NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

PROGRAM OF POSTGRADUATE STUDIES

PhD THESIS

Advance BPEL execution adaptation using QoS parameters and collaborative filtering techniques

Dionisios D. Margaris

ATHENS

DECEMBER 2014

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Προηγμένες τεχνικές προσαρμογής εκτέλεσης σεναρίων BPEL με χρήση παραμέτρων ποιοτικών χαρακτηριστικών και τεχνικών συνεργατικού φιλτραρίσματος

Διονύσιος Δ. Μάργαρης

ΑΘΗΝΑ

ΔΕΚΕΜΒΡΙΟΣ 2014

PhD THESIS

Advanced BPEL execution adaptation using QoS parameters and collaborative filtering techniques

Dionisios D. Margaris

SUPERVISOR: Panayiotis Georgiadis, Professor Emeritus NKUA

THREE-MEMBER ADVISING COMMITTEE:

Panayiotis Georgiadis, Professor Emeritus NKUA

Stathes Hadjiefthymiades, Associate Professor NKUA

Costas Vassilakis, Associate Professor University of Peloponnese

SEVEN-MEMBER EXAMINATION COMMITTEE

(Signature) (Signature)

Panayiotis Georgiadis,
Professor Emeritus NKUA

Stathes Hadjiefthymiades,
Associate Professor NKUA

(Signature) (Signature)

Costas Vassilakis,
Associate Professor University of Peloponnese

Stefanos Gritzalis,
Professor University of the Aegean

(Signature) (Signature)

Georgios Doukidis,
Professor AUEB

Panagiotis Stamatopoulos,
Assistant Professor NKUA

(Signature)

Nikolaos Tselikas,
Assistant Professor University of Peloponnese

Examination Date __/__/2014

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

Προηγμένες τεχνικές προσαρμογής εκτέλεσης σεναρίων BPEL με χρήση παραμέτρων ποιοτικών χαρακτηριστικών και τεχνικών συνεργατικού φιλτραρίσματος

Διονύσιος Δ. Μάργαρης

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: Παναγιώτης Γεωργιάδης, Ομότιμος Καθηγητής ΕΚΠΑ

ΤΡΙΜΕΛΗΣ ΕΠΙΤΡΟΠΗ ΠΑΡΑΚΟΛΟΥΘΗΣΗΣ:

Παναγιώτης Γεωργιάδης, Ομότιμος Καθηγητής ΕΚΠΑ

Ευστάθιος Χατζηευθυμιάδης, Αναπληρωτής Καθηγητής ΕΚΠΑ

Κωνσταντίνος Βασιλάκης, Αναπληρωτής Καθηγητής Πανεπιστήμιο Πελοποννήσου

ΕΠΤΑΜΕΛΗΣ ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ

(Υπογραφή)

Παναγιώτης Γεωργιάδης,
Ομότιμος Καθηγητής ΕΚΠΑ

(Υπογραφή)

Ευστάθιος Χατζηευθυμιάδης,
Αναπληρωτής Καθηγητής ΕΚΠΑ

(Υπογραφή)

Κωνσταντίνος Βασιλάκης,
Αναπληρωτής Καθηγητής
Πανεπιστημίου Πελοποννήσου

(Υπογραφή)

Στέφανος Γκρίτζαλης,
Καθηγητής Πανεπιστημίου Αιγαίου

(Υπογραφή)

Γεώργιος Δουκίδης,
Καθηγητής ΟΠΑ

(Υπογραφή)

Παναγιώτης Σταματόπουλος,
Επίκουρος Καθηγητής ΕΚΠΑ

Νικόλαος Τσελίκας,
Επίκουρος Καθηγητής Πανεπιστημίου
Πελοποννήσου

Ημερομηνία εξέτασης __/__/2014

ABSTRACT

Web Services are considered a dominant standard for distributed application composition and communication over the Internet. Consumer applications can locate and invoke complex functionality, through widespread XML-based protocols, without any concern about technological decisions or implementation details on the side of the service provider. Web Services Business Process Execution Language (WS-BPEL) allows designers to orchestrate individual services so as to construct higher-level business processes; the orchestration specification is expressed in an XML-based language, and it is deployed in a BPEL execution engine, made thus available for invocation by consumers.

WS-BPEL has been designed to model business processes that are fairly stable, and thus involve the invocation of web services that are known beforehand. Therefore, the BPEL scenario designer specifies, at the time the scenario is crafted, the exact services to be invoked for the realization of the business process. This setting is however considered inadequate in the context of the current web: many functionalities offered by the services invoked within the scenario (e.g. checking for free rooms in a hotel or booking an air flight) are typically offered by numerous providers (different hotels and flight companies, respectively), and each provider offers its service under different quality of service (QoS) parameters. In this environment, it would be highly desirable for consumers to be able to tailor the WS-BPEL scenario execution according to their QoS requirements. Indeed [2], lists governance for compliance with QoS and policy requirements as an open issue for the SOA architecture.

To tackle this shortcoming, numerous approaches have been proposed, following two main strategies: (i) *horizontal adaptation*, where the composition logic remains intact and the main adaptation task is to select the service and invoke the service best matching the client's QoS requirements; the selected services are substituted for either abstract tasks or concrete service invocations and (ii) *vertical adaptation*, where the composition logic may be modified. The incorporation of run-time adaptation introduces the need for service selection affinity maintenance: service selection affinity refers to cases where a service selection in the context of adaptation implies the binding of subsequent selections (e.g. selecting a hotel reservation from a travel agency dictates that the payment will be made to the same travel agency), to cater for preserving the transactional semantics that invocations to a specific service provider may bear.

QoS-based service selection, however, limits the adaptation criteria to aspects such as cost, availability and performance, not being able to take into account the satisfaction of service users “in the real world”; for instance, an airline company may offer low fares and a short trip duration, but the actual traveling experience may be very poor, an aspect not reflected in QoS attributes and therefore unavailable for the purposes of adaptation. On the other hand, *collaborative filtering* combines the informed opinions of humans (i.e. opinions taking into account the aspect of satisfaction), to make personalized, accurate predictions and recommendations. In the context of collaborating filtering, personalization is achieved by considering ratings of "similar users" (in our case a user is considered to rate a service favorably if she actually uses it or explicitly grades it with a high grade), under the collaborative filtering’s fundamental assumption that if users X and Y have similar behaviors (e.g., buying, watching, listening – in our case, selecting the same services) on some items, they will act on other items similarly.

Under this light, a prominent approach would be to combine QoS-based service selection with collaborative filtering to perform adaptation based on both objective data (QoS attributes) and subjective ratings (quality of experience). Similarly, the combination of content-based filtering with collaborative filtering has been proposed in a number of works.

An additional issue that needs to be taken into account is that in some cases, users may desire to select the exact services to be invoked for some cases and ask for recommendations on other services; for instance, in a holiday planning application, the user may require that reservation is made in a particular hotel, while at the same time asking for a recommendation about the airline. Once a user has made some explicit service selections, the combined approach can be used to make recommendations on the services that the user has not bound to specific providers.

Furthermore, the system may monitor the actual behavior of the services regarding their QoS characteristics (e.g. response time and availability), as opposed to the ones declared by their provider e.g. in an SLA statement, and collect feedback from the users concerning the degree to which they were satisfied by the services proposed to them. These data can be used as additional input to the adaptation process, penalizing services exhibiting negative QoS deviations and/or leaving users dissatisfied, and rewarding services favorably ranked by users and/or being consistent with their declared QoS values (or even surpassing them).

In this thesis, frameworks for providing runtime adaptation for BPEL scenarios are proposed. The adaptation is based on (a) quality of service parameters of available web services (b) quality of service policies specified by users and (c) collaborative filtering techniques, allowing clients to further refine the adaptation process by considering service selections made by other clients.

Moreover, the final proposed framework includes mechanisms for monitoring the behavior of the invoked services regarding their QoS aspects, collecting user satisfaction feedback about the invoked services and taking these data into account when formulating recommendations; it also caters for maintaining the transactional semantics that invocations to multiple services offered by the same provider may bear. Finally, performance, as well as qualitative, metrics for the proposed frameworks are presented, which validate its applicability to operational environments.

