβος p πληροί το ακόλουθο κριτήριο τότε τον σημειώνουμε ως Web Component, χωρίς να εξετάσουμε καν την εσωτερική δομή του:

(1) ή

$$1 \le Ratiop * (NumberOfContentNodes) \le 2 \quad (2)$$

όπου $$Ratiop = \frac{PureTextLengthInTheNodep}{PureTextLengthOfTheRootNode} \quad (3)$$

και $$AverageRatio = \frac{1}{NumberOfContentNodes} \quad (4).$$

Η σχέση 1 (ή η ισοδύναμή της 2) εκφράζει το διαισθητικό κριτήριο ότι ένα Web Component πρέπει να είναι «μέσου μεγέθους», δηλαδή ούτε πολύ μεγάλο ούτε πολύ μικρό σε σχέση με το μέγεθος της υπόλοιπης σελίδας. Το Ratiop υπολογίζεται διαιρώντας το μέγεθος του καθαρού κειμένου που περιλαμβάνεται στον κόμβο p με το μέγεθος του καθαρού κειμένου ολόκληρης της σελίδας, εκφράζοντας έτσι το ποσοστό του περιεχομένου του κόμβου σε σχέση με το περιεχόμενο της σελίδας. Αυτή η μετρική εκφράζει το σχετικό μέγεθος του Web Component (σχετικά με το μέγεθος ολόκληρης της σελίδας). Το AverageRatio είναι το ποσοστό του κειμένου ολόκληρης της σελίδας που ένας κόμβος θα είχε αν όλοι οι κόμβοι που περιέχουν περιεχόμενο (δηλαδή δε χρησιμοποιούνται για τη διαμόρφωση της σελίδας) είχαν το ίδιο μέγεθος. Αυτή την τεχνική τη χρησιμοποιούμε σα βάση για την προσέγγιση ενός μεσαίου μεγέθους component. Υπολογίζεται ως το αντίστροφο του αριθμού των κόμβων περιεχομένου του index tree. Αν το μέγεθος του περιεχομένου (κείμενο) ενός κόμβου είναι μεγαλύτερο από το μέσο μέγεθος (το οποίο εκφράζεται από το AverageRatio) ή μικρότερο από το διπλάσιο του μέσου μεγέθους, τότε ο κόμβος χαρακτηρίζεται «μέσου μεγέθους» και επιλέγεται σαν Web Component.

Η σχέση 2 θα μπορούσε να γραφεί σε μία πιο αφηρημένη μορφή ως εξής :

$$l \le Ratiop * (NumberOfContentNodes) \le u \quad (5)$$

όπου = (Number Of Content Nodes)

Οι τιμές των l και u εκφράζουν το κάτω (lower) και άνω (upper) όριο για το μέγεθος του κειμένου ενός κόμβου ώστε να θεωρηθεί «μέσου μεγέθους». Η σχέση 5 σημαίνει ότι αν το μέγεθος του κειμένου ενός κειμένου είναι μεγαλύτερο ή ίσο από το $\frac{l}{u\max}$ και μικρότερο ή ίσο από το $\frac{u}{u\max}$ του μεγέθους του κειμένου ολόκληρης της σελίδας, τότε ο κόμβος θεωρείται «μέσου μεγέθους» και επιλέγεται σαν Web Component. Αν αντικαταστήσουμε l =1 και u =2 στη σχέση 5,τότε παίρνουμε το κριτήριο που εκφράζει η σχέση 2. Οι τιμές 1 και 2 στα l και u αντίστοιχα έχουν επιλεγεί αυθαίρετα με μόνο κριτήριο την παρατήρηση ότι έχουν σαν αποτέλεσμα την καλή τμηματοποίηση των σελίδων. Στην περίπτωση που το l ήταν μικρότερο του 1 ο αλγόριθμος θα επέλεγε κόμβους με μέγεθος κειμένου μικρότερο από το μέγεθος του κειμένου του «μέσου κόμβου» (που είναι το AverageRatio). Αυτό όμως το μέγεθος είναι ήδη αρκετά μικρό. Η τιμή u =2 επιλέχθηκε μετά από πειράματα με αρκετά web site και αξιολόγηση των αποτελεσμάτων της τμηματοποίησης. Πρέπει να σημειωθεί ότι οι ιδανικές τιμές δε θα είναι πάντα οι ίδιες για όλα τα site αφού αυτές εξαρτώνται αποκλειστικά από τη δομή των σελίδων και το περιεχόμενό τους.

Το μέγεθος του περιεχομένου που περιλαμβάνεται σε ένα κόμβο TABLE δεν αποτελεί το μοναδικό κριτήριο τμηματοποίησης της σελίδας (αν και είναι κάτι σημαντικό). Το άλλο βασικό κριτήριο που χρησιμοποιείται για την επιλογή ενός κόμβου ως Web Component στηρίζεται πάνω στη δομή του index tree. Παρατηρήθηκε ότι συχνά οι περιοχές που λαμβάνονται σαν Web Components αποτελούνται από περισσότερους από ένα κόμβους (δηλαδή από εμφωλευμένα tags). Από αυτά τα TABLE Tags, το ένα περιέχει το κύριο σώμα του περιεχομένου του Component, ενώ τα υπόλοιπα είναι κόμβοι για τη διαμόρφωση του layout ή κόμβοι με ασήμαντη ποσότητα περιεχομένου (π.χ. ο τίτλος ενός άρθρου). Έτσι όταν ο αλγόριθμος τμηματοποίησης βρει έναν κόμβο του index tree,ο οποίος δεν είναι φύλλο και περιέχει λιγότερα από τέσσερα παιδιά και λιγότερο από πέντε (συνολικά) απογόνους (χωρίς εννοείται να υπολογίζουμε τους κόμβους που είναι για τη διαμόρφωση του layout) ο κόμβος επιλέγεται ως Web Component. Δοκιμές του αλγόριθμου με απενεργοποιημένο αυτό το κριτήριο έδειξαν ότι η τμηματοποίηση διάφορων διάσημων σελίδων δεν ήταν τόσο καλή όσο αυτή με το κριτήριο ενεργοποιημένο. Αυτό προκαλείται κυρίως από περιπτώσεις όπου οι κόμβοι, οι οποίοι διαισθητικά θα επιλεγούν ως Web Component απορρίπτονται εξαιτίας του μεγέθους τους. Έτσι αυτό το κριτήριο μας βοηθάει στη βελτίωση των αποτελεσμάτων του κριτηρίου που βασίζεται στο μέγεθος του περιεχομένου. Τα Web Component που έχουν επιλεγεί είτε λόγω του μεγέθους τους, είτε της δομής τους και περιέχουν περισσότερα από ένα Table Tag (με άλλα λόγια δεν αποτελούν φύλλα στο index tree) ονομάζονται «Complex Web Components» (CWC).

Τέλος, αν ο αλγόριθμος τμηματοποίησης φτάσει σε ένα φύλλο του index tree, το επιλέγει σαν Web Component, επειδή δε μπορεί να αναλυθεί περισσότερο. Αυτό μπορεί να μας οδηγήσει στην επιλογή κόμβων πολύ μικρών ή χωρίς καθόλου περιεχόμενο (απλά να περιέχουν για παράδειγμα εικόνες) ως Web Component. Παρόλα ταύτα το κριτήριο αυτό επιλέχθηκε επειδή δε θέλουμε να έχουμε απώλεια περιεχομένου από την αρχική σελίδα.

Όταν ο αλγόριθμος έχει διασχίσει όλο το index tree, κάνει κάποιες τελευταίες βελτιώσεις στην επιλογή των Components. Πιο συγκεκριμένα αν βρει ένα Web Component που είναι το μοναδικό παιδί του πατέρα του, επιλέγει τον πατέρα ως Web Component. Αυτό γίνεται επειδή πολύ πιθανά ο πατέρας του κόμβου που έχει επιλεγεί ως Component είναι ένα layout tag ή περιέχει κάποιο περιεχόμενο που συσχετίζεται με τον κόμβο-παιδί (όπως είναι για παράδειγμα ο τίτλος ή ο συγγραφέας ενός άρθρου).

Όταν το $4^{ο}$ βήμα του αλγόριθμου τμηματοποίησης τελειώσει, ο αλγόριθμος έχει διασχίσει όλο το index tree και έχει σημειώσει κάποιους κόμβους ως Web Component. Από το σημείο αυτό και μετά ο αλγόριθμός μας περιλαμβάνει δύο ακόμα βήματα, τα οποία χρησιμοποιούνται μόνο κατά τη φάση της ανανέωσης και θα αναλυθούν παρακάτω. Όσο για τη φάση της εκπαίδευσης, το index tree με τα επιλεγμένα Web Components κρίνεται αρκετό. Στη συνέχεια θα εξηγήσουμε αναλυτικά τι συμβαίνει στη φάση της εκπαίδευσης και τι στη φάση της ανανέωσης.

3.1 ΜΕΘΟΔΟΛΟΓΙΑ ΓΙΑ ΤΗΝ ΚΑΤΑΣΚΕΥΗ ΠΡΟΣΩΠΟΠΟΙΗΜΕΝΩΝ ΣΕΛΙΔΩΝ ΜΕ ΤΗΝ ΤΕΧΝΙΚΗ ΤΩΝ WEB COMPONENTS

Σε αυτή την ενότητα παρουσιάζεται η μεθοδολογία για την κατασκευή προσωποποιημένων σελίδων με την τεχνική των Web Components. Υπάρχουν τρεις φάσεις . ανάλυση των σελίδων και τμηματοποίηση (εξαγωγή των Web Components), επιλογή των Web Components από τους χρήστες και σύνθεση των προσωποποιημένων σελίδων από των Web Server για παρουσίαση στους χρήστες.

3.1.1 Ανάλυση και τμηματοποίηση των σελίδων

Η πρώτη φάση περιλαμβάνει ένα εργαλείο λογισμικού του οποίου ο ρόλος είναι να αναλύει συνεχώς τις επιλεγμένες σελίδες και να ανανεώνει τις πληροφορίες που είναι αποθηκευμένες σχετικά με αυτές και τον κώδικα HTML των Components. Αυτό το εργαλείο (Web Component Creator) δεν γίνεται install στους ηλεκτρονικούς υπολογιστές του χρήστη, αλλά λειτουργεί κεντρικά σαν πηγή δεδομένων για τον Web Server του παροχέα υπηρεσίας. Το εργαλείο αυτό κρατάει μία λίστα με τις σελίδες που έχει καθορίσει ο διαχειριστής της υπηρεσίας. Το εργαλείο αυτό κρατάει μία λίστα με τις σελίδες που έχει καθορίσει ο διαχειριστής υπηρεσίας. Τα Web Components που είναι διαθέσιμα για τη δημιουργία των προσωποποιημένων σελίδων εξάγονται από τα sites της λίστας. Αυτά τα Components δε δημιουργούνται δυναμικά με κάθε αίτηση των χρηστών αλλά εξάγονται και αποθηκεύονται από αυτό το εργαλείο. Ο κώδικας του κάθε Web Component ανανεώνεται σε τακτά χρονικά διαστήματα για την καλύτερη απόδοση του συστήματος. Αν επιλέγαμε τη δυναμική δημιουργία των Web Components με κάθε αίτηση των χρηστών, οι χρήστες θα έπρεπε να περιμένουν ένα αρκετά υπολογίσιμο χρονικό διάστημα μέχρι να εμφανιστεί η προσωποποιημένη τους σελίδα. Αυτό οφείλεται στο ότι όλες οι σελίδες που περιέχουν κάποιο από τα Components που έχει επιλέξει ο εκάστοτε χρήστης θα έπρεπε να κατέβουν στον υπολογιστή που δημιουργεί τα Components, να αναλυθούν και στη συνέχεια να δημιουργηθεί η σελίδα από τα επιλεγμένα Components. Αυτό θα μας οδηγούσε σε τεράστια σπατάλη σε Bandwidth και συνεπώς θα είχε σαν αποτέλεσμα την αύξηση του κόστους για τον παροχέα της υπηρεσίας. Επίσης μία τέτοια επιλογή θα είχε και σαν αποτέλεσμα τη σπατάλη πολλών υπολογιστικών πόρων στον υπολογιστή που δημιουργούν τα Web Components. Από την άλλη με τη χρήση του κατάλληλου εργαλείου εκτελείται μόνο το τελευταίο βήμα της προαναφερθείσας διαδικασίας (δημιουργία στον Web Server της προσωπικής σελίδας του χρήστη) για κάθε αίτηση χρήστη.

Η ανάλυση κα τμηματοποίηση της σελίδας είναι μία διαδικασία με πολλά βήματα. Περιλαμβάνει δύο κύριες φάσεις: η πρώτη είναι η φάση της εκπαίδευσης (training phase) όπου κάθε σελίδα εξετάζεται για μία συγκεκριμένη χρονική περίοδο και οι παροχές που μπορεί να αντιμετωπίζονται ως Web Component αντιμετωπίζονται. Σε αυτή τη διαδικασία, ο αλγόριθμος της εκπαίδευσης αναλύει τη σελίδα πολλές φορές, την τεμαχίζει (τμηματοποιεί) και αποθηκεύει κάποια δεδομένα από κάθε ανάλυση. Στη συνέχεια, όταν αρκετά δεδομένα έχουν συλλεχθεί, ο αλγόριθμος τα αναλύει και

υπολογίζει ποιες περιοχές της σελίδας θα εξαχθούν ως Web Components. Ο αλγόριθμος εκπαίδευσης αποθηκεύει πληροφορίες σχετικά με το κάθε Component που θα χρησιμοποιηθούν ως αναγνωριστής (identifier) αυτού του component στη φάση της ανανέωσης, η οποία ακολουθεί αμέσως μετά τη φάση της ανανέωσης, η οποία ακολουθεί αμέσως μετά της φάσης της εκπαίδευσης.