SUBJECT AREA: Web Services

KEYWORDS: Adaptation; Equivalent web services; Exception handling; Collaborative filtering; Performance evaluation; Quality of Service (QoS); User rating

ΠΕΡΙΛΗΨΗ

Οι υπηρεσίες διαδικτύου θεωρούνται κυρίαρχο πρότυπο για την επικοινωνία και σύνθεση κατανεμημένων εφαρμογών μέσω του διαδικτύου. Οι εφαρμογές-εξυπηρετούμενοι μπορούν εντοπίσουν και να καλέσουν πολύπλοκες λειτουργίες, χρησιμοποιώντας διαδεδομένα πρωτόκολλα XML, χωρίς να επηρεάζονται από τις τεχνολογίες που χρησιμοποιούνται στην πλευρά του παρόχου των υπηρεσιών ή τις λεπτομέρειες υλοποίησης των υπηρεσιών. Η γλώσσα WS-BPEL (Web Services Business Process Execution Language – Γλώσσα εκτέλεσης επιχειρησιακών διαδικασιών μέσω υπηρεσιών διαδικτύου) επιτρέπει στους σχεδιαστές να ενορχηστρώνουν μεμονωμένες υπηρεσίες, ούτως ώστε να συνθέσουν επιχειρησιακές διαδικασίες υψηλότερου επιπέδου. Η προδιαγραφή της ενορχήστρωσης αυτής εκφράζεται σε μια γλώσσα βασισμένη στην XML και η παραγωγική της λειτουργία ανατίθεται σε μία μηχανή εκτέλεσης BPEL, όπου και καθίσταται διαθέσιμη για κλήση από τους καταναλωτές.

Η WS-BPEL έχει σχεδιαστεί για τη μοντελοποίηση των επιχειρησιακών διαδικασιών που είναι αρκετά σταθερές, και ως εκ τούτου οι υπηρεσίες διαδικτύου που τις συνθέτουν είναι γνωστές εκ των προτέρων. Κατά συνέπεια, ο σχεδιαστής του σεναρίου BPEL καθορίζει, κατά τον χρόνο συγγραφής του σεναρίου, τις συγκεκριμένες υπηρεσίες που θα κληθούν για την υλοποίηση της επιχειρησιακής διαδικασίας. Αυτή η διευθέτηση θεωρείται ωστόσο ανεπαρκής στο πλαίσιο του τρέχοντος διαδικτύου, καθώς πολλές λειτουργικότητες που προσφέρονται από τις υπηρεσίες που καλούνται στα πλαίσια του BPEL σεναρίου (π.χ. έλεγχος για διαθέσιμα δωμάτια σε ένα ξενοδοχείο ή κράτηση μιας αεροπορικής πτήσης) συνήθως προσφέρονται από πολυάριθμους παρόχους (διαφορετικά ξενοδοχεία και αεροπορικές εταιρίες, αντίστοιχα), και καθένας πάροχος προσφέρει τις υπηρεσίες του σε διαφορετικές παραμέτρους ποιότητας υπηρεσιών (QoS). Σε αυτό το περιβάλλον, θα ήταν ιδιαίτερα επιθυμητό για τους καταναλωτές να είναι σε θέση να προσαρμόσουν την εκτέλεση του σεναρίου WS-BPEL, σύμφωνα με τις απαιτήσεις τους σε χαρακτηριστικά ποιότητας υπηρεσίας. Σημειώνεται ότι στο [2], η επιβολή συμμόρφωσης με απαιτήσεις ποιότητας υπηρεσίας και πολιτικές συγκαταλέγεται ανάμεσα στα ανοικτά ζητήματα που αφορούν την υπηρεσιοστρεφή αρχιτεκτονική (SOA).

Για την αντιμετώπιση του ανωτέρω προβλήματος, έχουν προταθεί πολλές προσεγγίσεις, που ακολουθούν δύο κύριες στρατηγικές: (i) την *οριζόντια προσαρμογή*,

όπου η λογική της σύνθεσης δεν τροποποιείται και η κύρια εργασία της προσαρμογής είναι να επιλέξει και να καλέσει την υπηρεσία που ταιριάζει καλύτερα στις απαιτήσεις ποιότητας υπηρεσίας του πελάτη (οι επιλεγμένες υπηρεσίες αντικαθιστούν είτε αφηρημένες είτε συγκεκριμένες κλήσεις υπηρεσιών που υπάρχουν στο αρχικό σενάριο) και (ii) την *κατακόρυφη προσαρμογή*, όπου η λογική της σύνθεσης είναι δυνατό να τροποποιηθεί. Η ενσωμάτωση της προσαρμογής κατά τον χρόνο εκτέλεσης γεννά την ανάγκη για διατήρηση της συνάφειας στην επιλογή των υπηρεσιών: η διατήρηση της συνάφειας αναφέρεται στις περιπτώσεις όπου η πραγματοποίηση μιας επιλογής, στα πλαίσια της προσαρμογής, εισάγει δεσμεύσεις αναφορικά με τις μετέπειτα επιλογές (π.χ. αν επιλεχθεί η πραγματοποίηση κράτησης δωματίου ξενοδοχείου μέσω υπηρεσίας ενός συγκεκριμένου ταξιδιωτικού γραφείου, θα πρέπει και η μετέπειτα πληρωμή να γίνει στην αντίστοιχη υπηρεσία του ίδιου ταξιδιωτικού γραφείου), έτσι ώστε να ληφθεί μέριμνα για τη διατήρηση της λογικής «εκτέλεσης δοσοληψίας» που υπονοούνται από τον προσδιορισμό πολλαπλών κλήσεων υπηρεσιών διαδικτύου προς τον ίδιο πάροχο υπηρεσιών.

Η επιλογή με βάση τα χαρακτηριστικά ποιότητας υπηρεσίας, ωστόσο, περιορίζει τα κριτήρια προσαρμογής σε παραμέτρους όπως το κόστος, τη διαθεσιμότητα και την απόδοση και δεν είναι σε θέση να λάβει υπόψη την ικανοποίηση των χρηστών της υπηρεσίας "στον πραγματικό κόσμο". Για παράδειγμα, μια αεροπορική εταιρεία μπορεί να προσφέρει χαμηλές τιμές και μικρής διάρκειας ταξίδια, αλλά η πραγματική εμπειρία ταξιδιού μπορεί να είναι πολύ κακή, μια πτυχή που δεν αντικατοπτρίζεται στα χαρακτηριστικά ποιότητας υπηρεσίας και ως εκ τούτου δεν είναι διαθέσιμη στα πλαίσια της προσαρμογής.

Από την άλλη πλευρά, το *συνεργατικό φιλτράρισμα* συνδυάζει τις εμπεριστατωμένες απόψεις των ανθρώπων (δηλαδή απόψεις που λαμβάνουν υπόψη την πτυχή της ικανοποίησης), προκειμένου να προβεί σε εξατομικευμένες και ακριβείς προβλέψεις και συστάσεις. Στο πλαίσιο του συνεργατικού φιλτραρίσματος, η εξατομίκευση επιτυγχάνεται λαμβάνοντας υπόψη τις αξιολογήσεις των "παρόμοιων χρηστών" (στην περίπτωσή μας ο χρήστης θεωρείται πως βαθμολόγησε μια υπηρεσία ευνοϊκά αν τη χρησιμοποιεί στην πραγματικότητα ή αν ρητώς τη απέδωσε υψηλή βαθμολογία), σύμφωνα με τη θεμελιώδη αρχή του συνεργατικού φιλτραρίσματος, ότι αν οι χρήστες Χ και Υ έχουν παρόμοιες συμπεριφορές (π.χ., στις αγορές, τα θεάματα ή τα ακούσματά τους - στην περίπτωσή μας, στην επιλογή των ίδιων υπηρεσιών) για ορισμένα αντικείμενα, τότε θα δρουν με παρόμοιο τρόπο και σε άλλα αντικείμενα.

Υπό το πρίσμα αυτό, μια πρόσφορη προσέγγιση θα ήταν να συνδυαστεί η επιλογή με βάση τα χαρακτηριστικά ποιότητας υπηρεσίας με το συνεργατικό φιλτράρισμα, ώστε η προσαρμογή να βασίζεται τόσο σε αντικειμενικά δεδομένα (χαρακτηριστικά ποιότητας υπηρεσίας), όσο και υποκειμενικές αξιολογήσεις (ποιότητα της εμπειρίας). Παρομοίως, σε μία σειρά ερευνητικών εργασιών έχει προταθεί ο συνδυασμός του φιλτραρίσματος με βάση το περιεχόμενο με το συνεργατικό φιλτράρισμα.

Ένα πρόσθετο ζήτημα που πρέπει να εξετασθεί είναι ότι, σε ορισμένες περιπτώσεις, οι χρήστες μπορεί να επιθυμούν την επιλογή συγκεκριμένων υπηρεσιών προς κλήση, ενώ σε ορισμένες άλλες να ζητήσουν συστάσεις για κάποιες υπηρεσίες. Για παράδειγμα, σε μια εφαρμογή σχεδιασμού διακοπών, ο χρήστης μπορεί να απαιτήσει η κράτηση να γίνει σε ένα συγκεκριμένο ξενοδοχείο, ενώ ταυτόχρονα να ζητήσει σύσταση σχετικά με την αεροπορική εταιρεία. Μόλις ένας χρήστης κάνει κάποιες συγκεκριμένες επιλογές υπηρεσιών, η συνδυασμένη προσέγγιση μπορεί να χρησιμοποιηθεί για να διαμορφώσει συστάσεις σχετικά με τις υπηρεσίες που ο χρήστης δεν έχει αντιστοιχίσει σε συγκεκριμένους παρόχους.

Επιπρόσθετα, το σύστημα μπορεί να παρακολουθεί την πραγματική συμπεριφορά των υπηρεσιών αναφορικά με τα χαρακτηριστικά ποιότητας υπηρεσίας που παρέχουν (π.χ. χρόνος απόκρισης και διαθεσιμότητα), σε αντιδιαστολή με αυτά που δηλώνονται από τους παρόχους τους, π.χ. εντός μιας δήλωσης Συμφωνίας Επιπέδου Υπηρεσίας (SLA). Επίσης, το σύστημα δύναται να συλλέγει ανατροφοδότηση από τους χρήστες αναφορικά με τον βαθμό ικανοποίησής τους από τις υπηρεσίες που τους προτάθηκαν. Τα δεδομένα αυτά μπορούν να αποτελέσουν πρόσθετη είσοδο για τη διαδικασία προσαρμογής, η οποία θα απομειώνει την αξιολόγηση καταλληλότητας για τις υπηρεσίες που επιδεικνύουν υποβαθμισμένη ποιότητα υπηρεσίας σε σχέση με τη δηλωθείσα ή/και αξιολογούνται αρνητικά από τους χρήστες, ενώ –αντίστροφα- μπορεί να επιβραβεύει τις υπηρεσίες που επιδεικνύουν συνεπή (ή και υπέρτερα) χαρακτηριστικά ποιότητας υπηρεσίας σε σχέση με τα δηλωθέντα ή/και αξιολογούνται θετικά από τους χρήστες.

Στην παρούσα εργασία, προτείνονται πλαίσια που περιλαμβάνουν την προσαρμογή της εκτέλεσης σεναρίων BPEL σε πραγματικό χρόνο. Η προσαρμογή βασίζεται (α) σε χαρακτηριστικά ποιότητας υπηρεσίας των διαθέσιμων παρεχόμενων υπηρεσιών διαδικτύου (β) σε πολιτικές ποιότητας υπηρεσίας που καθορίζονται από τους χρήστες και (γ) σε τεχνικές συνεργατικού φιλτραρίσματος, επιτρέποντας στους πελάτες να

εκλεπτύνουν περαιτέρω τη διαδικασία προσαρμογής, εξετάζοντας τις επιλογές υπηρεσιών που έγιναν από άλλους πελάτες στο παρελθόν.

Επιπλέον, τα προτεινόμενα πλαίσια μεριμνούν για τη διατήρηση της σημασιολογίας «εκτέλεσης δοσοληψιών» που υπονοούνται από τον προσδιορισμό πολλαπλών κλήσεων υπηρεσιών διαδικτύου προς τον ίδιο πάροχο υπηρεσιών. Το τελικό προτεινόμενο πλαίσιο περιλαμβάνει μηχανισμούς για την παρακολούθηση της συμπεριφοράς των καλούμενων υπηρεσιών, όσον αφορά τις πτυχές των χαρακτηριστικών ποιότητας υπηρεσίας που επιτυγχάνουν, καθώς και τη συλλογή του επιπέδου της ικανοποίησης των χρηστών σχετικά με τις κλήσεις των υπηρεσιών και λαμβάνει αυτά τα δεδομένα υπόψη κατά τη δημιουργία συστάσεων.

Τέλος, παρουσιάζονται μετρικές επιδόσεων, τόσο όσον αφορά τον χρόνο εκτέλεσης όσο και στην ποιότητα των συνθέσεων που διαμορφώνονται κατά την προσαρμογή, για τα προτεινόμενα πλαίσια, οι οποίες καταδεικνύουν τη δυνατότητα εφαρμογής τους σε λειτουργικά επιχειρησιακά περιβάλλοντα.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Υπηρεσίες Διαδικτύου

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Προσαρμογή, Ισοδύναμες υπηρεσίες διαδικτύου, Χειρισμός εξαιρέσεων; Συνεργατικό φιλτράρισμα, Αξιολόγηση απόδοσης, Ποιότητα υπηρεσίας, Αξιολόγηση χρηστών;

ΕΥΧΑΡΙΣΤΙΕΣ

Ολοκληρώνοντας τη διδακτορική μου διατριβή, θα ήθελα αρχικά να ευχαριστήσω τους καθηγητές που την επέβλεψαν.

Πρώτα από όλα, τον Καθηγητή μου, Παναγιώτη Γεωργιάδη, αρχικά για τις αμέτρητες συναντήσεις, μέχρι να καταλήξουμε στο (ενδιαφέρον ερευνητικά και προσωπικά) θέμα αυτής της διατριβής και για την πλήρη κατανόηση και ανεξάντλητη αρωγή του, επιστημονική και ανθρώπινη, σε όλα τα προβλήματα που παρουσιάστηκαν στη διάρκεια της διδακτορικής μου πορείας. Αποτελεί για μένα πρότυπο πανεπιστημιακού δασκάλου και αισθάνομαι χαρά και τιμή που υπήρξα μαθητής του.

Τον Αναπληρωτή Καθηγητή του Πανεπιστημίου Πελοποννήσου Κώστα Βασιλάκη, που δυστυχώς δεν είχα την τύχη να τον έχω καθηγητή στις προπτυχιακές και μεταπτυχιακές μου σπουδές. Ο κος Βασιλάκης με την επιστημονική εμπειρία και εργασία, αλλά συνάμα αμεσότητα και αποτελεσματικότητα, υπήρξε ένας από τους βασικούς παράγοντες ολοκλήρωσης αυτής της εργασίας.

Τον Αναπληρωτή Καθηγητή του Πανεπιστημίου Αθηνών Στάθη Χατζηευθυμιάδη, που, πέραν των άλλων, ήταν ο πρώτος που μου δίδαξε το ενδιαφέρον αντικείμενο των Εφαρμογών Διαδικτύου, θεματική περιοχή στην οποία εντάσσεται η διατριβή μου.

Ένα μεγάλο ευχαριστώ και στον αείμνηστο Καθηγητή Δρακούλη Μαρτάκο, για τις πολύωρες συζητήσεις, επιστημονικές και μη, που με έμαθε να σκέφτομαι λίγο πιο εναλλακτικά σε διάφορες πτυχές της επιστήμης της πληροφορικής και της κοινωνίας, καθώς και για την εμπιστοσύνη που μου έδειξε ως συνεργάτη του σε ερευνητικά προγράμματα καθώς και σε προπτυχιακά και μεταπτυχιακά μαθήματά του.

Θα ήθελα να ευχαριστήσω και τον Χρήστο Καρελιώτη, διδάκτορα του τμήματος Πληροφορικής και Τηλ/νιών, γιατί μέρος της διατριβής αυτής βασίστηκε πάνω στη δική του δουλειά, κυρίως όσον αφορά τον σχεδιασμό και υλοποίηση του συστήματος και το κομμάτι του χειρισμού εξαιρέσεων.

Τέλος, θα ήθελα να ευχαριστήσω τους συμφοιτητές και φίλους μου, Καλλιρόη Αράπογλου, Παρασκευή Ρήγα, Χαρά Σκούρα, Κατερίνα Δρόσου και Γιάννη Μπόλαρη, για τη βοήθεια που μου προσέφεραν σε αυτή τη διατριβή, είτε στα πλαίσια μιας πτυχιακής-διπλωματικής εργασίας είτε από απλό ενδιαφέρον.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ΣΥΝΟΠΤΙΚΗ ΠΑΡΟΥΣΙΑΣΗ ΤΗΣ ΔΙΔΑΚΤΟΡΙΚΗΣ ΔΙΑΤΡΙΒΗΣ

Η διατριβή αυτή αποσκοπεί στη διαμόρφωση αλγορίθμων και τεχνικών για τη δυναμική προσαρμογή της εκτέλεσης των σεναρίων BPEL, λαμβάνοντας υπ' όψιν τόσο τα χαρακτηριστικά ποιότητας υπηρεσίας όσο και την ικανοποίηση άλλων χρηστών από την υπηρεσία αυτή. Σε όλη την έκτασή της ακολουθείται η προσέγγιση της οριζόντιας προσαρμογής, όπου δεν τροποποιείται η λογική της σύνθεσης και η εστίαση τοποθετείται στην επιλογή και κλήση της υπηρεσίας που ταιριάζει καλύτερα στην πολιτική ποιότητας υπηρεσίας που ορίζει ο πελάτης, αλλά και που προκρίνεται βάσει της λογικής του συνεργατικού φιλτραρίσματος, όπου αυτό χρησιμοποιείται. Η εκπόνηση της διατριβής αυτής οργανώθηκε σε πέντε διακριτά στάδια.