Όταν η διαδικασία της εκπαίδευσης του συστήματος σχετικά με κάποια σελίδα έχει ολοκληρωθεί, η διαδικασία της ανανέωσης ξεκινάει. Η διαδικασία αυτή διαρκεί όσο η συγκεκριμένη σελίδα είναι διαθέσιμη από το σύστημα για την επιλογή Web Components από αυτή. Η διαδικασία ανανέωσης τεμαχίζει τη σελίδα κα βασιζόμενη στις πληροφορίες που έχουν συλλεχθεί κατά τη διάρκεια της φάσης της εκπαίδευσης, ανανεώνει τα Web Components με τα τελευταία στιγμιότυπά τους.

Ας εξετάσουμε λοιπόν με μεγαλύτερη λεπτομέρεια τη διαδικασία εκπαίδευσης και ανανέωσης.

3.1.1.1 Διαδικασία εκπαίδευσης

Η διαδικασία εκπαίδευσης της ανάλυσης μιας σελίδας πρέπει να πραγματοποιηθεί πριν τα Web Components που απαρτίζουν αυτή τη σελίδα γίνουν διαθέσιμα στο σύστημα. Αυτά τα Components μπορούν να χρησιμοποιηθούν από τους χρήστες του συστήματος και να συμπεριληφθούν στις προσωπικές τους σελίδες μόνο όταν η φάση της εκπαίδευσης έχει ολοκληρωθεί με επιτυχία. Ο ρόλος της φάσης αυτής είναι να αναλύει τη σελίδα και να αποφασίσει πως θα τεμαχίζεται η σελίδα και πως θα επιλέγονται τα Web Components που την απαρτίζουν κατά τη φάση της ανανέωσης. Στο τέλος της διαδικασίας εκπαίδευσης για μία συγκεκριμένη σελίδα, η έξοδος του αλγόριθμου εκπαίδευσης είναι ο αριθμός των Web Components της σελίδας και ένας μοναδικός αναγνωριστής (unique identifier) για κάθε ένα από αυτά τα Components. Αυτός ο μοναδικός αναγνωριστής μπορεί να χρησιμοποιηθεί για την αναγνώριση ενός Web Component σε κάποιο στιγμιότυπο της σελίδας που έχει αλλαγές στη δομή της σελίδας ή στον αριθμό των Web Components που έχουν επιλεγεί. Η φάση της εκπαίδευσης δε θα χρειαζόταν αν δε συνέβαιναν αλλαγές στη δομή της σελίδας και στο σχετικό μέγεθος των περιοχών που την αποτελούν. Αν και αλλαγές αυτού του τύπου είναι σχετικά σπάνιες είναι σχεδόν βέβαιο ότι κάποια στιγμή θα συμβούν. Η φάση της εκπαίδευσης επιτρέπει στο σύστημα να έχει προκαθορισμένη γνώση σχετικά με την έξοδο που πρέπει να δίνει ο αλγόριθμος τμηματοποίησης, ώστε να μπορεί να διορθώσει τα προβλήματα που προκαλούνται όταν αυτή η έξοδος δεν είναι η αναμενόμενη.

Ο στόχος της φάσης της εκπαίδευσης είναι να επιτρέψει τη δημιουργία ενός μηχανισμού που θα μπορούσε να αντιμετωπίσει τα προβλήματα που προκαλούνται από αλλαγές στη δομή των σελίδων ή στον αριθμό των Web Components που επιλέγονται. Για να γίνουν όλα αυτά όμως βάζουμε βασική υπόθεση ότι κατά τη διάρκεια της φάσης εκπαίδευσης τέτοιες αλλαγές δε συμβαίνουν. Αν αλλαγές σαν αυτές που αναφερόμασταν συμβούν κατά τη φάση της εκπαίδευσης ο αλγόριθμος θα έπρεπε να γίνει αρκετά πιο περίπλοκος για να αντιμετωπίσει την αλλαγή. Στη φάση της εκπαίδευσης το σύστημα προσπαθεί να κατασκευάσει ένα μοναδικό αναγνωριστή (unique identifier) για κάθε Web Component. Κατά τη διάρκεια αυτής της φάσης χρησιμοποιούμε σαν αναγνωριστή για κάθε Component το ID του, δηλαδή το μονοπάτι από τη ρίζα του index tree στον κόμβο που αντιστοιχεί στο Component.

Άρα αν αυτό το μονοπάτι αλλάξει ή αν το Component δεν αναγνωρίζεται σε κάποιο στιγμιότυπο της σελίδας, το σύστημα δεν μπορεί να το εντοπίσει και να εξάγει τον κώδικά του. Υπάρχουν αρκετοί τρόποι για την επίλυση αυτού του προβλήματος. Το πιο απλό είναι να απορρίψουμε τα δείγματα στα οποία παρουσιάζονται αλλαγές. Είναι επίσης πιθανόν, χρησιμοποιώντας τόσο τη δομή όσο και το περιεχόμενο των Web Component nνα αναπροσαρμοστεί η δομή των index trees και η επιλογή των Web Components, αλλά αυτό είναι μία πολύπλοκη διαδικασία. Βέβαια τα προβλήματα που η διαδικασία εκπαίδευσης προσπαθεί να επιλύσει εμφανίζονται σπάνια, άρα είναι πιο σπάνιο λοιπόν να εμφανιστούν κατά τη διάρκεία της.

Ένα Web Component έχει διάφορα χαρακτηριστικά που το διαφοροποιούν από τα υπόλοιπα (εκτός από τις διαφορές στο περιεχόμενο εννοείται). Τη θέση μέσα στο index tree, η οποία καθορίζει το ID του, τη σχετική του θέση με τα άλλα Web Components, το περιεχόμενό του (σε κείμενο ή εικόνες) και άλλα. Παρόλα είναι αρκετά δύσκολο να βρεθεί ένα κριτήριο που να χρησιμοποιείται για να αναγνωρίζεται δυναμικά ένα Web Component από τα υπόλοιπα που περιέχονται στη σελίδα. Οι χρήστες πρέπει να μπορούν να επιλέξουν ποια Components επιθυμούν να βλέπουν στην προσωποποιημένη τους σελίδα και το σύστημα θα πρέπει να μπορεί να τα αναγνωρίσει από τη λίστα των Components που έχουν εξαχθεί από τον αλγόριθμο τμηματοποίησης. Η φάση της εκπαίδευσης που διαρκεί από μία ως και αρκετές μέρες στοχεύει στη δημιουργία ενός μοναδικού αναγνωριστή για κάθε Web Component από κάθε σελίδα.

Θα πρέπει εδώ να αναφερθεί ότι τα Web Component ανάλογα με τη σελίδα κατατάσσονται σε τρεις κατηγορίες χρησιμοποιώντας το κριτήριο της αλλαγής του περιεχομένου τους:

- Web Components που το περιεχόμενό τους δεν αλλάζει ποτέ (για παράδειγμα μία περιοχή με συνδέσμους σε κατηγορίες ειδήσεων)
- Web Components που όλο το περιεχόμενό τους είναι δυνατό να αλλάξει (για παράδειγμα μια περιοχή με τίτλους ειδήσεων)
- Web Components που κάποιο μέρος αλλάζει ,ενώ κάποιο άλλο μέρος μένει το ίδιο (για παράδειγμα μια περιοχή με τίτλους ειδήσεων που έχει στην κορυφή μια επιγραφή «Τίτλοι Ειδήσεων»).

Η φάση της εκπαίδευσης χρησιμοποιεί το σταθερό κομμάτι του περιεχομένου των Component για την πρώτη και τρίτη κατηγορία ώστε να αναθέσει ένα μοναδικό αναγνωριστή (unique identifier), ενώ χρησιμοποιεί για τον ίδιο σκοπό τη σχετική θέση των Component και το μέγεθος του περιεχομένου όσον αφορά στα Components της δεύτερης κατηγορίας.

Η φάση της εκπαίδευσης για μία σελίδα μπορεί να χωριστεί σε 4 υποφάσεις:

1. Φάση αλλαγής δεδομένων
2. Σύγκριση των Content Vectors (διανυσματικού περιεχομένου) των στιγμιότυπων του ίδιου Web Component και εξαγωγή ενός μόνο Constant Content Vector (CCV) για κάθε Component.
3. Σύγκριση των Content Vectors όλων των Web Components μιας σελίδας και καθορισμός ενός μοναδικού Identifier Content Vector (ICV-Αναγνωριστικό Διάνυσμα Περιεχομένου) για κάθε Web Component. Το ICV ενός Web Component το αναγνωρίζει μοναδικά σε σχέση με τα άλλα Web Components στη σελίδα, με την εξαίρεση των ICV που είναι κενά.

4. Ανάθεση μίας υπογραφής (signature) για κάθε Web Component της σελίδας, η οποία στις περισσότερες περιπτώσεις είναι το ICV του Component.

Κατά τη φάση συλλογής δεδομένων, ο αλγόριθμος εκπαίδευσης χρησιμοποιεί τον αλγόριθμο τμηματοποίησης για να συλλέξει τα δεδομένα. Σε τακτά χρονικά διαστήματα ο αλγόριθμος τμηματοποίησης ενεργοποιείται και αποθηκεύεται το index tree για το συγκεκριμένο στιγμιότυπο της σελίδας σε εκείνη τη χρονική στιγμή. Ο σκοπός είναι να συγκεντρωθούν αρκετά δείγματα του index tree για ένα χρονικό διάστημα στο οποίο όλες οι αλλαγές που συμβαίνουν τακτικά στο περιεχόμενο της σελίδας έχουν γίνει. Στα πιο δημοφιλή sites ειδήσεων ή portal, τα οποία ανανεώνουν κάθε μέρα το περιεχόμενό τους το χρονικό διάστημα αυτό είναι 24 ώρες. Σε άλλα site στα οποία οι αλλαγές είναι λιγότερο συχνές, αυτό το χρονικό διάστημα μεγαλύτερο (αυτό εξαρτάται από τη συχνότητα αλλαγής περιεχομένου).

Όταν αυτό το προκαθορισμένο χρονικό διάστημα περάσει, έχουν συλλεχθεί k δείγματα του index tree, όπου $k = \left\lfloor \frac{\Pi εριοδος\_\Delta ειγματοληψιας}{\Delta ιαστημα\_\Delta ειγματοληψιας} \right\rfloor$. Ο αλγόριθμος της εκπαίδευσης έχει συγκεντρώσει όλα τα στοιχεία που χρειάζεται και πρέπει να προχωρήσει τώρα στην ανάλυσή τους.

Στη δεύτερη φάση της διαδικασίας εκπαίδευσης γίνονται κάποιοι υπολογισμοί που σκοπό έχουν να αναγνωρίσουν το περιεχόμενο που μένει σταθερό κατά τη διάρκεια της δειγματοληψίας. Ο αλγόριθμος τμηματοποίησης δημιουργεί μία δομή δεδομένων για κάθε Web Component που αναγνωρίζει κατά τον τεμαχισμό της σελίδας, όπου και αποθηκεύεται το περιεχόμενο του Web Component. Αυτή η δομή ονομάζεται Content Vector (CV- Διάνυσμα Περιεχομένου) και είναι χαρακτηριστικό κάθε στιγμιότυπου ενός Web Component (πράγμα που σημαίνει ότι το CV μπορεί να είναι διαφορετικό για διαφορετικά στιγμιότυπα του ίδιου Web Component). Η αναπαράσταση εγγράφων σαν διανύσματα όρων (term vectors) είναι πολύ δημοφιλής τεχνική σε εφαρμογές ανάκτησης πληροφορίας. Το CV είναι ένα ζεύγος δύο διανυσμάτων, το πρώτο από τα οποία περιέχει τους όρους κειμένου (λέξεις) που υπάρχουν στο καθαρό κείμενο του Web Component, ενώ το δεύτερο περιλαμβάνει τα ονόματα των αρχείων εικόνας που υπάρχουν στον κώδικα του Web Component. Αυτά τα διανύσματα τα συμβολίζουμε T_p και I_p αντίστοιχα.

T_p ={w/ w είναι μία λέξη μέσα στο καθαρό κείμενο του WC_p} (6)
I_p ={z/ z είναι μια εικόνα που περιέχεται στον κώδικα του WC_p } (7)
CV_p =(T_p,I_p) (8)

Υποθέτουμε ότι για τα k δείγματα του index tree, ο αριθμός των Web Component που έχουν επιλεγεί σε κάθε τμηματοποίηση και η δομή των δέντρων δεν αλλάζουν. Χρησιμοποιώντας το ID του κάθε Component, ο αλγόριθμος της εκπαίδευσης ανακτά τα στιγμιότυπα των component και τα CV τους από το σύνολο των k στιγμιότυπων των index tree. Στη συνέχεια συγκρίνει τα k CV του κάθε Component και κρατάει μόνο το περιεχόμενο (κείμενο και ονόματα αρχείων εικόνων) που υπάρχουν σε όλα τα CV. Στο τέλος αυτής της διαδικασίας ο αλγόριθμος έχει κατασκευάσει μία δομή δεδομένων που περιέχει το περιεχόμενό του κάθε component που παρέμεινε σταθερό κατά τη διάρκεια της διαδικασίας

εκπαίδευσης. Αυτή η δομή ονομάζεται ‘Constant Content Vector’ (CCV- Διάνυσμα Σταθερού Περιεχομένου) και είναι χαρακτηριστική ενός Web Component ανεξαρτήτως των στιγμιότυπων του σε διαφορετικές χρονικές στιγμές.

$$CCV_p = \coprod_{t=1}^{k} CV_{p,t} \quad (9)$$

($CV_{p,t}$ είναι το Content Vector του στιγμιότυπου t του Web Component p)

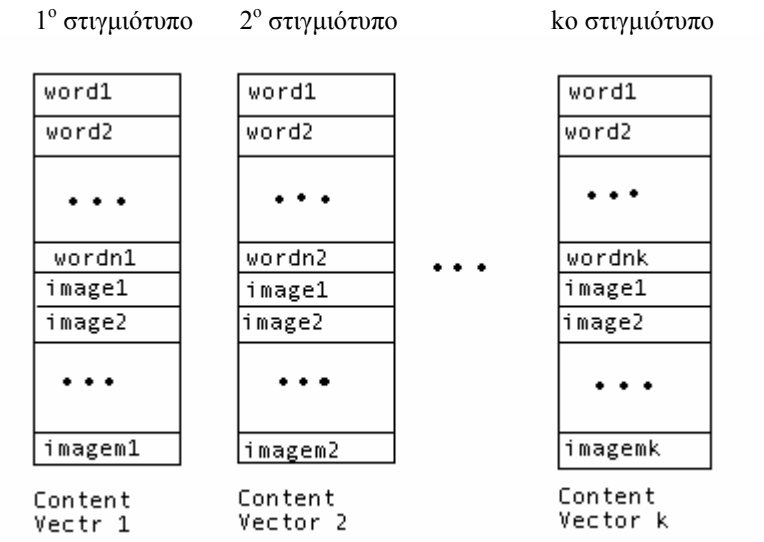
Αν και το βήμα 2 της διαδικασίας εκπαίδευσης παράγει μία δομή που θα μπορούσε να προσδιορίσει ένα Web Component με μεγάλη ακρίβεια, υπάρχουν κάποιες περιπτώσεις όπου το CCV ενός Component δεν είναι αρκετό. Το Διάνυσμα Σταθερού Περιεχομένου (CCV) ενός Web Component παράγεται χρησιμοποιώντας μόνο το περιεχόμενο αυτού του component. Ο σκοπός όμως της διαδικασίας εκπαίδευσης είναι να παράγει αναγνωριστές (identifiers) για όλα τα Components μιας σελίδας, οι οποίοι θα μπορούσαν να χρησιμοποιηθούν για να προσδιορίσουν κάθε Component μοναδικά από τα υπόλοιπα components της σελίδας. Άρα στο βήμα 3 της διαδικασίας εκπαίδευσης όλα τα Web Components συγκρίνονται αμοιβαία. Υπάρχουν δύο αλλαγές που γίνονται στο περιεχόμενο των CCV. Η πρώτη είναι ότι το κείμενο ή οι εικόνες που υπάρχουν δε όλα τα CCV απομακρύνονται. Αυτό γίνεται διότι αυτό το τμήμα του περιεχομένου δεν προσθέτει καμία επιμέρους πληροφορία που να διαφοροποιεί κάποιο Component από τα υπόλοιπα. Η δεύτερη ενέργεια είναι ότι αν το περιεχόμενο ενός CCV περιέχεται ακριβώς μέσα στο περιεχόμενο ενός άλλου CCV, τότε το πρώτο Web Component (και το ICV του, το οποίο είναι η έξοδος του βήματος 3 του αλγόριθμου εκπαίδευσης) σημειώνονται ως αδύναμα (weak). Αυτό σημαίνει ότι το Web Component δεν μπορεί να προσδιοριστεί μοναδικά από ένα identifier που βασίζεται μόνο στο περιεχόμενο. Στο τέλος του βήματος 3 της διαδικασίας εκπαίδευσης, κάθε Web Component έχει ένα CCV (με πιθανώς μειωμένο περιεχόμενο από αυτό που είχε στο βήμα2), το οποίο το προσδιορίζει μοναδικά στο σύνολο των WEB Components της σελίδας. Μοναδική εξαίρεση αποτελούν τα Web Components που έχουν χαρακτηριστεί αδύναμα ή έχουν κενά CCV (προφανώς τα Web Components με κενά CCV είναι και αδύναμα). Ο αναγνωριστής που παράγεται στο βήμα 3 ονομάζεται ‘Instant Content Vector’ (ICV- Αναγνωριστικό ΔιάνυσμαΠεριεχομένου).

$$ICV_p = CCV_p - \coprod_{i=1}^{p\max} CCV_i \quad (10)$$

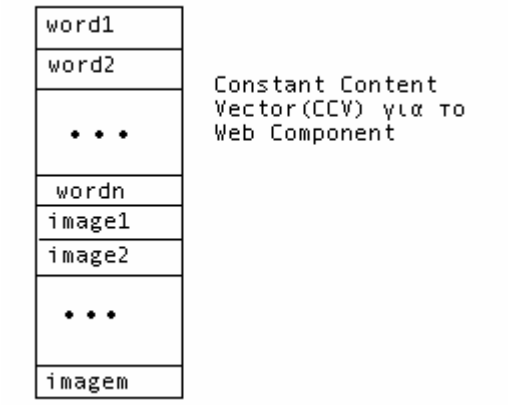
το WC είναι αδύναμο αν (11)

Το βήμα 4 είναι η τελευταία φάση της διαδικασίας εκπαίδευσης. Μέχρι αυτό το σημείο, σχεδόν όλα τα Web Components που περιέχονται μέσα σε μια σελίδα έχουν ένα ICV που μπορεί να χρησιμοποιηθεί σαν υπογραφή για αυτά και να τα προσδιορίζει από τα υπόλοιπα Web Components της σελίδας. Η μοναδική εξαίρεση είναι components, τα οποία έχουν σημειωθεί ως αδύναμα ή των οποίων το CCV είναι κενό (δηλαδή το περιεχόμενό του άλλαξε εξ’ ολοκλήρου κατά τη διάρκεια της εκπαίδευσης). Στο βήμα 4 της διαδικασίας εκπαίδευσης δημιουργείται η υπογραφή (signature) των components. Αυτή είναι το ICV για όλα τα components εκτός από αυτά που αναφέρθηκαν παραπάνω. Σε αυτά ανατίθεται ως υπογραφή μια άλλη δομή δεδομένων που βασίζεται στη σχετική θέση των component μέσα στο index tree και

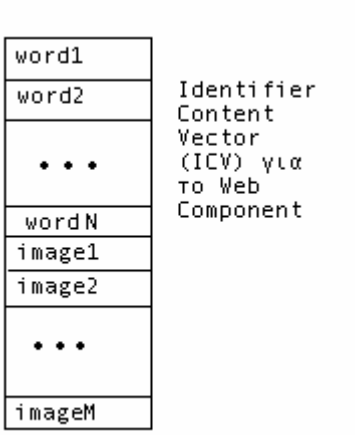
στο μέγεθος του περιεχομένου τους. Ο τρόπος που κατασκευάζεται αυτή η δομή δίνεται με ένα παράδειγμα.



Σύγκριση των k Content Vector



Σύγκριση με τα CVs των υπόλοιπων WCs της σελίδας



Εύρεση του ICV ενός Web Component κατά τη φάση της εκπαίδευσης

3.1.1.2 Διαδικασία ανανέωσης

Μετά το τέλος της διαδικασίας εκπαίδευσης το σύστημα έχει γνώση σχετικά με τι έξοδο πρέπει να περιμένει από την κάθε τμηματοποίηση των σελίδων. Η πλήρης δομή της σελίδας είναι διαθέσιμη, αφού είναι αποθηκευμένη σε κάθε ένα από τα k index trees που έχουν συλλεχθεί κατά τη φάση της εκπαίδευσης (εννοείται ότι δε χρειαζόμαστε όλες αυτές τις πληροφορίες και μόνο το τελευταίο index tree αποθηκεύεται, ενώ όλα τα άλλα σβήνονται για οικονομία μνήμης). Επίσης για κάθε σελίδα υπάρχει ένα ευρετήριο σελίδας (page index) το οποίο περιέχει τις υπογραφές (αναγνωριστές) του κάθε component. Το ευρετήριο σελίδας περιέχει τόσο το ID του κάθε component (που εξαρτάται από τη θέση του μέσα στο index tree), όσο και την υπογραφή του, η οποία το διαφοροποιεί από όλα τα άλλα components. Αυτό συμβαίνει επειδή το ID ενός component μπορεί να αλλάξει αν υπάρχουν σε κάποιο στιγμιότυπο αλλαγές στη δομή της σελίδας, ενώ η υπογραφή παραμένει σταθερή.

Ο ρόλος της φάσης ανανέωσης είναι να ανανεώνει τα αποθηκευμένα δεδομένα με τα τελευταία στιγμιότυπα των Web Components (html κώδικα, εικόνες κ.α.). Το χρονικό διάστημα μεταξύ κάθε ανανέωσης εξαρτάται από τη συχνότητα αλλαγής του περιεχομένου. Αυτό το χρονικό διάστημα είναι σταθερό, αλλά σε μελλοντικές εκδόσεις θα κυμαίνεται ανάλογα με τη συχνότητα αλλαγής του περιεχομένου στο συγκεκριμένο χρονικό σημείο.

Η όλη διαδικασία της φάσης ανανέωσης έχει πολλά κοινά στοιχεία με τη φάση εκπαίδευσης. Η σελίδα κατεβαίνει διαρκώς στον υπολογιστή (ή υπολογιστές) που είναι υπεύθυνος για τη δημιουργία των Web Components, αναλύεται (με τη βοήθεια του αλγόριθμου τμηματοποίησης) και υπολογίζονται τα Web Components της σελίδας. Στη συνέχεια, οι τελευταίες εκδόσεις των Web Components αποθηκεύονται στο Web Server της υπηρεσίας για να χρησιμοποιηθούν από τους χρήστες για τη δημιουργία των προσωπικών τους σελίδων. Οι κύριες διαφορές με τη φάση της εκπαίδευσης είναι ότι:

- Ο αλγόριθμος τμηματοποίησης έχει τρία βήματα.
- Δε γίνονται υπολογισμοί για τις υπογραφές των Web Components, αφού αυτοί οι υπολογισμοί γίνονται κατά τη φάση εκπαίδευσης και στη συνέχεια θεωρούνται σταθερές.

- Στη φάση της εκπαίδευσης τα Web Components δεν αποθηκεύονται στο Web Server.

Ο αλγόριθμος εκπαίδευσης, όπως είδαμε, παράγει στο $4^{ο}$ βήμα το index tree του τελευταίου στιγμιότυπου της σελίδας και σημειώνει κάποιους κόμβους ως Web Components. Η διαδικασία εκπαίδευσης χρησιμοποιεί μια συλλογή από διαφορετικά στιγμιότυπα των index tree για να αποκτήσει γνώση σχετικά με το πώς πρέπει να γίνεται ο τεμαχισμός της σελίδας. Πρέπει όμως κατά τη διάρκεια της φάσης εκπαίδευσης να μη συμβαίνουν αλλαγές στη δομή της σελίδας ή στον αριθμό των Web Components που υπολογίζονται από την τμηματοποίηση. Αλλά γενικά αν και σπάνιο μπορεί να συμβούν αλλαγές κατά τη φάση ανανέωσης. Σε αυτή την περίπτωση ο αλγόριθμος τμηματοποίησης πρέπει να διορθώσει τα προβλήματα που προκαλούνται από τις αλλαγές χρησιμοποιώντας τις πληροφορίες από τη φάση εκπαίδευσης. Αυτό γίνεται στο $5^{ο}$ βήμα του αλγόριθμου τμηματοποίησης.

Στο βήμα αυτό, ο αλγόριθμος τμηματοποίησης παίρνει ως είσοδο την έξοδο του βήματος 4,δηλαδή το index tree και το page index και ελέγχει αν υπάρχουν διαφορές στη δομή της σελίδας ή στον αριθμό των Web Components. Αν και είναι πιθανόν αυτός ο έλεγχος να γίνει κατευθείαν πάνω στα index trees, γίνεται ελέγχοντας για διαφορές στο πεδίο του ID στο page index του τελευταίου που παρήχθη στη διαδικασία εκπαίδευσης. Το page index των στιγμιότυπων των σελίδων περιέχει το Content Vector των Web Component στη θέση της υπογραφής, αφού η υπογραφή είναι χαρακτηριστικό όλων των στιγμιότυπων ενός Web Component, ενώ το Content Vector χαρακτηρίζει ένα στιγμιότυπο μόνο. Αν δεν εμφανιστούν αλλαγές (το οποίο είναι ο κανόνας στις περισσότερες περιπτώσεις), ο αλγόριθμος συνεχίζει στο βήμα 6. Διαφορετικά μία ειδική διαδικασία ακολουθείται, η οποία αποσκοπεί στη διόρθωση των προβλημάτων που προκαλούνται από τις διαφορές.

Ο αλγόριθμος, ο οποίος διορθώνει τον υπολογισμό των Web Components παρουσιάζεται στις παρακάτω παραγράφους. Είναι ο πιο περίπλοκος που υλοποιήθηκε για τη λειτουργία του συστήματος, αφού πολλές διαφορετικές καταστάσεις μπορεί να οδηγήσουν στην ενεργοποίησή του. Μερικές από αυτές είναι:

- Το περιεχόμενο ενός σύνθετου (complex) Web Component (δηλαδή ένα Component που δεν είναι φύλλο) μεγάλωσε αρκετά σε μέγεθος στο τελευταίο στιγμιότυπο της σελίδας και ο αλγόριθμος τμηματοποίησης το χώρισε σε περισσότερα από ένα Web Components. Αυτό έχει σαν συνέπεια το index tree να έχει την ίδια δομή με την αναμενόμενη, αλλά με μεγαλύτερο αριθμό Web Components.
- Το μέγεθος του κειμένου αρκετών Web Components, τα οποία έχουν κοινό πρόγονο μειώθηκε και ο αλγόριθμος τμηματοποίησης επέλεξε τον πρόγονο ως Web Component. Αυτή η κατάσταση οδηγεί σε index tree που έχει την επιθυμητή δομή, αλλά μικρότερο αριθμό Web Components.
- Ένα νέο TABLE tag προστέθηκε κάπου στη δομή της σελίδας ή κάποιο TABLE tag αφαιρέθηκε. Αυτή η κατάσταση οδηγεί σε index tree με διαφορετική δομή. Επίσης, τα IDs των κόμβων από το σημείο αυτό και μετά είναι διαφορετικά. Η κατάσταση περιπλέκεται περισσότερο αν πάνω από ένα TABLE tag προστέθηκαν ή αφαιρέθηκαν.
- Αμοιβαία αλλαγή στη θέση δύο TABLE tags
- Οποιοσδήποτε συνδυασμός των παραπάνω καταστάσεων.

Αν και είναι δυνατό να δημιουργήσουμε έναν αλγόριθμο που θα διορθώνει τις αρκετά περίπλοκες καταστάσεις (όταν έχουμε πολύ μεγάλες αλλαγές, όπως η ριζική

ανανέωση της κεντρικής σελίδας ενός portal), ο αλγόριθμος που παρατίθεται παρουσιάζει πολύ καλή συμπεριφορά στις πιο πολλές περιπτώσεις.