Στο πρώτο στάδιο, μελετήθηκε η διεθνής επιστημονική βιβλιογραφία στον τομέα των υπηρεσιών διαδικτύου με επίκεντρο τον τομέα της δυναμικής προσαρμογής της εκτέλεσης επιχειρησιακών διαδικασιών. Από τις προτεινόμενες προσεγγίσεις, ιδιαίτερη έμφαση δόθηκε στη διδακτορική διατριβή με τίτλο «Εκτέλεση σεναρίων BPEL: Δυναμική προσαρμογή και επίλυση εξαιρέσεων με βάση ποιοτικά χαρακτηριστικά» του Δρ. Χρήστου Καρελιώτη, η οποία περιελάμβανε τόσο τον σχεδιασμό, όσο και την υλοποίηση ενός πλαισίου για τη δυναμική προσαρμογή της εκτέλεσης σεναρίων BPEL. Συγκεκριμένα, μελετήθηκε το ASOB (Alternative Service. Operation Binding) Framework ([4], [73], [74] και [75]), ένα πλαίσιο που επιτρέπει την προσαρμογή σε ένα δυναμικό περιβάλλον, όπως αυτό του διαδικτύου, όπου νέες υπηρεσίες μπορούν να εισαχθούν, παλαιές να αποσυρθούν ή οι υπάρχουσες να αλλάξουν τα ποιοτικά χαρακτηριστικά, Το πλαίσιο ASOB επιτυγχάνει εκτέλεση και προσαρμογή διαδικασιών BPEL, λαμβάνοντας υπ' όψιν τα ποιοτικά χαρακτηριστικά των υπηρεσιών (QoS-aware) και τις παραμέτρους της πολιτικής προσαρμογής που έχει θέσει ο χρήστης. Επιπλέον, το πλαίσιο ASOB λαμβάνει μέριμνα για τον χειρισμό εξαιρέσεων/σφαλμάτων που μπορούν να υπάρξουν στο κατανεμημένο περιβάλλον του διαδικτύου, με την επιλογή και εκτέλεση ημιβέλτιστων λύσεων, οι οποίες όμως ανταποκρίνονται στις παραμέτρους της πολιτικής προσαρμογής, προκειμένου να επιτευχθεί η ολοκλήρωση της επιχειρησιακής διαδικασίας. Ειδικότερα, το πλαίσιο αυτό ανιχνεύει και επιλύει τα σφάλματα επιπέδου συστήματος (π.χ. μη διαθεσιμότητα μηχανήματος ή κατάτμηση δικτύου, σε αντιδιαστολή με τα *σφάλματα επιχειρησιακής λογικής*, π.χ. προσπάθεια ανάληψης από λογαριασμό με ανεπαρκές υπόλοιπο) που εμφανίζονται κατά την κλήση των υπηρεσιών, σεβόμενη τους περιορισμούς των ποιοτικών χαρακτηριστικών που καθορίζονται από τον πελάτη

της επιχειρησιακής διαδικασίας. Τέλος, στο πλαίσιο ASOB εξετάστηκαν μέθοδοι για την αντιμετώπιση συντακτικών διαφορών μεταξύ λειτουργικά ισοδύναμων υπηρεσιών, επιτυγχάνοντας έτσι τη διεύρυνση της ομάδας των διαθέσιμων υπηρεσιών για κάθε προσαρμογή.

Όπως αναφέρθηκε, το πλαίσιο αυτό επιτρέπει στους χρήστες να καθορίσουν τις ποιοτικές παραμέτρους που απαιτούν για την εκτέλεση των υπηρεσιών ιστού που συνιστούν μια επιχειρησιακή διαδικασία και αναλαμβάνει τις ενέργειες δυναμικής ανακάλυψης και κλήσης των κατάλληλων υπηρεσιών. Στην εργασία [73], εξετάστηκαν δύο στρατηγικές για την επιλογή της καταλληλότερης υπηρεσίας, μια *άπληστη στρατηγική* (greedy strategy) και μια στρατηγική επιπέδου *σύνδεσης συνεργάτη* (partner-link level).

Προκειμένου να επιτευχθεί η απαραίτητη λειτουργικότητα, το προτεινόμενο πλαίσιο εισάγει δύο πρόσθετες ενότητες λογισμικού σε ένα τυποποιημένο (standard) περιβάλλον εκτέλεσης BPEL. Η πρώτη ενότητα βρίσκεται στο επίπεδο μεσάζοντα (middleware layer) και ονομάζεται ASOB (Alternate Service Operation Binding – Σύνδεση Εναλλακτικών Λειτουργιών Υπηρεσιών), το οποίο αναλαμβάνει τις ακόλουθες λειτουργίες:

(α) τη δυναμική εύρεση λειτουργιών υπηρεσιών (service operations) που παρέχουν συγκεκριμένη λειτουργικότητα και ικανοποιούν δεδομένους περιορισμούς αναφορικά με τα χαρακτηριστικά QoS, τα οποία καθορίζονται από τον πελάτη,

(β) την επιλογή της καταλληλότερης από αυτές τις λειτουργίες, με βάση την πολιτική που έχει ορίσει ο πελάτης,

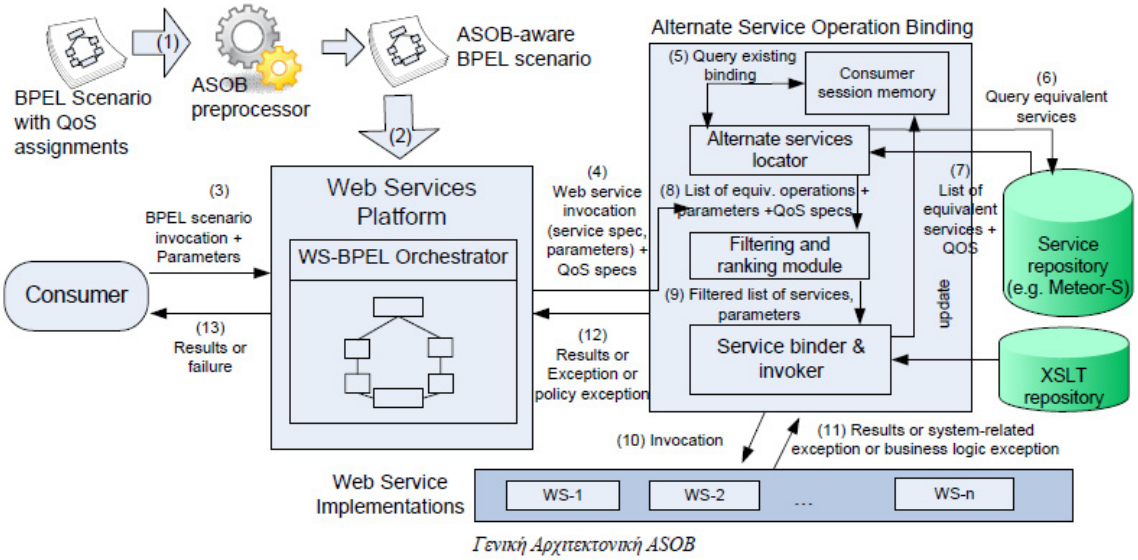
(γ) την κλήση της επιλεγείσας λειτουργίας και τη συλλογή της απάντησης,

(δ) την εφαρμογή των τυχόν απαραίτητων μετασχηματισμών για τα μηνύματα εισόδου και τα αποτελέσματα εξόδου, έτσι ώστε να αντιμετωπιστούν οι συντακτικές διαφορές μεταξύ των σημασιολογικά ισοδύναμων αλλά συντακτικά διαφορετικών διεπαφών υπηρεσιών και

(ε) τη διαχείριση εξαιρέσεων που εμφανίζονται εξ αιτίας σφαλμάτων επιπέδου συστήματος, όπως οι αστοχίες κεντρικών υπολογιστών ή κατάτμηση δικτύου (network partitioning), και της επίλυσής τους με την κλήση άλλων ισοδύναμων λειτουργιών υπηρεσιών.