Αλγόριθμος Διόρθωσης Της Τμηματοποίησης

```
if (page index of the page=latest instance page index)
  if (WCcount in the page index from training==WCcount in
                                        the instabce page index)
  //έχουμε τον ίδιο αριθμό Web Components
  {check the signature contained in the page index with the
  Content Vectors contained in the instance page index
  if(signature match)
  {
                    //Πιθανότατα κάποια μικρή αλλαγή στη δομή της σελίδας
                    //άλλαξε τα ID των Web Components που περιέχονται σε
                       //αυτή. Παρόλα ταύτα, οι υπογραφές στο page index
                       //ταιριάζουν με τα Content Vector, συνεπώς τα Web
                        //Components έχουν υπολογιστεί όπως θα έπρεπε.
                        Extract (mark for extraction) the Web Components
                                                              based on their signature
                                                  }
  else
  {
                      extract all the Web Components that their CVs match
                                                  with signatures in the page index. Extract all the
                                                  rest WCs based on their order of appearance in the
                                                                        page index.
  }
  }

  else
  {
                    //Διαφορετικός αριθμός Web Components. Η πιο πιθανή
   //κατάσταση είναι ότι η δομή του index tree παρέμεινε
   //η ίδια, αλλά ο αλγόριθμος τμηματοποίησης δεν υπολόγισε
                 //τα Web Components σωστά λόγω των αλλαγών στο μέγεθός τους
                       if(index tree structure from training matches with the
                                                                  instance index tree)
  {
                       //Εχουμε τα ίδια index tree. αγνοούμε τον υπολογισμό
                       //των Web Components στο τελευταίο στιγμιότυπο και
                      //εξάγουμε τα σωστά Web Components βασιζόμενοι στη
                                  //θέση τους στο index tree.
                          Extract Web Components based on their IDs.
  }
  else
  {
                       //υπάρχουν αλλαγές στη δομή του index tree επίσης.Ο
                    //αλγόριθμος προσπαθεί να επαναυπολογίσει τα Components
                   //ώστε να είναι όσα αναμένεται.Ο αλγόριθμος τμηματοποίησης
                    //θα εκτεστεί ξανά αν υπάρχουν ακόμα αλλαγές,ο αλγόριθμος
```

```
                    //διόρθωσης της τμηματοποίησης θα εκτελεστεί ξανά. Ο στόχος
                        //είναι αυξάνοντας ή μειώνοντας την παράμετρο u του
                    //αλγορίθμου τμηματοποίησης να έχουμε έναν υπολογισμό όπου
                    //ο αριθμός των Web Components θα είναι αυτός που αναμένεται
                        //και ο αλγόριθμος δίορθωσης της τμηματοποίησης θα
                    //ακολουθήσει το πρώτι κλαδί (ίδιος αριθμός Web Components).
                       //παρόλα ταύτα, είναι δυνατο να δημιουργηθεί ένα άπειρο
                        //loop και έτσι χρησιμοποιείται ένας καθολικός μετρητής
                        //μεταξυ των εκτελέσεων του αλγόριθμου διόρθωσης.
      Counter++;
  if (Counter<4)
  {
                          if(WCcount in the instance>WCcount from training)
  {
                            //πιθανώς ο αριθμός των WC είναι μεγαλύτερος
                             //εξαιτίας μεγάλων WC που διαχωρίστηκαν σε
                            //μικρότερα, εξαιτίας αύξησης στο μέγεθός τους
Increase the u parameter of the fragmatation
algorithm and recalculate the index tree
and the Web Components.
}
else
  {
  //πιθανώς ο αριθμός των WC είναι μικρότερος
  //εξαιτίας μικρών WC που συνενώθηκαν σε
  //μεγαλύτερα,εξαιτίας μείωσης στο μέγεθός τους
  Decrease the u parameter of the fragmatation
algorithm and recalculate the index tree
and the Web Components.
  }
  }
  else
  {
  //Η κατάσταση δεν διορθώθηκε με τις επανηλημένες
  //εκτελέσεις του αλγόριθμου τμηματοποίησης. Είναι
  //η πιο δύσκολη κατάσταση, αφού υπάρχει διαφορετική
  //δομή της σελίδας και διαφορετικός αριθμός
  //υπολογισμένων Web Components.
  Get the initial fragmentation (with the default value
  of the u parameter). Extract all the Web
  Components that can be extracted based on the
  Content Vectors. Extract all the remaining
  Web Components based on their order of appearance
  and their content size (closet match).
  }
    }
     }
```

Πρέπει να σημειωθεί εδώ ότι ο αλγόριθμος διόρθωσης που παρουσιάστηκε παραπάνω χρησιμοποιεί τα Content Vectors των στιγμιότυπων των Web Components για τις συγκρίσεις με τις υπογραφές των Web Components που έχουν

επιλεγεί κατά τη φάση της εκπαίδευσης. Παρόλα ταύτα αυτό δεν είναι δυνατό αν η σελίδα περιέχει Components χωρίς σταθερό περιεχόμενο ή με αδύναμα ICVs. Αυτά τα Components έχουν υπογραφή διαφορετικού είδους, η οποία δεν είναι συγκρίσιμη με τα Content Vector των στιγμιότυπων. Όποτε συμβαίνει μια τέτοια κατάσταση, ο αλγόριθμος συγκρίνει πρώτα όλα τα components που έχουν ICV σαν υπογραφή. Στη συνέχεια αναγνωρίζει τα υπόλοιπα με βάση τη σχετική τους θέση και το μέγεθός τους (αυτές είναι οι πληροφορίες που περιέχει η υπογραφή των Components της δεύτερης κατηγορίας).

Ένα άλλο ζήτημα είναι το πώς γίνεται η σύγκριση μεταξύ των Content Vector των στιγμιότυπων των Web Component και των Web Components των Component που βρίσκονται στο page index (για τα component που χρησιμοποιούν ICV ως υπογραφή). Χρησιμοποιείται μία βασική ιδιότητα του ICV (Αναγνωριστικό Διάνυσμα Περιεχομένου) η οποία προκύπτει από τον τρόπο κατασκευής του. Το περιεχόμενο του ICV είναι υποσύνολο του περιεχομένου του CCV, το οποίο κατασκευάζεται συγκρίνοντας αρκετά στιγμιότυπα των Content Vectors και αφαιρώντας το περιεχόμενο που δεν υπάρχει σε όλα τα στιγμιότυπα. Επίσης προσδιορίζει μοναδικά ένα Web Component, δηλαδή δεν υπάρχουν περισσότερα από ένα Web Component που το περιεχόμενο των Content Vector τους είναι υπερσύνολο του περιεχομένου του ίδιου ICV (εκτός και αν ένα ICV είναι αδύναμο, οπότε δε χρησιμοποιείται ως υπογραφή). Ο αλγόριθμος διόρθωσης της τμηματοποίησης χρησιμοποιεί αυτή την ιδιότητα των ICV για τις συγκρίσεις που έχει να κάνει. Συγκεκριμένα, ελέγχει ένα ICV με όλα τα CV των στιγμιότυπων των Web Component. Το πρώτο στιγμιότυπο του οποίου το CV (το περιεχόμενό του) είναι υπερσύνολο του ICV είναι στιγμιότυπο του συγκεκριμένου Web Component. Υπάρχει μια ακραία περίπτωση όπου αυτός ο έλεγχος μπορεί να παρουσιάσει περισσότερα από ένα αποτελέσματα. Συμβαίνει αν σε ένα στιγμιότυπο ενός Web Component έχει προστεθεί όλο το περιεχόμενο του ICV ενός άλλου component. Όμως αυτή η περίπτωση είναι απίθανο ως και αδύνατο να συμβεί και έτσι δεν θα ασχοληθούμε ιδιαίτερα λαμβάνοντας ειδικά μέτρα για την αντιμετώπισή της.

Όταν ο αλγόριθμος διόρθωσης της τμηματοποίησης τελειώσει ($5^{ο}$ βήμα του αλγόριθμου τμηματοποίησης κατά τη φάση της ανανέωσης) όλα τα στιγμιότυπα των Web Components έχουν βρεθεί στο index tree. Στο $6^{ο}$ βήμα εξάγονται και αποθηκεύονται στο Web Server. Το index tree διασχίζεται και για κάθε σημειωμένο κόμβο ο αλγόριθμος ο αλγόριθμος ακολουθεί τον σύνδεσμο στον αντίστοιχο HTML tree και ανακτά τον HTML κώδικα. Στη συνέχεια γίνονται κάποιες αλλαγές στον κώδικα (οι οποίες αναφέρονται στο τμήμα της σύνθεσης της προσωπικής σελίδας του χρήστη) και αποθηκεύει τον παραγόμενο κώδικα σαν απλό κείμενο στο Web Server. Αυτός ο κώδικας χρησιμοποιείται για την παρουσίαση του Web Component στους χρήστες. Κάθε Web Component προσδιορίζεται στο web Server από τον αύξοντα αριθμό του στο page index της αντίστοιχης σελίδας και το όνομα της σελίδας.

Εδώ θα μπορούσαμε να εισάγουμε και δύο νέες έννοιες που θα μας βοηθήσουν να αυξήσουμε την ταχύτητα του συστήματος κατά τη φάση της ανανέωσης. Εισάγουμε το Point System και το Word Weight.

Το Point System είναι μία μεταβλητή που χρησιμοποιείται για να αυξηθεί η ταχύτητα στα Web Components καθώς τα WC μεταξύ τους μέσα στην προσωποιημένη σελίδα έχουν διαφορετική βαρύτητα ανάλογα με τον πλήθος των παρουσιών ή απουσιών

μέσα σε αυτή. Έτσι ένα WC το οποίο εμφανίζεται αρκετά συχνά στην σελίδα μας για οικονομία χρόνου θα μπορούσε να αποθηκεύεται στη μνήμη ή στη βάση δεδομένων.
Είναι εύλογο ότι όσο πιο συχνά εμφανίζεται τόσο μεγαλύτερο βάρος έχει καθώς επίσης και ότι η μεγάλη απουσία του θα πρέπει να περιορίζει τη βαρύτητα του.
Για αυτό το λόγο θεσπίσαμε το Point System. Το Point System είναι η μεταβλητή που θα μας καθορίζει την βαρύτητα ενός Web Component και την ορίζουμε ως:

$$PS(m,n) = \frac{n(m+n)}{m^2+1}$$

όπου n: πλήθος παρουσίας του WC και
m: πλήθος απουσίας του WC.

Ο αριθμητής μας δείχνει ότι το PS είναι ανάλογο του πλήθους των εμφανίσεων και επίσης ανάλογο του πλήθους των φορών που έχουμε αναζητήσει το WC έτσι ώστε να αυξάνει το βάρος ενός WC όταν αυτό μας έχει απασχολήσει πολύ. Ο παρανομαστής μας δείχνει ότι το PS είναι αντιστρόφως ανάλογο του τετραγώνου των απουσιών του WC. Το «+1» στον παρανομαστή το βάζουμε για να έχει νόημα ο PS και στην περίπτωση που δεν έχουμε απουσίες, έτσι ώστε ο παρανομαστής να μην είναι 0.
Θεωρήσαμε σωστό όταν έχουμε αύξηση του PS αυτό να γίνεται πιο «αργά» από όταν έχουμε μείωση διότι ένα WC το οποίο δεν το συναντήσαμε διαδοχικές φορές υπάρχει μεγάλη πιθανότητα να μην το ξανασυναντήσουμε στο μέλλον και γι αυτό το λόγο πρέπει να πέσει γρήγορα η βαρύτητα του έτσι ώστε να μην μας καθυστερεί έναντι των άλλων τα οποία έχουν διαρκή εμφάνιση. Από την άλλη δεν θα ήταν ιδιαίτερα σοφό να μηδενίσουμε το PS του επειδή σημειώθηκαν απουσίες διότι στο μέλλον μπορεί να το ξαναχρειαστούμε.
Ας δούμε όμως μερικές εφαρμογές του τύπου.

- Για n=10 και m=3 έχουμε PS=13
- Για n=5 και m=0 έχουμε PS=25
- Για n=10 και m=2 έχουμε PS=26
- Για n=3 και m=0 έχουμε PS=9
- Για n=20 και m=6 έχουμε PS=14,05

Άρα βλέπουμε ότι ένα WC το οποίο έχει εμφανιστεί 5 φορές χωρίς απουσία έχει περίπου την ίδια βαρύτητα με ένα που έχει εμφανιστεί 10 φορές με 2 απουσίες, όπως περίπου θα περιμέναμε ενώ ένα που έχει κλείσει 20 εμφανίσεις και 6 απουσίες έχει πολύ μικρότερη βαρύτητα από αυτά επειδή οι 6 απουσίες μας δείχνουν ότι υπάρχει σημαντική πιθανότητα να μην μας απασχολήσει στο μέλλον.
Αναλύοντας το PS τέλος βλέπουμε ότι για πεπερασμένο πλήθος παρουσιών και άπειρες απουσίες το PS τείνει στο 0 ενώ για πεπερασμένο πλήθος απουσιών και άπειρες παρουσίες τείνει στο άπειρο όπως αναμέναμε.

Στην ίδια λογική βρίσκεται και το Word Weight που αξιολογεί τις λέξεις κλειδιά ανάλογα με το ‘βάρος’ τους.

Αρκετές φορές στην αναζήτηση ενός Web Component συναντούνται κάποιες λέξεις αρκετές φορές (κοινώς μπορούν να χαρακτηριστούν λέξεις με ιδιαίτερη βαρύτητα). Θα ήταν αρκετά χρήσιμο να μπορούμε να αξιολογήσουμε αυτές τις λέξεις διότι μια γρήγορη αναζήτησή τους θα μας οδηγούσε σε μια γρήγορη αναζήτηση του Web Component που μας ενδιαφέρει. Εδώ θα προσπαθήσουμε να βρούμε έναν τρόπο αξιολόγησης αυτών των λέξεων ανάλογα με το πόσο συχνά εμφανίζονται στο κείμενο που μας αφορά με σκοπό να κατασκευάζουμε πιο εύκολα την signature του κάθε Web

Component. Για την κατασκευή της signature θα ήταν αρκετά βολικό να λάβουμε υπόψη το WW (αναφερόμαστε μετά εκτενέστερα σε αυτό). Θα μπορούσαμε να θεωρήσουμε για παράδειγμα ως signature ενός WC τις 10 λέξεις του Web Component με τη μεγαλύτερη τιμή του WW. Σημαντική τροχοπέδη σε αυτό μας το έργο είναι κάποιες κοινές λέξεις χωρίς ιδιαίτερη σημασία (άρθρα, σύνδεσμοι, μόρια και αντωνυμίες) τα οποία εμφανίζονται πολύ συχνά μέσα στο κείμενο χωρίς όμως να αξίζουν την ανάλογη βαρύτητα οι οποίες είναι γνωστές σαν stop words και οι οποίες «κόβονται» στις περισσότερες μηχανές αναζήτησης.
Άρα και στη σελίδα λογικό θα ήταν αυτές οι λέξεις να μην λαμβάνονται υπόψη. Ανάλογα με το interface του κάθε user το site θα ενημερώνεται για τις stop words των γλωσσών που είναι γραμμένες οι σελίδες που κατεβάζει ο χρήστης και αυτόματα δεν θα τις εξαιρεί όσες φορές και αν τις συναντήσει στην πορεία. Με άλλα λόγια η μεταβλητή WW (για την οποία θα αναφερθούμε πιο κάτω) δεν θα έχει νόημα για το πλήθος εμφανίσεων και απουσιών των λέξεων αυτών. Αυτή την εξαίρεση των stop words θα την αναλάβει η βάση δεδομένων που θα τρέχει στο server και αναλύοντας την «καταγωγή» των ιστοσελίδων που «σερφάρει» ο κάθε χρήστης θα βρίσκει τις stop words της κάθε γλώσσας για να τις διαχειριστεί ανάλογα.
Για τον διαχωρισμό της βαρύτητας της κάθε λέξης θα μπορούσαμε να χρησιμοποιήσουμε ανάλογα τον τύπο του Point System:

$$PS(m,n) = \frac{n(m+n)}{m^2+1}$$

όπου n: πλήθος παρουσίας του WC και
m: πλήθος απουσίας του WC.

Τη νέα αυτή μεταβλητή που θα εισάγουμε θα την ονομάσουμε WW(από το word weight) και η οποία θα μας βοηθάει να βρούμε το signature κάθε Web Component. (Με τον όρο signature εννοούμε κάποια χαρακτηριστικά στοιχεία για την ανεύρεση του Web Component και στη συγκεκριμένη περίπτωση είναι λέξεις που το αποτελούν). Άρα ο τύπος θα είναι :

$$ww = \frac{k(k+l)}{l^2+1}$$

όπου k: πλήθος παρουσίας της κάθε λέξης και
l: πλήθος απουσίας της κάθε λέξης.

Η μεταβλητή WW σχεδιάστηκε έτσι ώστε ο αριθμητής να μεγαλώνει τετραγωνικά όσο δεν έχουμε απουσίες (l=0) έτσι ώστε να αυξάνει με ένα συγκεκριμένο ρυθμό αλλά κρίθηκε σκόπιμο να μειώνεται η τιμή της το ίδιο γοργά όταν σημειώνονται απουσίες της λέξης. Ο παράγοντας (k+l) έχει προστεθεί ώστε το συνολικό διάστημα ενασχόλησης με την συγκεκριμένη λέξη να επηρεάζει την βαρύτητά της.
Ο αριθμητής μας δείχνει ότι το WW είναι ανάλογο του πλήθους των εμφανίσεων και επίσης ανάλογο του πλήθους των φορών που έχουμε αναζητήσει τη λέξη έτσι ώστε να αυξάνει το βάρος μίας λέξης όταν αυτή μας είναι χρήσιμη στην αναζήτηση του Web Component. Ο παρανομαστής μας δείχνει ότι το WW είναι αντιστρόφως ανάλογο του τετραγώνου των απουσιών της λέξης. Το «+1» στον παρανομαστή το βάζουμε για να έχει νόημα ο WW και στην περίπτωση που δεν έχουμε απουσίες, έτσι ώστε ο παρανομαστής να μην είναι 0.
Θεωρήσαμε σωστό όταν έχουμε αύξηση του WW αυτό να γίνεται πιο «αργά» από όταν έχουμε μείωση διότι μία λέξη την οποία δεν τη συναντήσαμε σε διαδοχικές αναζητήσεις υπάρχει μεγάλη πιθανότητα να μην το ξανασυναντήσουμε στο μέλλον

και γι αυτό το λόγο πρέπει να πέσει γρήγορα η βαρύτητα της έτσι ώστε να μην μας καθυστερεί έναντι των άλλων οι οποίες παρουσιάζουν διαρκή εμφάνιση. Από την άλλη δεν θα ήταν ιδιαίτερα σοφό να μηδενίσουμε τη βαρύτητά της επειδή σημειώθηκαν απουσίες διότι στο μέλλον μπορεί να την εντοπίσουμε πάλι.

Ας δούμε όμως μερικές εφαρμογές του τύπου.

- Για k=10 και l=3 έχουμε WW=13
- Για k=5 και l=0 έχουμε WW=25
- Για k=10 και l=2 έχουμε WW=26
- Για k=3 και l=0 έχουμε WW=9
- Για k=20 και l=6 έχουμε WW=14,05

Άρα βλέπουμε ότι μία λέξη η οποία έχει εμφανιστεί 5 φορές χωρίς απουσία έχει περίπου την ίδια βαρύτητα με εκείνη που έχει εμφανιστεί 10 φορές με 2 απουσίες, όπως περίπου θα περιμέναμε ενώ μία που έχει κλείσει 20 εμφανίσεις και 6 απουσίες έχει πολύ μικρότερη βαρύτητα από τις άλλες επειδή οι 6 απουσίες μας δείχνουν ότι υπάρχει σημαντική πιθανότητα να μην μας απασχολήσει στο μέλλον.
Αναλύοντας το WW τέλος βλέπουμε ότι για πεπερασμένο πλήθος παρουσιών και άπειρες απουσίες τείνει στο 0 ενώ για πεπερασμένο πλήθος απουσιών και άπειρες παρουσίες τείνει στο άπειρο όπως αναμέναμε.
Έτσι με αυτό τον τύπο έχουμε τη δυνατότητα να αξιολογούμε το βάρος μιας λέξης. Το σύστημα δηλαδή εντοπίζει τις απουσίες και τις παρουσίες μιας λέξης, εφαρμόζει τον τύπο του WW και ανάλογα με τα αποτελέσματα μπορεί να ψάξει με την ανάλογη βαρύτητα στο intenet για sites τα οποία περιέχουν αυτή τη λέξη και είναι πιθανό να είναι Web Components τα οποία θέλουμε να εντάξουμε στην προσωποποιημένη μας σελίδα.

3.2 ΔΗΜΙΟΡΓΙΑ ΠΡΟΣΩΠΙΚΗΣ ΣΕΛΙΔΑΣ

Η ανάλυση και τμηματοποίηση μιας σελίδας στοχεύει στο να έχουμε πάντα τα τελευταία στιγμιότυπα των Web Components που αποτελούν τις σελίδας που προσφέρονται στους χρήστες για τη δημιουργία της προσωπικής τους σελίδας. Το λογισμικό που υλοποιεί την τεχνική της τμηματοποίησης αποθηκεύει τον HTML κώδικα, τις εικόνες, τα javascript αρχεία και όλα τα άλλα αρχεία της σελίδας στον υπολογιστή που παρέχει την υπηρεσία στους χρήστες.

Η φάση 2 στοχεύει στο χρήστη. Κατά τη φάση αυτή κατασκευάζεται μια λίστα των components που κάθε χρήστης επιθυμεί να συμπεριλάβει στην προσωπική του σελίδα. Μέσω ενός εύχρηστου web interface ο χρήστης μπορεί να δημιουργήσει αυτή τη λίστα ή να την τροποποιήσει. Χρησιμοποιώντας μια ειδική σελίδα του site του συστήματος, ο χρήστης επιλέγει κάποιο από τα sites που έχουν αναλυθεί από το σύστημα. Στη συνέχεια μεταφέρεται σε μία σελίδα όπου όλα τα Web Components παρουσιάζονται και ο χρήστης μπορεί να επιλέξει ποία από αυτά επιθυμεί να βλέπει στην προσωπική του σελίδα. Όταν ο χρήστης τελειώσει τις επιλογές του με τη συγκεκριμένη σελίδα, μπορεί να μεταφερθεί πίσω στην αρχική και να επιλέξει ένα άλλο site. Το αποτέλεσμα της φάσης 2 είναι η δημιουργία μιας λίστας με τα επιλεγμένα Web Components για κάθε χρήστη που βρίσκεται στη βάση δεδομένων. Αυτή η λίστα χρησιμοποιείται όταν ο χρήστης επιλέγει να δει την προσωπική του σελίδα.

3.3 ΣΥΝΘΕΣΗ ΠΡΟΣΩΠΙΚΗΣ ΣΕΛΙΔΑΣ

Η φάση 3 περιλαμβάνει μία διαδικασία που εκτελείται κάθε φορά που ένας χρήστης επιλέγει να δει την προσωπική του σελίδα. Εκτελείται από ένα script στο web server του παροχέα υπηρεσίας. Αυτό το script ελέγχει τη βάση δεδομένων και ανακτά τη λίστα με τα επιλεγμένα components του χρήστη που έχει ζητήσει να παρουσιάζονται στην προσωπική του σελίδα. Στη συνέχεια ανακτά από τον Web Server (που είναι αποθηκευμένα) τα τελευταία στιγμιότυπα των επιλεγμένων components και χρησιμοποιεί τον κώδικά τους για να συνθέσει την προσωπική σελίδα του χρήστη.

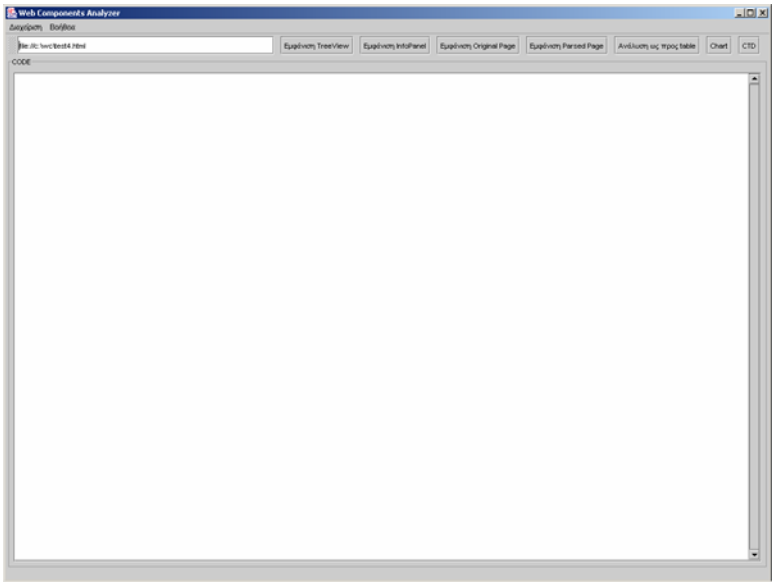
Πρέπει να σημειωθεί εδώ ότι κατά τη διάρκεια της σύνθεσης της προσωπικής σελίδας του χρήστη μία ειδικά διαδικασία πρέπει να ακολουθηθεί για τα Web Components που προέρχονται από σελίδες που χρησιμοποιούν CSS (Cascading StyleSheets). Σε αυτή τη διαδικασία τα αρχεία των stylesheets των σελίδων προστίθενται στο αρχείο των stylesheet της προσωπικής σελίδας. Όταν η σύνθεση της προσωπικής σελίδας έχει τελειώσει, το αρχείο CSS περιέχει όλα τα style που περιέχονται στις σελίδες προέλευσης των components στη φάση ανανέωσης κατά την ανάλυση και τμηματοποίηση της σελίδας πραγματοποιείται μία μετατροπή στον κώδικα των components και των stylesheet. Πιο συγκεκριμένα, όλα τα style μετονομάζονται ώστε να μην υπάρχει περίπτωση να υπάρξουν δύο style με το ίδιο όνομα που προέρχονται από διαφορετικές σελίδες. Ένα style που ονομάζεται 'X' και χρησιμοποιείται στην 'Y' μετονομάζεται και το νέο του όνομα είναι 'Y_X'. Με αυτό τον τρόπο αποφεύγουμε συγκρούσεις μεταξύ των style από διαφορετικές σελίδες. Μια παρόμοια διαδικασία ακολουθείται για τα αρχεία javascript.

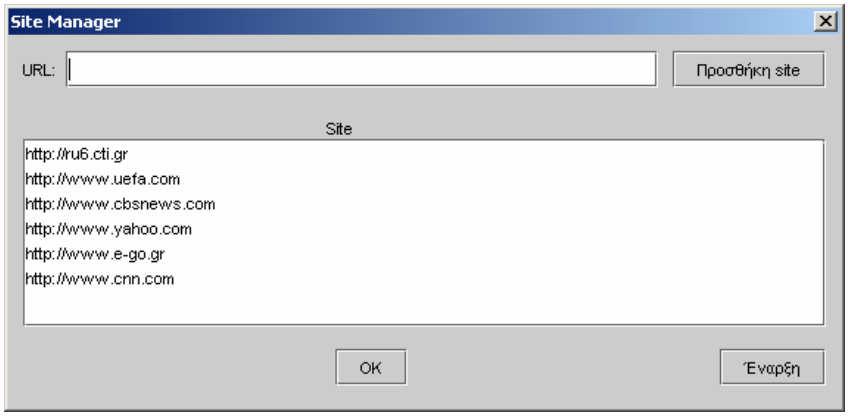
3.4 ΖΗΤΗΜΑΤΑ ΥΛΟΠΟΙΗΣΗΣ

Τρία είναι τα κύρια υποσυστήματα λογισμικού που εμπλέκονται στη λειτουργία της τεχνικής των Web Components. Αυτά είναι το εργαλείο «Web Component Analyzer», η μηχανή προσωποποίησης («Personalization Engine»), καθώς και η βάση δεδομένων του συστήματος.