Η δεύτερη ενότητα που εισάγεται είναι ένας προεπεξεργαστής (preprocessor), με τον οποίο μετασχηματίζονται τα σενάρια BPEL που δημιουργούνται από τους σχεδιαστές

ώστε να είναι δυνατόν να (α) εκτελούνται κλήσεις προς το ενδιάμεσο λογισμικό (middleware) και (β) ενσωματώνονται σε κάθε κλήση όλες οι απαραίτητες πληροφορίες για την επιλογή της καλύτερης λειτουργίας που ταιριάζει στα χαρακτηριστικά QoS που επιλέγονται από τον σχεδιαστή BPEL κατά τη φάση σχεδίασης (BPEL designing phase) και των οποίων οι τιμές ορίζονται από τον πελάτη. Το πλεονέκτημα της χρήσης προεπεξεργαστή για την «προετοιμασία» των σεναρίων BPEL για προσαρμογή της εκτέλεσής τους έγκειται στο ότι με τον τρόπο αυτό η δυναμική προσαρμογή και επίλυση εξαιρέσεων με βάση ποιοτικά χαρακτηριστικά παρουσιάζει τις κάτωθι επιθυμητές ιδιότητες: (α) κάθε μεμονωμένη εκτέλεση BPEL διαδικασίας προσαρμόζεται σε διαφορετικά ποιοτικά χαρακτηριστικά εκτέλεσης (β) η πολιτική εκτέλεσης (execution policy) τίθεται από τον χρήστη, βάσει των αναγκών και προτιμήσεών του, και όχι από τον σχεδιαστή του σεναρίου (γ) μπορεί να εφαρμόζει τεχνικές διατήρησης συνάφειας κατά την επιλογή υπηρεσιών διαδικτύου (δ) μπορεί να εφαρμόζει στρατηγική επιπέδου PartnerLink για την προσαρμογή της εκτέλεσης, ενώ χωρίς την προεπεξεργασία μπορεί να εφαρμοστεί μόνο η άπληστη (greedy) στρατηγική. Ο προεπεξεργαστής παράγει στην έξοδό του ένα σενάριο BPEL έτοιμο για προσαρμογή (adaptation ready) το οποίο τίθεται σε λειτουργία σε μία συνήθη πλατφόρμα εκτέλεσης BPEL. Για περισσότερες λεπτομέρειες σχετικά με τη λειτουργία του προεπεξεργαστή και τον ορισμό της πολιτικής εκτέλεσης, ο ενδιαφερόμενος αναγνώστης παραπέμπεται στο [45]. Ένα σημαντικό πλεονέκτημα αυτής της αρχιτεκτονικής είναι ότι μπορεί να χρησιμοποιηθεί απ' ευθείας σε οποιαδήποτε μηχανή BPEL στην οποία εκτελούνται συνήθη σενάρια BPEL, όπως ακριβώς αυτά έχουν διαμορφωθεί από τους συγγραφείς τους.



Εικόνα1. Γενική Αρχιτεκτονική ASOB

Στο δεύτερο στάδιο, δόθηκε βαρύτητα στη δυναμική επιλογή υπηρεσιών που θα κληθούν κατά την εκτέλεση σεναρίων BPEL βάσει των ποιοτικών τους χαρακτηριστικών. Πιο συγκεκριμένα, σχεδιάστηκε και υλοποιήθηκε ένα πλαίσιο που επαυξάνει την εκτέλεση σεναρίων BPEL με επεκτάσεις για (α) τον καθορισμό των απαιτήσεων σε ποιοτικά χαρακτηριστικά για κλήσεις των υπηρεσιών διαδικτύου σε ένα σενάριο WS-BPEL (β) την προσαρμογή της εκτέλεσης του σεναρίου WS-BPEL, σύμφωνα με τις απαιτήσεις ποιοτικών χαρακτηριστικών, διατηρώντας παράλληλα τη συνάφεια επιλογής υπηρεσιών και (γ) την αυτόματη επίλυση εξαιρέσεων επιπέδου συστήματος. Η αρχιτεκτονική αυτού του συστήματος είναι συμβατή με το πρότυπο SOA, ενώ όλες οι συμπληρωματικές πληροφορίες που απαιτούνται για τους σκοπούς αυτής της προσαρμογής (δηλαδή οι προδιαγραφές των ποιοτικών χαρακτηριστικών για τις μεμονωμένες κλήσεις υπηρεσιών διαδικτύου) εκφράζονται χρησιμοποιώντας το τυπικό συντακτικό της WS-BPEL. Το προτεινόμενο πλαίσιο υποστηρίζει τόσο τις σειριακές όσο και τις παράλληλες δομές εκτέλεσης της WS-BPEL (ετικέτες <sequence> και <flow>, αντίστοιχα), επιτρέποντας την επιτυχή προσαρμογή του κάθε σεναρίου WS-BPEL. Η υποστήριξη των παράλληλων δομών εκτέλεσης αποτελεί μία καινοτομία, σε σχέση με την εργασία [74]. Σε αυτό το στάδιο, έχει υιοθετηθεί η οριζόντια προσέγγιση στην προσαρμογή, δεδομένου ότι (α) επιθυμούμε η προσαρμογή να σέβεται τη λογική σύνθεσης που επέλεξε ο σχεδιαστής του σεναρίου BPEL και (β) με τον τρόπο αυτό καθίσταται δυνατή η χρήση των χειριστών εξαιρέσεων (exception handlers), που μπορεί να έχουν σχεδιαστεί και υλοποιηθεί προσεκτικά από τον σχεδιαστή του σεναρίου BPEL και να έχουν ενσωματωθεί στο αρχικό σενάριο.

Για λόγους συντομίας και χωρίς βλάβη της γενικότητας, στη συνέχεια θα εξετάσουμε μόνο τρία ποιοτικά χαρακτηριστικά: *τον χρόνο απόκρισης* (response time, rt), το κόστος (cost, c) και την διαθεσιμότητα (availability, av). Στο προτεινόμενο πλαίσιο, οι προδιαγραφές ποιοτικών χαρακτηριστικών για μια υπηρεσία στο πλαίσιο του σεναρίου BPEL μπορεί να περιλαμβάνουν και ένα άνω φράγμα και ένα κάτω φράγμα για κάθε ποιοτικό γνώρισμα, καθώς και ένα βάρος, το οποίο καταδεικνύει πόσο σημαντικό θεωρείται το κάθε χαρακτηριστικό από το σχεδιαστή στο πλαίσιο της αυτής της λειτουργίας. Έτσι, οι προδιαγραφές ποιοτικών χαρακτηριστικών ορίζονται με τη μορφή τριών διανυσμάτων, MAX = (rt_{max}, c_{max}, av_{max}), MIN = (rt_{min}, c_{min}, av_{min}) και W = (rt_w, c_w, av_w).

Τα βάρη μπορεί να είναι αρνητικά για να δείξουν ότι οι μικρότερες τιμές είναι προτιμότερες έναντι των μεγαλύτερων, κάτι που αναμένεται για χαρακτηριστικά όπως

το κόστος και ο χρόνος απόκρισης. Για λόγους ευκολίας, υποθέτουμε ότι όλα τα ποιοτικά χαρακτηριστικά είναι κανονικοποιημένα στο διάστημα [0, 10]. Οι τιμές των χαρακτηριστικών ποιότητας για σύνθετες υπηρεσίες που διαμορφώνονται με σειριακή ή παράλληλη εκτέλεση των συνιστωσών υπηρεσιών (s_1, ..., s_n) υπολογίζονται μέσω των τύπων που δίνονται στον ακόλουθο πίνακα.

Πίνακας 1. Υπολογισμός χαρακτηριστικών ποιότητας υπηρεσίας για σύνθετες υπηρεσίες

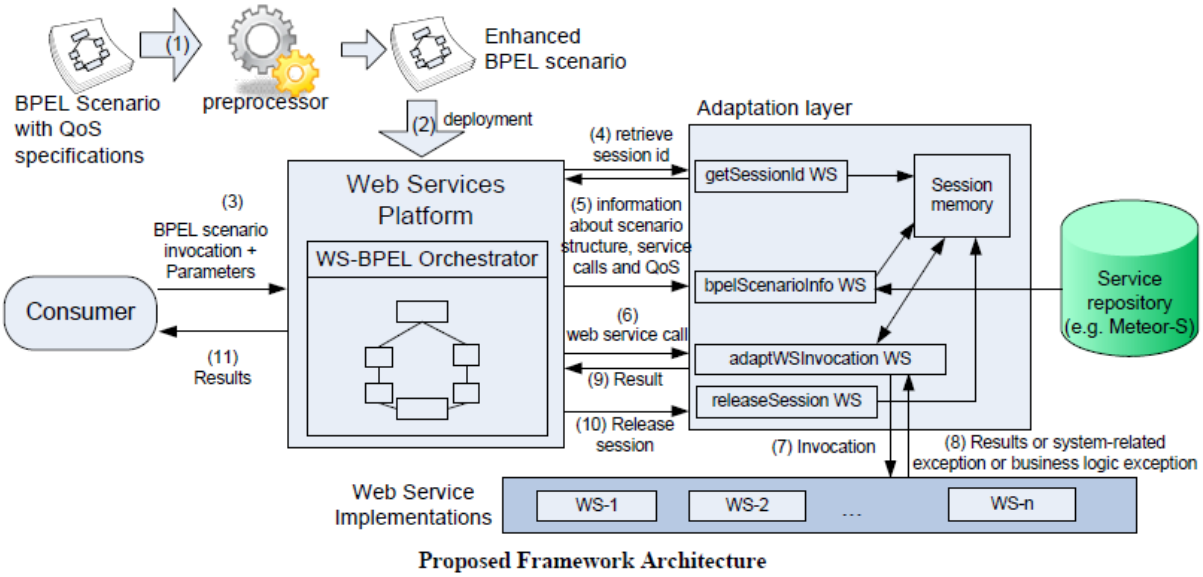
	Χαρακτηριστικό ποιότητας υπηρεσίας		
	responseTime	cost	availability
Σειριακή σύνθεση	$\sum_{i=1}^{n} rt_i$	$\sum_{i=1}^{n} c_i$	$\prod_{i=1}^{n} av_i$
Παράλληλη σύνθεση	$\max_i(rt_i)$	$\sum_{i=1}^{n} c_i$	$\prod_{i=1}^{n} av_i$

Το εμπλουτισμένο σενάριο BPEL περιλαμβάνει και μια κλήση στην υπηρεσία web bpelScenarioInfo, η οποία παρέχεται από το επίπεδο προσαρμογής (middleware), μέσω του οποίου το σενάριο μεταφέρει στο ενδιάμεσο αυτό επίπεδο και τις τιμές όλων των σχετιζόμενων ποιοτικών παραμέτρων (QoSmax_, QoSmin_ και QoSweight_), καθώς και για τη δομή του σεναρίου.