3.4.1 WEB COMPONENTS ANALYZER

Το «Web Components Analyzer» είναι ένα εργαλείο λογισμικού, το οποίο δουλεύει ανεξάρτητα από τον Web Server του συστήματος. Ο σκοπός του είναι να αναλύει συνεχώς τις σελίδες οι οποίες χρησιμοποιούνται για την εξαγωγή των Web Components και να ανανεώνει τον κώδικα, τις εικόνες, κτλ που είναι αποθηκευμένα στο File System του Web Server, καθώς και τις πληροφορίες που είναι αποθηκευμένες στη βάση δεδομένων. Η κεντρική οθόνη και το πλαίσιο διαλόγου για την εισαγωγή νέων σελίδων προς ανάλυση φαίνονται παρακάτω.





Το εργαλείο αυτό είναι φτιαγμένο με java. Η java προσφέρει ένα ευέλικτο περιβάλλον προγραμματισμού και επιτρέπει στα προγράμματα να εκτελούνται σε πληθώρα λειτουργικών συστημάτων και αρχιτεκτονικών, χωρίς καν να χρειάζεται να επαναμεταγλωτιστούν. Το "Web Component Analyzer" πρέπει να έχει πρόσβαση μέσω του δικτύου στο Web Server της υπηρεσίας και στη Βάση Δεδομένων (φυσικά εννοείται ότι μπορεί να εκτελεστεί και στον υπολογιστή).

Έκτος από τη συνεχή ανάλυση και ανανέωση των σελίδων και των Web Components, αυτό το εργαλείο μας επιτρέπει τον πειραματισμό με σελίδες και την οπτικοποίηση των αποτελεσμάτων.

3.4.2 PERSONALIZATION ENGINE

Η μηχανή προσωποποίησης αποτελείται από ένα σύνολο από σελίδες και μηχανισμούς γραμμένα για την πλατφόρμα JSP/J2EE. Είναι αυτή που παρουσιάζει στο χρήστη τις σελίδες για τη δημιουργία της προσωποποιημένης σελίδας, καθώς

και αυτή που κατασκευάζει την προσωποποιημένη σελίδα από τα Web Components (Δημιουργία και σύνθεση σελίδας). Πρέπει να έχει πρόσβαση στη Βάση Δεδομένων από την οποία λαμβάνει στοιχεία (όπως το ποια web components έχει επιλέξει ένας συγκεκριμένος χρήστης).

3.4.3 ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

Δεν υπάρχει ξεχωριστή βάση δεδομένων για τη λειτουργία της τεχνικής των Web Components, αλλά αποτελεί μέρος της κεντρικής βάσης δεδομένων της εφαρμογής στο διαδίκτυο. Στη βάση δεδομένων αποθηκεύονται όλες οι πληροφορίες που αφορούν στους χρήστες και στις επιλογές τους, όπως το ποια Web Components έχουν επιλέξει. Ο HTML κώδικας των Web Components όμως, όπως και το καθαρό κείμενο, το οποίο αυτά περιέχουν ,αποθηκεύονται στο File System του Web Server της διαδικτυακής εφαρμογής.

4.1 HTML PARSER

4.1.1 ΕΙΣΑΓΩΓΗ-ΚΛΑΣΕΙΣ

Σε αυτή την ενότητα θα αναλύσουμε τη δομή ενός απλού HTML parser που υλοποιήθηκε για χρήση στο πρόγραμμα 'Web Components Analyzer'. Χρησιμοποιείται για τη δημιουργία του HTML tree μιας σελίδας. Δύο λόγοι μας οδήγησαν στη δημιουργία του. Αφ' ενός είναι πιο εύκολο να προσαρμοστεί στις ανάγκες του προγράμματος, όπως είναι η διόρθωση προβλημάτων στη σύνταξη του HTML κώδικα γνωστών site που χρησιμοποιούμε και αφ' ετέρου να μπορεί να εξελιχθεί μελλοντικά. Θα μπορούσε να τροποποιηθεί ο parser έτσι ώστε εκτός από το HTML tree να μας εξάγει και το αντίστοιχο XHTML κώδικα. Ο HTML parser αποτελεί ξεχωριστό πακέτο (packet) που χρησιμοποιείται από το υπόλοιπο πρόγραμμα. Το πακέτο αυτό ονομάζεται WC.HTMLParser και αποτελείται από τις ακόλουθες κλάσεις:

1. HTML Parser. Αποτελεί την κύρια κλάση του πακέτου. Παίρνει τον HTML κώδικα σαν ένα string και αναλαμβάνει να δημιουργήσει μία δενδρική δομή από στιγμιότυπα της τάξης "Element", η οποία αναπαριστά τη δενδρική δομή του HTML κώδικα. Δημιουργείται ένα στιγμιότυπο (instance) της κλάσης HTMLParser για κάθε ανάλυση που γίνεται.
2. Element. Τα στιγμιότυπα αυτής της κλάσης αναπαριστούν τους κόμβους της δενδρικής δομής που παράγεται από την ανάλυση του HTML κώδικα και αποτελούν προιόν της ανάλυσης που γίνεται από τον parser. Επίσης τα στιγμιότυπα της Element κλάσης χρησιμοποιούνται για να αναπαραστήσουν τους κόμβους του index tree. Κάθε αντικείμενο τύπου Element (στιγμιότυπο της κλάσης) περιλαμβάνει όλες τις απαραίτητες πληροφορίες που χρειάζονται για να αναπαραστήσει πλήρως ένα tag του HTML κώδικα (ή του index tree) ή μια περιοχή κειμένου. Επίσης περιέχει όλες τις πληροφορίες που χρειάζονται για να διασυνδεθεί με τα υπόλοιπα Element που αποτελούν το δέντρο που αναπαριστά τον HTML κώδικα ή το index tree. Επίσης, περιέχει όλες τις πληροφορίες που χρειάζονται για να διασυνδεθεί με τα υπόλοιπα Element που απαρτίζουν το δέντρο που αναπαρίσταται τον HTML κώδικα ή το index tree. Στην περίπτωση που με ένα στιγμιότυπο της κλάσης Element αναπαριστά ένας κόμβος του HTML tree, αυτό μπορεί να έχει έναν από τους παρακάτω τύπους:
    - **OPENING_TAG:** Το στιγμιότυπο αναπαριστά ένα opening tag που «ανοίγει» την περιοχή κειμένου που χαρακτηρίζει. Για παράδειγμα ένα opening tag είναι το «<TABLE>»
    - **CLOSING_TAG:** Το στιγμιότυπο αναπαριστά ένα closing tag που «κλείνει» μια μαρκαρισμένη περιοχή. Πέρα από αυτό το tag το κείμενο δε χαρακτηρίζεται από τις ιδιότητες που του προσδίδει. Για παράδειγμα το closing tag "</TABLE>", «κλείνει» έναν πίνακα, ενώ το tag «</B>» κλείνει μια περιοχή όπου το κείμενο εμφανίζεται Bold Face.
    - **OPENING_CLOSING_TAG:** Το στιγμιότυπο αναπαριστά ένα tag, το οποίο δε χρειάζεται να έχει closing tag (επειδή δε μαρκάρει κάποια περιοχή κειμένου). Χαρακτηριστικό παράδειγμα για αυτή την περίπτωση αποτελεί το tag "<IMG>", το οποίο χρησιμοποιείται για την εισαγωγή μιας εικόνας στη Web σελίδα. Βέβαια κάποια tag αυτού του τύπου μπορεί να έχουν και closing tag, όπως είναι για παράδειγμα

το tag “<P>”, το οποίο αναπαριστά την έναρξη καινούριας παραγράφου και μπορεί να συνοδεύεται από το αντίστοιχο closing tag ‘</P>’ που οριοθετεί το τέλος της παραγράφου.

- **TEXT:** Στην περίπτωση που κάποιο στιγμιότυπο της κλάσης Element είναι τύπου Text, τότε αυτό αναπαριστά μια περιοχή κειμένου του HTML αρχείου και όχι κάποιο tag. Τα elements τύπου text, μαζί με τα elements τύπου OPENING_CLOSING_TAG αποτελούν τα φύλλα του HTML δέντρου.
- **COMMENT:** Αναπαριστά μία περιοχή με σχόλια
- **SPECIAL:** Τα elements τύπου SPECIAL αναπαριστούν tag που αρχίζουν με “<!” ή “<?”. Εξαίρεση αποτελούν τα σχόλια που αναπαρίστανται με Element τύπου Comment. Τα SPECIAL elements έχουν ειδικές λειτουργίες και δε χρησιμοποιούνται για markup, αλλά για μεταδεδομένα.
- **SCRIOT:** Με ένα Element τύπου SCRIPT αναπαριστούνται περιοχές κειμένου από τον HTML κώδικα που αποτελούν scripting κώδικα και περικλείονται από τα tag “<SCRIPT>” και “</SCRIPT>”. Αυτός ο κώδικας μπορεί να είναι JavaScript, Jscript ή VBScript.

Οι μεταβλητές στιγμιότυπου (instance variables) της κλάσης Element περιλαμβάνουν εκτός των άλλων τις παρακάτω μεταβλητές μέλη:

- **(String) name.** Το όνομα του Element. Αν πρόκειται για tag, το όνομα είναι το είδος του (π.χ. TABLE), ενώ αν πρόκειται για περιοχή κειμένου το όνομά του είναι το κείμενο.
- **(String) id.** Αναπαριστά το id ενός κόμβου στο index tree. Η χρήση του γίνεται σε αρκετά σημεία του κώδικα για να προσδιορίσει έναν κόμβο του index tree.
- **(Hashtable) attributes.** Στο hashtable αυτό περιλαμβάνονται όλα τα χαρακτηριστικά (attributes) που μπορεί να έχει ένα tag, σε μορφή ζεύγους (key, value). Για παράδειγμα το tag TABLE μπορεί να περιλαμβάνει στο hashtable attributes, τα ζεύγη (‘width’, ‘400’) και (‘height’, ‘250’). Τόσο το κλειδί (key), όσο και η τιμή (value) σε ένα ζεύγος είναι αντικείμενα τύπου String.
- **(Element) parent.** Στη μεταβλητή αυτή αποθηκεύεται ο πατέρας του συγκεκριμένου κόμβου που αναπαρίσταται με το στιγμιότυπο της κλάσης Element. Με τη χρήση αυτής της μεταβλητής (σε συνδυασμό με το Vector children) μπορεί να δημιουργηθεί η δενδρική δομή του HTML tree ή του index tree και όλες οι απαραίτητες μέθοδοι για τους υπολογισμούς που γίνονται.
- **(Vector) children.** Στο διάνυσμα αυτό αποθηκεύονται όλα τα αντικείμενα τύπου Element, που αποτελούν τα παιδιά ενός tag στο HTML tree ή στο index tree. Είναι null για ένα στιγμιότυπο της Element που αναπαριστά μία περιοχή κειμένου ή ένα tag που δεν έχει αντίστοιχο closing tag (π.χ. ‘IMG’).
- **(Element) original.** Στην περίπτωση που ένα στιγμιότυπο της κλάσης Element αναπαριστά ένα κόμβο του index tree, αυτή η μεταβλητή κρατά μία αναφορά στον αντίστοιχο κόμβο στο HTML tree. Χρησιμοποιείται στα σημεία του κώδικα όπου έχουν γίνει οι απαραίτητοι υπολογισμοί στο index tree και πρέπει η ροή του προγράμματος να περάσει στο HTML tree για να έχει πρόσβαση στον

κώδικα της σελίδας. Αν το στιγμιότυπο αναπαριστά κόμβο του HTML tree, τότε η μεταβλητή αυτή είναι null.

Η κλάση element συνολικά περιλαμβάνει 97 μεθόδους. Υπάρχουν μέθοδοι πρόσβασης σε μεταβλητές, μέθοδοι για υπολογισμούς που γίνονται κατά την ανάλυση του HTML κειμένου, καθώς και μέθοδοι για τον αλγόριθμο τμηματοποίησης ή τις διαδικασίες εκπαίδευσης και ανανέωσης. Τα δύο τελευταία είδη μεθόδων έχουν συμπεριληφθεί στον κώδικα του parser, επειδή αυτό διευκόλυνε τη συγγραφή του κώδικα του προγράμματος.

3. CharBuffer. Η κλάση CharBuffer είναι βοηθητική για τον Parser. Χρησιμεύει σαν buffer χαρακτήρων. Κατά την ανάλυση που γίνεται ο parser (στιγμιότυπο της κλάσης HTMLParser) λειτουργεί σαν αυτόματο που μεταβαίνει σε διάφορες καταστάσεις, ανάλογα με το κείμενο (HTML κώδικα) που διαβάζει. Ανάλογα με την κατάσταση που βρίσκεται ο parser, υπάρχουν διάφορες συμβολοσειρές οι οποίες προκαλούν μετάβαση σε διαφορετική κατάσταση ('Escape Strings'). Το στιγμιότυπο της κλάσης CharBuffer που χρησιμοποιεί ο parser επιστρέφει ένα token που είναι η περιοχή κειμένου μέχρι την πρώτη συμβολοσειρά αλλαγής κατάστασης. Ο parser παίρνει το κείμενο αυτό και τη συμβολοσειρά αλλαγής κατάστασης, ώστε να μπορεί να υπολογίσει σε ποια κατάσταση θα μεταβεί. Στη συνέχεια δημιουργεί ένα στιγμιότυπο της κλάσης Element (το τι είδους Element θα δημιουργηθεί εξαρτάται από την κατάσταση του parser και από το escape string που επέστρεψε ο buffer). Το element αυτό τοποθετείται στο HTML tree που δημιουργείται.

4. ExpandedTreeCellRenderer. Η κλάση αυτή είναι υποκλάση της 'DefaultTreeCellRenderer', η οποία είναι μια κλάση του Swing (API της java για δημιουργία γραφικών κα περιβάλλοντος διεπαφής). Χρησιμοποιείται για να απεικονίζει το HTML tree στο GUI του προγράμματος διαχείρισης «Web Components Analyzer». Αυτό ήταν ιδιαίτερα χρήσιμο κατά την ανάπτυξη του αλγορίθμου τμηματοποίησης και της διαδικασίας εκπαίδευσης / ανανέωσης. Η κλάση ExpandedTreeCellRenderer δεν έχει καμία άλλη χρησιμότητα πέραν της απεικόνισης του HTML tree (δε χρησιμοποιείται στους αλγόριθμους του συστήματος) και αυτός είναι ο μόνος σκοπός για τον οποίο υλοποιήθηκε.

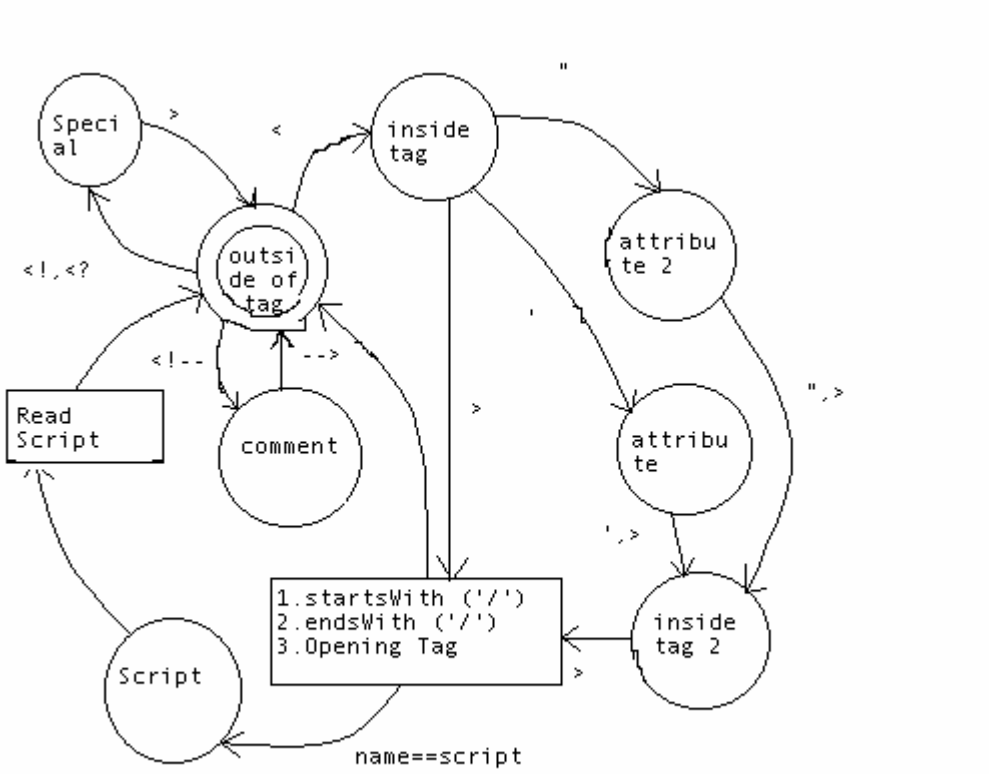
4.1.2 ΛΕΙΤΟΥΡΓΙΑ

Στο βήμα 1 του αλγόριθμου τμηματοποίησης (fragmentation algorithm), η σελίδα του Web που αναλύεται μεταφέρεται από το πρόγραμμα 'Web Components Analyzer' στον υπολογιστή που εκτελεί την ανάλυση και χρησιμεύει ως πηγή δεδομένων για τον Web Server της υπηρεσίας. Για να γίνει αυτό χρησιμοποιούμε την τάξη URL του API της java. Στη συνέχεια , στο βήμα 2 του αλγόριθμου τμηματοποίησης η κλάση HTMLParser που αναφέρθηκε στην προηγούμενη ενότητα αναλαμβάνει να αναλύσει το κείμενο του HTML κώδικα και να το αναπαραστήσει σε δενδρική δομή (HTML tree). Στην παρούσα ενότητα αναλύεται ο τρόπος λειτουργίας του parser, δηλαδή ενός αντικειμένου της κλάσης HTMLParser.

Η κλάση HTMLParser έχει κατασκευαστεί ώστε ένα στιγμιότυπο της να χρησιμοποιείται για μία μόνο ανάλυση. Τα αποτελέσματα της ανάλυσης (για

παράδειγμα το HTML tree ή κάποια στατιστικά) μπορούν να ληφθούν από αυτό το στιγμιότυπο, το οποίο στη συνέχεια καταστρέφεται, αφού πλέον δε χρειάζεται.

Η κλάση HTMLParser λειτουργεί ως πεπερασμένο αυτόματο, το οποίο μεταβαίνει σε διάφορες καταστάσεις, καθώς διαπερνά και αναλύει τον HTML κώδικα (το οποίο λαμβάνει ως string) και δημιουργεί τη δενδρική δομή του HTML tree, η οποία αποτελείται από διασυνδεμένα στιγμιότυπα της κλάσης Element. Μεγάλο ρόλο στη λειτουργία της HTMLParser διατελεί η κλάση CharBuffer, η λειτουργία της οποίας απλοποιεί πολύ τον κώδικα της HTMLParser, η οποία περιέχει μόνο τη 'λογική' του parser. Ένα στιγμιότυπο της κλάσης CharBuffer αποθηκεύει το κείμενο του HTML κώδικα και το σπάει σε 'tokens', δηλαδή καθαρό κείμενο, κείμενο που βρίσκεται μέσα σε tag, κείμενο που βρίσκεται μέσα σε attribute κάποιου tag, κτλ. Σε κάθε βήμα, ανάλογα με την κατάσταση που βρίσκεται ο parser, η CharBuffer γνωρίζει ποια είναι τα Escape Strings, δηλαδή ποιες είναι οι ακολουθίες χαρακτήρων που προκαλούν αλλαγή κατάστασης του parser. Όταν διαβάσει από το κείμενο μια τέτοια ακολουθία χαρακτήρων επιστρέφει στον parser το Escape String, καθώς και το token (κείμενο) που διάβασε (μέχρι να συναντήσει το Escape String). Στη συνέχεια, ο parser αποφασίζει τι ενέργειες θα κάνει με βάση την κατάσταση που βρίσκεται και το τι Escape String επέστρεψε η CharBuffer. Η σχηματική λειτουργία φαίνεται παρακάτω.



Υπάρχουν 9 διαφορετικές καταστάσεις στις οποίες μπορεί να βρίσκεται το αυτόματο που μοντελοποιεί την κλάση HTMLParser. Ανάλογα με την κατάσταση στην οποία βρίσκεται θα εκτελέσει διαφορετικές ενέργειες κάθε φορά και σε συνάρτηση με την επόμενη συμβολοσειρά αλλαγής κατάστασης ('Escape String') μεταβαίνει στην επόμενη κατάσταση. Κατά τη διάρκεια αυτής της διαδικασίας, ο parser δημιουργεί ένα HTML tree ξεκινώντας από ένα element (root element) και προσθέτει σαν

απογόνους του όλα τα άλλα Elements που αναπαριστούν τους κόμβους του HTML tree. Το πρώτο παιδί του root element είναι πάντα ένα element τύπου opening tag, το οποίο αναπαριστά το tag 'HTML' και είναι πρόγονος όλων των υπόλοιπων κόμβων του δέντρου. Τα παιδιά του κόμβου 'HTML' είναι το 'HEAD' και το 'BODY'. Η λειτουργία του HTMLParser ολοκληρώνεται όταν τελειώσει όλη την ανάλυση του HTML κώδικα.

Η πρώτη κατάσταση στην οποία βρίσκεται ο Parser, ξεκινώντας τη λειτουργία του, είναι η 'Outside Of Tag'. Όταν το αυτόματο (parser) βρίσκεται σε αυτή την κατάσταση, βρίσκεται στο κείμενο του HTML κώδικα έξω από κάποιο tag. Αυτό σημαίνει ότι διαβάζει καθαρό κείμενο. Για να αλλάξει η κατάσταση πρέπει να διαβάσει το χαρακτήρα '<', ο οποίος σηματοδοτεί την έναρξη κάποιου tag. Ανάλογα με τους χαρακτήρες τώρα που ακολουθούν το σύμβολο '<' το αυτόματο μεταβαίνει στην ανάλογη κατάσταση, όπως φαίνεται στο πιο κάτω σχήμα..

Σε όλες τις περιπτώσεις το κείμενο πριν το χαρακτήρα '<' τοποθετείται ως περιοχή κειμένου στο HTML tree. Αυτό σημαίνει ότι δημιουργείται ένα νέο αντικείμενο Element (τύπου TEXT), το οποίο τοποθετείται σαν παιδί του τρέχοντος κόμβου (δηλαδή του τελευταίου κόμβου που έχει δημιουργήσει ο parser).

Οι μεταβάσεις 2 και 3 προκαλούν μετάβαση του parser σε περιοχές σχολίων (comment) ή ειδικές παροχές (special). Στις περιοχές σχολίων βρίσκεται κείμενο που έχει εισάγει ο συγγραφέας μιας σελίδας και δε λαμβάνεται υπ' όψη από τους browsers, ενώ οι ειδικές περιοχές χρησιμοποιούνται όχι για μαρκάρισμα κάποιας περιοχής κειμένου, αλλά για παροχή πληροφοριών, όπως π.χ. metadata σχετικά με τη σελίδα. Μια περιοχή σχολίου τερματίζεται με το escape string '-->', ενώ μια ειδική περιοχή με το '>'.

Όταν ο parser βρίσκεται έξω από κάποιο tag και συναντήσει το escape string '<', μεταβαίνει στην κατάσταση 'Inside Tag', κατά την οποία πρέπει να δημιουργήσει έναν κόμβο του HTML tree που αναπαριστά κάποιο tag. Αυτός ο κόμβος μπορεί να είναι είτε κάποιο Opening Tag, δηλαδή tag που ξεκινάει μια περιοχή (π.χ. <BOLD>), είτε κάποιο Closing tag (π.χ.</BOLD>), το οποίο τερματίζει μια περιοχή μέσα στο κείμενο, είτε κάποιο Opening Closing Tag, δηλαδή ένα tag που δεν ορίζει κάποια περιοχή στο κείμενο και δεν χρειάζεται να έχει Closing Tag (π.χ. '<IMG src='logo.gif' />'). Επίσης ο Parser μπορεί να δημιουργήσει και ένα τέταρτο είδος κόμβου, το 'SCRIPT', το οποίο αναπαριστά μια περιοχή που αποτελεί client-side scripting κώδικα (π.χ. Javascript, VBScript, JScript). Το είδος του κόμβου εξαρτάται από το πώς τελειώνει και το πώς αρχίζει το περιεχόμενο που περικλείεται από τους χαρακτήρες '<' και '>'. Επίσης, ανάλογα με το είδος του tag μπορεί να υπάρχουν επιπλέον χαρακτηριστικά ('attributes'). Όλα αυτά καθορίζουν τις επόμενες μεταβάσεις του Parser οι οποίες φαίνονται στον παρακάτω πίνακα.

A/A	Κατάσταση	Escape String	Ενέργεια	Επόμενη Κατάσταση
1.	Outside Tag	<	Τοποθέτηση του token σαν κείμενο (Element τύπου TEXT) στο HTML tree (σαν παιδί του τρέχοντος Element)	Inside Tag
2.	Outside Tag	<!--	Τοποθέτηση του token σαν κείμενο στο HTML tree	Comment

3.	Outside Tag	<?,<!	Τοποθέτηση του token σαν κείμενο στο HTML tree	Special
4.	Comment	-- >	Τοποθέτηση του token σαν σχόλιο (Element τύπου Comment) στο HTML tree (σαν παιδί του Τρέχοντος Element)	Outside Tag
5.	Special	>	Τοποθέτηση του token σαν ειδική περιοχή (Element τύπου Special) στο HTML tree (σαν Παιδί του τρέχοντος Element)	Outside Tag
6.	Inside Tag	‘	Δημιουργία νέου attribute	Attribute
7.	Inside Tag	“	Δημιουργία νέου attribute	Attribute 2
8.	Attribute	‘,>	Έξοδος από attribute	Inside Tag 2
9.	Attribute 2	“,>	Έξοδος από attribute	Inside Tag 2
10.	Inside Tag	>	Έξοδος από Tag	Outside Tag
11.	Inside Tag 2	>	Έξοδος από Tag	Outside Tag
12.	Inside Tag	>, name==string	Έξοδος από Tag	Script
13.	Inside Tag 2	>, name==string	Έξοδος από Tag	Script
14.	Script	</script>	Έξοδος από Tag	Outside Tag

5 ΜΕΛΛΟΝΤΙΚΗ ΒΕΛΤΙΩΣΗ

Η συγκεκριμένη διπλωματική εργασία δεν είναι από μόνη της ικανή για την κατασκευή της προσωποποιημένης σελίδας, καθώς δεν έχει ξεκαθαριστεί πως θα επιτευχθεί η επικοινωνία της μηχανής με τη βάση δεδομένων. Κατά τη διάρκεια της διπλωματικής εργασίας έγιναν κάποιες προσπάθειες επικοινωνίας της μηχανής με τη Data Base μεσω του mySQL. Ένας SQL server θα πρέπει να τρέχει όπου τρέχει το πρόγραμμα και να συγκρατεί τις προτιμήσεις και τις σελίδες που απασχολούν τον κάθε χρήστη καθώς επίσης και τα username και passwords των χρηστών.