Το προεπεξεργασμένο σενάριο τίθεται σε παραγωγική λειτουργία με την τοποθέτησή του εντός μίας μηχανής εκτέλεσης BPEL και έτσι καθίσταται διαθέσιμο για κλήση από τους πελάτες. Σε κάθε κλήση, του σεναρίου BPEL αρχικά διαβιβάζονται στο επίπεδο προσαρμογής πληροφορίες που αφορούν τη δομή του σεναρίου (κλήσεις υπηρεσιών διαδικτύου που χρησιμοποιούνται και η διάρθρωση εκτέλεσής τους (παράλληλες και σειριακές δομές)), καθώς και τα όρια χαρακτηριστικών ποιότητας υπηρεσίας για κάθε μεμονωμένη κλήση υπηρεσίας διαδικτύου. Κατόπιν το επίπεδο προσαρμογής προβαίνει στη διαμόρφωση του *σεναρίου εκτέλεσης* (execution plan) για τη συγκεκριμένη κλήση του σεναρίου, ως ακολούθως:

1. για κάθε κλήση υπηρεσίας διαδικτύου στο πλαίσιο του σεναρίου WS-BPEL, ανακτώνται από τη βάση δεδομένων (repository) οι υπηρεσίες που είναι λειτουργικά ισοδύναμες προς την καλούμενη. Σημειώνεται ότι στο στάδιο αυτό γίνεται και

φιλτράρισμα των ισοδυνάμων υπηρεσιών, έτσι ώστε να ανακτηθούν μόνο εκείνες που ικανοποιούν τα όρια χαρακτηριστικών ποιότητας υπηρεσίας που ορίζεται για την αντίστοιχη κλήση (μέσω των μεταβλητών *QoSmax_* και *QoSmin_*). Εάν, για κάποια υπηρεσία στο αρχικό σενάριο WS-BPEL, δεν υπάρχουν ισοδύναμες υπηρεσίες που να ικανοποιούν τα όρια που ορίσθηκαν, τότε το επίπεδο προσαρμογής επιστρέφει μία εξαίρεση QoS_PolicyFault στη μηχανή εκτέλεσης BPEL. Ο σχεδιαστής του σεναρίου WS-BPEL μπορεί, τότε είτε να παγιδεύσει την εμφάνιση του σφάλματος χρησιμοποιώντας τους τυπικούς μηχανισμούς WS-BPEL (ετικέτα <catch>) και να προσπαθήσει να το επιλύσει, π.χ. καθιστώντας πιο ελαστικούς τους περιορισμούς και επανεκκινώντας το σενάριο, ή απλώς να ενημερώσει τον αιτούντα πελάτη για την εμφάνιση του σφάλματος.



Εικόνα 2. Αρχιτεκτονική προτεινόμενου πλαισίου

2. το επίπεδο προσαρμογής σχηματίζει όλα τα υποψηφία σχέδια εκτέλεσης για τη συγκεκριμένη εκτέλεση του σεναρίου BPEL. Υποθέτοντας ότι το σενάριο περιέχει N κλήσεις $\{inv_1, inv_2, ..., inv_N\}$ και ότι για κάθε κλήση inv_j υπάρχει ένα σύνολο με ισοδύναμες υπηρεσίες $EQ_j = \{s_{j,1}, s_{j,2}, ..., s_{j,k}\}$, το μέγιστο σύνολο υποψήφιων σχεδίων εκτέλεσης είναι $EQ_1 \times EQ_2 \times ... \times EQ_N$. Από αυτό το μέγιστο σύνολο αφαιρούνται στοιχεία που παραβιάζουν την αρχή της συνάφειας επιλογής, Εάν, κατόπιν της αφαίρεσης, το απομένον σύνολο υποψήφιων σχεδίων εκτέλεσης είναι κενό, τότε επιστρέφεται μια εξαίρεση QoS_PolicyFault, η οποία μπορεί να τύχει χειρισμού, όπως περιγράφηκε ανωτέρω.

3. Για κάθε σχέδιο εκτέλεσης που ανήκει στο σύνολο υποψήφιων σχεδίων εκτέλεσης που διαμορφώθηκε στο βήμα 2, υπολογίζεται μια συνολική βαθμολογία. Η διαδικασία υπολογισμού βαθμολογίας εκτελείται με ανοδικό (bottom-up) τρόπο: αρχικά, το υπολογίζεται η βαθμολογία των μεμονωμένων κλήσεων χρησιμοποιώντας τον τύπο

$sc(inv_i) = rt_i * w_{rt} + c_i * w_c + av_i * w_{av}$

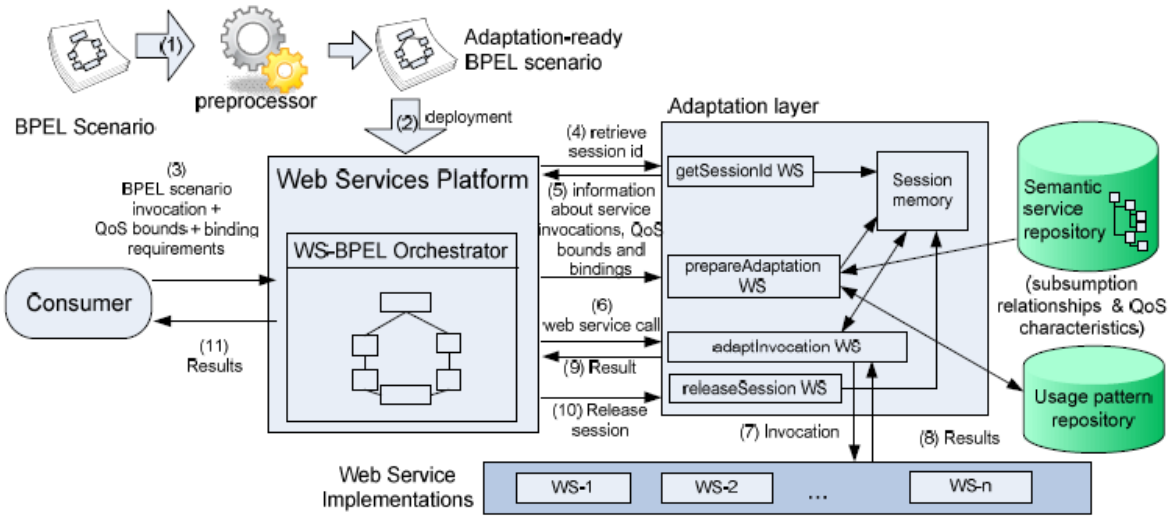
όπου rt_i, c_i και av_i είναι οι τιμές των χαρακτηριστικών ποιότητας υπηρεσίας για την υπηρεσία που είναι υποψήφια για αντικατάσταση της inv_i στο σχέδιο εκτέλεσης, ενώ w_{rt}, κ.λπ. τα βάρη που προσδιορίζονται στη μεταβλητή *QoS_weight_* για την κλήση inv_i. Μετά τον υπολογισμό των βαθμολογιών όλων των μεμονωμένων κλήσεων, οι τύποι που παρουσιάζει ο Πίνακας 1 χρησιμοποιούνται για τον υπολογισμό της συνολικής βαθμολογία του υποψήφιου σχεδίου εκτέλεσης. Τέλος, επιλέγεται προς εκτέλεση το πρόγραμμα με την υψηλότερη βαθμολογία, και οι αντιστοιχίες ανάμεσα στις αρχικές κλήσεις και στις υπηρεσίες που χρησιμοποιούνται στο επιλεγμένο σχέδιο εκτέλεσης αποθηκεύονται στη *μνήμη συνόδου* (session memory), σε συνδυασμό με το τρέχον id της συνόδου. Οι αντιστοιχίες σημειώνονται ως *μη δεσμευμένες* (unbound) θα χρησιμοποιηθούν για επίλυση εξαιρέσεων (exception resolution).