Επίσης ο αλγόριθμος της μηχανής πιθανόν να μπορεί να δεχτεί κάποια βελτίωση. Κάποια sites τα οποία ανανεώνονται για παράδειγμα ανά κάποια περίοδο. Για τα συγκεκριμένα sites για τα οποία θα δηλώνεται από το χρήστη η περίοδος θα μπορεί η μηχανή να κρατάει στη μνήμη της τα συγκεκριμένα Components και να μην ανατρέχει κάθε φορά στη συγκεκριμένη σελίδα. Για παράδειγμα αν υπάρχει Component για τη βαθμολογία στο πρωτάθλημα της Formula 1 δεν είναι απαραίτητο η μηχανή να ανατρέχει στο site της FIA κάθε φορά που ο χρήστης ανοίγει την προσωποποιημένη σελίδα αλλά μπορεί να κρατάει τη βαθμολογία στη μνήμη και να παίρνει δεδομένα από το site μόνο κάθε Κυριακή όπου πιθανό να έχουμε αγώνα οπότε θα υπάρξει αλλαγή στη βαθμολογία του πρωταθλήματος και συνεπώς αλλαγή στα δεδομένα του συγκεκριμένου Component.

6 ΣΥΜΠΕΡΑΣΜΑΤΑ

Γενικά για τη συγκεκριμένη μηχανή μπορούμε να πούμε ότι ο χρήστης κερδίζει πολύ χρόνο και δεν χρειάζεται να πληκτρολογεί κάθε φορά πολλές διευθύνσεις για την καθημερινή του ενημέρωση. Είναι λοιπόν μια μηχανή χρήσιμη για τις καθημερινές του πληροφορίες από sites με συγκεκριμένη δομή και standard Components όπου έχουν συγκεκριμένη πληροφορία. Δεν ενδείκνυται πάντως η χρήση του προγράμματος αυτού για sites μεταβαλλόμενα. Δεν θα ήταν σωστό δηλαδή να βάλουμε ένα Component για την κίνηση των μετοχών μιας εταιρίας στο Χρηματιστήριο και να θέσουμε σαν "πηγή" σελίδα τη σελίδα της εταιρείας διότι αυτή υπάρχει περίπτωση να μην ανανεώνεται έγκαιρα οπότε καλό θα ήταν να παίρνουμε τα δεδομένα από τη σελίδα του Χρηματιστηρίου Αξιών Αθηνών ή από το Χρηματιστήριο στο οποίο διακινούνται οι μετοχές της συγκεκριμένης εταιρείας.

Επίσης δεν είναι ορθό να χρησιμοποιούμε το συγκεκριμένο πρόγραμμα για Components τα οποία υπάρχουν σε ένα site απλά τη συγκεκριμένη χρονική στιγμή αλλά στο μέλλον το πιο πιθανό είναι να μην υπάρχουν. Γενικά απαιτείται προσοχή στα path τα οποία θα δίνονται στη μηχανή για να βρίσκει τα διάφορα Components.

7 ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] C. Bouras and A. Konidaris, “Web Components: A Concept for Improving Personalization and Reducing User Perceived Latency on the World Wide Web”, Proceedings of the 2nd International Conference on Internet Computing (IC2001), Las Vegas, Nevada, USA, June 25th - 28th 2001, Vol 2, pp.238-244.

[2] C. Bouras and A. Konidaris, “Performance Evaluation of a Hybrid Run-time Management Policy for Data Intensive Web Sites”, World Wide Web Journal: Internet and Web Information Systems, Kluwer Academic Publishers, Vol 6, Issue 1, March 2003, pp. 23-47

[3] C. Bouras and A. Konidaris, “Estimating and Eliminating Redundant Data Transfers Over the Web: A Fragment Based Approach”, Proceedings of the 3rd International Conference on Internet Computing (IC 2002), USA, 2002.

[4] C. Bouras, V. Kapoulas and I. Misedakis, “A Web-page Fragmentation Technique for Personalized Browsing” (Poster Absract), To appear in ACM SAC 2004 (IDM track), March 14-17, 2004

[5] I. Misedakis, V. Kapoulas, C. Bouras, "Web page fragmentation and content manipulation for constructing personalized portals", The Sixth Asia Pasific Web Conference (APWeb ’04), Hangshou, China, April 14 - 17 2004, (to appear)

[6] C. Bouras, V. Kapoulas, I. Misedakis, "Web Page Fragmentation for Personalized Portal Construction", IEEE International Conference on Information Technology: Coding and Computing - ITCC 2004 (Web/IR Track), The Orleans, Las Vegas, Nevada, USA, April 5 - 7 2004, (to appear)

[7] E. Hwang and Sieun Lee, “Web Surfing Assistant for Improving Web Accessibility”, International Conference on Internet Computing (IC'03), Las Vegas, Nevada, USA, June 23-26, 2003.

[8] J. Challenger, A. Iyengar, K. Witting, C. Ferstat, and P. Reed. “A publishing system for efficiently creating dynamic web content”, Proceedings of the IEEE Conference on Computer Communications (INFOCOM'00), March 2000.

[9] Craig E. Wills and Mikhail Mikhailov, “Studying the impact of more complete server information on Web caching”, 5th International Web caching and Content delivery Workshop, Lisbon, Portugal, 22-24 May 2000

[10] Masahiro Hori, Goh Kondoh, Kohichi Ono, Shin-ichi Hirose, and Sandeep Singhal, “Annotation-based Web Content Transcoding”. In Proceedings of the 9th International World Wide Web Conference, Amsterdam, Netherlands , May 2000.

[11] Buyukkokten, H. Garcia-Molina, A, Paepcke, 'Accordion Summarization for End-Game Browsing on PDAs and Cellular Phones', In Proceedings of the Conference on Human Factors in Computing Systems, CHI'01, 2001.

[12] Juliana Freire, Bharat Kumar, Daniel Lieuwen, 'WebViews: Accessing Personalized Web Content and Services', Proceedings of the 10th international conference on World Wide Web, Hong Kong, 2001

[13] E. Kaasinen, M. Aantonen, J. Kolari, S. Melakoski and T. Laakko, 'Two Approaches to Bringing Internet Services to WAP devices', Proceedings of the 9th International World Wide Web Conference, 2000

[14] K.H.Britton et al., "Transcoding: Extending E-Business to New Environments", IBM Systems Journal, vol. 40, no 1, 2001

[15] T. Bickmore and W. Schilit, "Digestor: Device-Independent Access to the World Wide Web", Computer Networks and ISDN Systems, vol. 29, no 8, 1997, p. 1075-1082

[16] J. Chen, B. Zhou, J. Shi, H. Zhang, Q. Fengwu, "Function-Based Object Model Towards Web-site Adaptation", Proceedings of the 10th WWW Conference, ACM Press, 2001, p. 587-596

[17] Y. Hwang, C. Jung, J. Kim, S. Chung, "WebAlchemist: A Web Transcoding System for Mobile Web Access in Handheld Devices", Proceedings of ITCom, SPIE – The International Society for Optical Engineering, 2001, pages 37-46

[18] Y. Hwang, J. Kim, E. Seo, "Structure-Aware Web Transcoding for Mobile Devices", IEEE Internet Computing Journal, September-October 2003

[19] D. Buttler, L. Liu, C. Pu, "A Fully Automated Object Extraction system for the World Wide Web", Proceedings of the 2001 International Conference on Distributed Computing Systems (IDCS '01).

[20] J. Caverlee, D. Butler, L. Liu, "Discovering Objects in Dynamically-Generated Web Pages"

[21] S. Yu, D. Cai, J. Wen, W. Ma, "Improving Pseudo-Relevance Feedback in Web Information Retrieval Using Web Page Segmentation", Proceedings of WWW '03, May 20-24, 2003, Budapest, Hungary

[22] Y. Chen, W. Ma, H. Zhang, "Detecting Web Page Structure for Adaptive Viewing on small Form Factor Devices", Proceedings of WWW '03, May 20-24, 2003, Budapest, Hungary

[23] B. Adelberg. “Nodose – a tool for semi-automatically extracting structured and semi-structured data from text documents”. ACM SIGMOD, 1998.

[24] N. Ashish and C.A Knoblock. “Semi-automatic wrapper generation for internet information sources”. In Proceedings of Coopis Conference, 1997

[25] D. W. Embley, Y. Jiang and Y.-K. Ng. “Record-boundary discovery in web-documents”. In Proceedings of the 1999 ACM SIGMOD, Philadelphia, Pennsylvania, USA, June 1999.

[26] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, A. Crespo. “Extracting semi-structured data from the web. Proceedings of Workshop on Management of Semi-structured data”, pages 18-25, 1997.

[27] N. Kushmerick, D. Weil, R. Doorenbos. “Wrapper induction for information extraction”. In Proceedings of International Joint Conference on Artificial Intelligence (IJCAI), 1997

[28] L. Liu, C. Pu, W. Han. “XWrap: An XML-enabled Wrapper Construction System for Web Information Sources”. Proceedings of the International Conference on Data Engineering, 2000.

[29] A. Sahuguet, F. Azavant. “WysiWyg Web Wrapper Factory (W4F)”. Proceedings of WWW Conference, 1999.

[30] S. Soderland. “Learning to extract text-based information from the world-wide web”. Proceedings of Knowledge Discovery and Data Mining”, 1997

[31] Y. D. Yang, J. L. Chen and H.J. Zhang, “Adaptive Delivery of HTML contents”, WWW9 Poster Proceedings, May 2000, pages 24-25

[32] N. Ashish, C. Knoblock, “Wrapper generation for semi-structured information sources”, Proceedings PODS/SIGMOD 97, May 1997

[33] D. Simth, M. Lopez, “Information extraction for semi-structured documents”, Proceedings PODS/SIGMOD 97, May 1997

[34] S. Nestorov, S. Abiteboul, R. Motwani, “Inferring Structure in Semistructured Data”, Proceedings PODS/SIGMOD 97, May 1997

[35] D. W. Embley, Y.K. Ng, L. Xu, “Filtering Multiple-record Web Documents based on Application Ontologies”

[36] S.J. Lim, Y.K. Ng, “An Automated Approach for Retrieving Hierarchical Data from HTML Table”, Proceedings of CIKM ’99, Kansas, 1999, p. 466-474

[37] W3C, “HTML 4.0 Specification”

[38] W3C, “Cascading Style sheets”

[39] W3C, “HTML Tidy”

[40] A. F. R. Rahman, H. Alam, R. Hartono, “Content Extraction from HTML Documents”, 1st International Workshop on Web Document Analysis (WDA2001), 2001.

[41] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, “Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices”. In Proceedings of 10th International WWW Conference, 2001

[42] N. Wacholder, D. Evans, J. Klavans. “Automatic Identification and Organization of Index Terms for Interactive Browsing”. In Joint Conference on Digital Libraries ’01, 2001

[43] Manuela Kunze, Dietmar Rosner. “An XML-based Approach for the Presentation and Exploitation of Extracted Information”. In 19th International Conference on Computational Linguistics, Coling, 2002

[44] A. F. R. Rahman, H. Alam, R. Hartono, “Understanding the Flow of Content in Summarizing HTML Documents”, In International Workshop on Document Layout Interpretation and its Applications, DLIA01, September 2001

[45] A. P. Asirvatham, K.K. Ravi, “Web page Classification based on Document Structure”,

[46] V. Anupam, J. Freire, B. Kumar, D. Lieuwen, “Automatic Web Navigation with the WebVCR”, Proceedings of the 9th International WWW, p. 503-517, 2000

[47] H. Davulcu, J. Freire, M. Kifer, I. Ramakrishnan, “A layered architecture for quering dynamic web content”, In Proceedings of SIGMOD99, p. 491-502

[48] H. Davulcu, G. Yang, M. Kifer, I. Ramakrishnan, “Computational Aspect of Resilient Data Extraction from Semistructured Sources”, In Proceedings of PODS, 2000

[49] T. Phelps, R. Wilensky, “Robust intra-document locations”, Proceedings of the 9th International WWW Conference, 2000

[50] A. Sahuguet, F. Azavant. “Building light-weight wrappers for legacy web data-sources using W4F”, In Proceedings of VLDB, p. 738-741, 1999

[51] J. Freire, “Using Wrappers for Device Independent Web Accaess: Opportunities, Challenges and Limitations”,

[52] T. Kistlera, H. Marais, “WebL: A programming language for the Web”. In Proceedings of the 7th WWW Conference, p. 259-270, 1998

[53] A. Sahuguet, F. Azavant, “Web Ecology: Recycling HTML pages as XML documents using W4F”,

[54] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J.L. Wiener, “The Lorel Query Language for Semistructured Data”, Journal on Digital Libraries, 1997

[55] J.R. Gruser, L. Raschid, M.E. Vidal, L. Bright. “Wrapper Generation for Web Accessible Data Sources”, COOPIS, 1998

[56] G. Huck, P. Fankhauser, K. Aberer, E. J. Neuhold. “JEDI: Extracting and Synthesizing Information from the Web”. In COOPIS, New York, 1998.

[57] O. Buyukkokten, H. Garcia-Molina, A. Paepcke, T. Winograd, “Power Browser: Efficient Web Browsing for PDAs”. Proceedings of CHI’ 2000, p. 430-437

[58] M. Jones, G. Marsden, N. Mohd-Nasir, K. Boone, G. Buchanan, “Improving Web Interaction on Small Displays”, Proceedings of 8th WWW Conference, 1999, p. 51-59

[59] T. Bickmore, A. Girgensohn, J.W. Sullivan, “Web Page Filtering and Reauthoring for Mobile Users”, Computer Journal, vol. 42, no.6, 1999, p. 534-546

[60] S. Nestorov, S. Abiteboul, R. Motwani, “Inferring Structure in Semistructured Data”, In Proceedings of PODS/SIGMOD 97, May 1997, p.117-121

[61] K. Nagao, Y. Shirai, K. Squire, “Semantic annotation and transcoding”, IEEE Multimedia, vol. 8(2), p. 69-81, 2001

[62] Y.D. Yang, H.J. Zhang, “HTML Page Analysis Based on Visual Cues”, in Proceedings of the 6th International Conference on Document Analysis and Recognition, Seattle, USA, 2001

[63] S. Chakrabarti, M. Joshi, V. Tawde, “Enhanced Topic Distillation using Text, Markup Tags and Hyperlinks”

[64] J. Misedakis “Research and Development of Technics For Improving The User Surfing On The Net ” Patra, 2004