Σημειώνεται ότι είναι δυνατόν αντί να χρησιμοποιηθεί εξαντλητική αναζήτηση, όπως περιγράφηκε ανωτέρω, η διαδικασία εύρεσης του σχεδίου εκτέλεσης με την υψηλότερη βαθμολογία να διαμορφωθεί ως επίλυση ενός προβλήματος ακέραιου προγραμματισμού (integer programming), προσέγγιση η οποία προσφέρει καλύτερες επιδόσεις, ειδικά υπό την παρουσία μεγάλου πλήθους υποψήφιων σεναρίων εκτέλεσης. Η διαμόρφωση του προβλήματος ακέραιου προγραμματισμού παρουσιάζεται στο τέταρτο στάδιο της παρούσας διατριβής.

Στο τρίτο στάδιο, διαμορφώθηκε μια προσέγγιση για την ενσωμάτωση τεχνικών συνεργατικού φιλτραρίσματος στη διαδικασία προσαρμογής της εκτέλεσης σεναρίων WS-BPEL, παρουσιάζοντας τόσο έναν αλγόριθμο προσαρμογής όσο και ένα σχετικό πλαίσιο εκτέλεσης. Ο αλγόριθμος προσαρμογής χρησιμοποιεί τόσο προδιαγραφές χαρακτηριστικών ποιότητας υπηρεσίας, αλλά και σημασιολογικές τεχνικές συνεργατικού φιλτραρίσματος εξατομίκευσης, προκειμένου να αποφασίσει σχετικά με το ποια από τις προσφερόμενες υπηρεσίες ταιριάζει καλύτερα με το προφίλ του πελάτη, ενώ το πλαίσιο εκτέλεσης BPEL περιλαμβάνει διατάξεις για (α) προσδιορισμό των απαιτήσεων των χαρακτηριστικών ποιότητας υπηρεσίας για τις κλήσεις των υπηρεσιών διαδικτύου μέσα σε ένα σενάριο WS-BPEL (β) την επιλογή των υπηρεσιών προς κλήση και (γ) την

προσαρμογή της εκτέλεσης του σεναρίου WS-BPEL σύμφωνα με τις προτάσεις του αλγορίθμου.

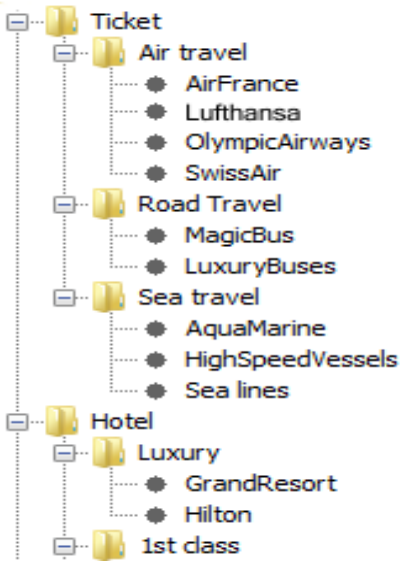
Σε σχέση με το αποθετήριο ισοδύναμων υπηρεσιών (στο οποίο περιλαμβάνονται και οι τιμές των χαρακτηριστικών ποιότητας υπηρεσίας για κάθε διαθέσιμη υπηρεσία διαδικτύου) που χρησιμοποιήθηκε στο προηγούμενο στάδιο, στο στάδιο αυτό χρησιμοποιούνται δύο επιπλέον δεδομένα, τα οποία είναι (α) ένα ιεραρχικό δέντρο υπηρεσιών (κάθε κόμβος αναπαριστά μια κατηγορία ή μια υπηρεσία), όπου οι υπηρεσίες διαδικτύου διαρθρώνονται σε μορφή ταξονομίας, με τις πιο γενικές να είναι τοποθετημένες προς τη ρίζα και τις πιο εξειδικευμένες να είναι τοποθετημένες προς τα φύλλα (οπότε, σε περίπτωσης αίτησης πρότασης για μια υπηρεσία X, υποψήφιες προτεινόμενες υπηρεσίες είναι οι υπηρεσίες που βρίσκονται σε κόμβο από την υπηρεσία X και κάτω – υποκατηγορίες), και ένας πίνακας με παρελθούσες εκτελέσεις σεναρίων WS-BPEL από τους χρήστες (Πίνακας 2: ο πίνακας αυτός καταγράφει αν στα πλαίσια μιας συγκεκριμένης εκτέλεσης έχει κληθεί κάποια συγκεκριμένη λειτουργικότητα -και αν ναι- ποια).



Εικόνα 3. Γενική αρχιτεκτονική πλαισίου υποστήριξης συνεργατικού φιλτραρίσματος

Πίνακας 2 :Πίνακας με παρελθούσες εκτελέσεις σεναρίων WS-BPEL

# exec	Travel	Hotel	Event
1	OlympicAirways	YouthHostel	ChampionsLeague
2	SwissAir	Hilton	GrandConcert
3	HighSpeedVessels	YouthHostel	
4	LuxuryBuses		EuroleagueFinals
5	Lufthansa	YouthHostel	GrandConcert
6	AirFrance	Hilton	
7	SwissAir	YouthHostel	ChampionsLeague



Εικόνα 4. Τμήμα υπαγωγής των σχέσεων για το σενάριο προγραμματισμού ταξιδιού WS-BPEL

Αρχικά, το σύστημα δέχεται ως είσοδο τα κάτω και άνω φράγματα για τα ποιοτικά χαρακτηριστικά (MIN και MAX), τις προδιαγραφές των συνδέσεων (bindings) των λειτουργιών σε συγκεκριμένες υπηρεσίες B = (b_1, b_2, ..., b_n) [b_i == null αν δεν ορίζεται συγκεκριμένη σύνθεση, ειδάλλως το b_i τίθεται ίσο με το id της υπηρεσίας προς την οποία συνδέεται η αντίστοιχη λειτουργικότητα], την προδιαγραφή του ποιες λειτουργίες δεν θα κληθούν O = (o_1, o_2, ..., o_n) [o_i == true αν δεν κληθεί η fi, αλλιώς false], την προδιαγραφή του για ποιες λειτουργίες οποίες ζητούνται συστάσεις R = (r_1, r_2, ..., r_n) [αν για τη λειτουργικότητα f_i ζητείται σύσταση τότε r_i == *κατηγορία λειτουργικότητας*, αλλιώς f_i == null]. Επιπλέον, το σύστημα έχει αποθηκευμένη τη δενδρική δομή με τις σχέσεις (υποκατηγορίες) των υπηρεσιών, συμπεριλαμβανομένων και τιμών των ποιοτικών χαρακτηριστικών των υπηρεσιών αυτών. Όταν ένας χρήστης ζητά μια σύσταση για κάποια λειτουργικότητα A, μπορούμε να χρησιμοποιήσουμε οποιαδήποτε υπηρεσία που προσφέρει την ίδια (ο ίδιος ο κόμβος A) ή οποιαδήποτε περισσότερο συγκεκριμένη λειτουργία (κόμβος του υποδένδρου του) από A.

Τα βήματα κατά την εκτέλεση του σεναρίου BPEL είναι τα εξής:

i. διαμορφώνεται ένα διάνυσμα λειτουργικότητας επιπέδου σεναρίου F = (f_1, f_2, ..., f_n), όπου κάθε fi αντιστοιχεί σε μια λειτουργικότητα που είναι μέρος του σεναρίου WS-BPEL. Οι τιμές των στοιχείων fi καθορίζονται ως εξής: αν η λειτουργία που αντιστοιχεί

στο στοιχείο f_i έχει αντιστοιχηθεί από τον χρήστη σε μια συγκεκριμένη υπηρεσία, τότε η τιμή της f_i θέτεται στο αναγνωριστικό της υπηρεσίας αυτής, ενώ σε όλες τις άλλες περιπτώσεις (δηλαδή, αν η αντίστοιχη λειτουργικότητα δεν θα κληθεί στη συγκεκριμένη εκτέλεση του σεναρίου WS-BPEL ή ζητείται μια σύσταση για αυτή), η τιμή της f_i ορίζεται σε *null*.

ii. για κάθε λειτουργικότητα $funct_i$(request) για την οποία ζητείται μια σύσταση, ο αλγόριθμος ανακτά από τη βάση δεδομένων με παρελθούσες εκτελέσεις σεναρίων τις γραμμές εκείνες οι οποίες περιέχουν είτε μια κλήση σε κάποια υπηρεσία διαδικτύου με ταυτόσημη ή πιο εξειδικευμένη λειτουργικότητα. Αυτές είναι οι μόνες γραμμές οι οποίες είναι χρήσιμες για τη δημιουργία της τρέχουσας σύστασης, δεδομένου ότι περιλαμβάνουν υπηρεσίες που προσφέρουν την αιτούμενη λειτουργικότητα. Αν για παράδειγμα ζητηθεί σύσταση για λειτουργικότητα αεροπορικού ταξιδιού, μόνο οι γραμμές 1, 2, 5, 6 και 7 του πίνακα παρελθουσών εκτελέσεων που παρουσιάζει ο Πίνακας θα ανακτηθούν, δεδομένου ότι όλες οι άλλες γραμμές δεν περιλαμβάνουν λειτουργικότητα αεροπορικού ταξιδιού. Επιπροσθέτως, ένα διάνυσμα λειτουργικότητας επιπέδου αιτήματος F($funct_i$(request)) δημιουργείται, αντικαθιστώντας την τιμή *null* που αντιστοιχούσε στην $funct_i$ με την κατηγορία στην οποία αντιστοιχεί η λειτουργικότητα αυτή. Το νέο αυτό διάνυσμα θα χρησιμοποιηθεί για τον υπολογισμό της ομοιότητας της τρέχουσας αίτησης με τα πρότυπα χρήσης στη βάση δεδομένων (παρελθούσες εκτελέσεις σεναρίων BPEL).

iii. οι γραμμές για τις οποίες οι τιμές χαρακτηριστικών ποιότητας υπηρεσίας για την υπηρεσία $funct_i$(row) δεν ικανοποιούν τα όρια που τέθηκαν μέσω των διανυσμάτων MIN(functi) και MAX(functi) απορρίπτονται, μιας και αυτές οι γραμμές δεν μπορούν να χρησιμοποιηθούν στη σύσταση, εφόσον αφορούν υπηρεσίες των οποίων τα ποιοτικά χαρακτηριστικά τους δεν πληρούν τις απαιτήσεις του καταναλωτή.

iv. Για κάθε γραμμή που διατηρήθηκε από το βήμα 3, υπολογίζουμε ομοιότητα της με την τρέχουσα αίτηση, όπως η αίτηση αυτή αναπαρίσταται από το διάνυσμα F(functi (request)). Η ομοιότητα υπολογίζεται με βάση το δείκτη ομοιότητας Sørensen (εναλλακτικά γνωστός και ως συντελεστής του Dice), σύμφωνα με τον οποία η ομοιότητα δύο συνόλων A = {a_1, a_2, ..., a_n}, B = {b_1, b_2, ..., b_m}, είναι ίση με $S(A,B) = \frac{2|A \cap B|}{|A|+|B|}$ κατάλληλα προσαρμοσμένη για να ταιριάζει σε περιοχή με σημασιολογικές ομοιότητες. Η προσαρμογή ακολουθεί την προσέγγιση που χρησιμοποιείται στο ασαφές (fuzzy) σύνολο υπολογισμού του δείκτη ομοιότητας, όπου η πληθικότητα (cardinality) της τομής των δύο συνόλων (δηλαδή ο αριθμητής στον τύπο του δείκτη ομοιότητας Sørensen) υπολογίζεται ως το άθροισμα των πιθανοτήτων ότι ένα μέλος

ανήκει και στα δύο σύνολα. Αντίστοιχα, όταν υπολογίζουμε ομοιότητα συνόλων, ο αριθμητής του κλάσματος αντικαθίσταται από $2 * \sum_i \mathrm{sim}(a_i, b_i)$, όπου sim($a_i$, b_i) είναι μια μετρική υπολογισμού της ομοιότητας μεταξύ a_i και b_i, ανάλογη με τις προσεγγίσεις που έχουν υιοθετηθεί στην ευθυγράμμιση οντολογιών και αντιστοιχία οντολογικών τομέων. Ως απόσταση ομοιότητας μεταξύ δύο λειτουργιών (υπηρεσίες ή κατηγορίες), υιοθετήθηκε η μετρική:

sim (s_1, s_2) = C - lw * PathLength – NumberOfDownDirection

όπου *C* είναι μια σταθερά που τίθεται σε 8, *lw* είναι το βάθος του επιπέδου για κάθε διαδρομή (path) στο δέντρο της ταξονομίας, *PathLength* είναι το μήκος διαδρομής (ο αριθμός των ακμών από την s_1 μέχρι την s_2) και *NumberOfDownDirection* είναι ο αριθμός των ακμών με κατεύθυνση προς το κάτω μέρος του δένδρου, από την s_1 στην s_2. Στο τέλος του βήματος αυτού, η μετρική ομοιότητας διαιρείται με 8, προκειμένου να κανονικοποιηθεί στο εύρος [0, 1].

v. Τέλος, ο αλγόριθμος διατηρεί μόνο τους *Κ-κοντινότερους γείτονες* (δηλαδή τις γραμμές με τα υψηλότερα σκορ ομοιότητας), τις ομαδοποιεί ως προς την υπηρεσία που υλοποιεί την υπό εξέταση λειτουργικότητα (π.χ. ως προς την υπηρεσία που παρέχει τη λειτουργικότητα *Travel*) και υπολογίζει το άθροισμα των επιμέρους βαθμολογιών σε κάθε ομάδα. Η υπηρεσία που αντιστοιχεί στην ομάδα με το μεγαλύτερο άθροισμα επιλέγεται ως εκείνη που θα παράσχει τη συγκεκριμένη λειτουργικότητα στο πλαίσιο της τρέχουσας εκτέλεσης. Στην εργασία μας, θέσαμε την τιμή της παραμέτρου K (δηλ. του πλήθους των κοντινών γειτόνων που διατηρείται) σε 10, η οποία είναι μια ευρέως χρησιμοποιούμενη τιμή.

Όπως αναφέρθηκε, η επιλογή με βάση τα χαρακτηριστικά ποιότητας υπηρεσίας των υπηρεσιών διαδικτύου στα πλαίσια της προσαρμογής εκτέλεσης σεναρίων BPEL, περιορίζει τα κριτήρια προσαρμογής, αφού δεν είναι σε θέση να λάβει υπόψη την ικανοποίηση των χρηστών της υπηρεσίας "στον πραγματικό κόσμο", ενώ το συνεργατικό φιλτράρισμα συνδυάζει τις εμπεριστατωμένες απόψεις των ανθρώπων (δηλαδή απόψεις που λαμβάνουν υπόψη την πτυχή της ικανοποίησης), προκειμένου να κάνει εξατομικευμένες και ακριβείς προβλέψεις και συστάσεις.

Υπό το πρίσμα αυτό, μια πρόσφορη προσέγγιση θα ήταν να συνδυαστεί η επιλογή με βάση τα χαρακτηριστικά ποιότητας υπηρεσίας με το συνεργατικό φιλτράρισμα, ώστε η προσαρμογή να βασίζεται τόσο σε αντικειμενικά δεδομένα (ποιοτικά χαρακτηριστικά), όσο και υποκειμενικές αξιολογήσεις (ποιότητα της εμπειρίας). Παρομοίως, ο

συνδυασμός του φιλτραρίσματος βάσει περιεχομένου με το συνεργατικό φιλτράρισμα έχει προταθεί σε μια σειρά ερευνητικών εργασιών.

Στο τέταρτο στάδιο παρουσιάστηκε ένα πλαίσιο που περιλαμβάνει την προσαρμογή της εκτέλεσης σεναρίων BPEL σε πραγματικό χρόνο, στο οποίο πλαίσιο η προσαρμογή βασίζεται αφ' ενός στα χαρακτηριστικά ποιότητας υπηρεσίας των διαθέσιμων υπηρεσιών διαδικτύου, με το επιθυμητό επίπεδο ποιότητας των υπηρεσιών να καθορίζεται από τους ίδιους τους χρήστες, και αφ' ετέρου στις τεχνικές συνεργατικού φιλτραρίσματος, επιτρέποντας την περαιτέρω βελτίωση της διαδικασίας προσαρμογής μέσω της εξέτασης των επιλογών υπηρεσιών που έγιναν από άλλους πελάτες στο παρελθόν.

Προκειμένου να επιτευχθεί ο συνδυασμός των μεθόδων που παρουσιάστηκαν στα δύο προηγούμενα στάδια, τροποποιείται ο υπολογισμός της καταλληλότητας κάθε μεμονωμένης υποψήφιας υπηρεσίας $s_{j,l}$ που υλοποιεί τη λειτουργικότητα f_j για χρήση στο σχέδιο εκτέλεσης που αφορά στην συγκεκριμένη κλήση του σεναρίου WS-BPEL, καθώς και ο τρόπος υπολογισμού του προσαρμοσμένου σεναρίου εκτέλεσης.

Ειδικότερα, σε ό,τι αφορά την προσαρμογή με βάση τα χαρακτηριστικά ποιότητας υπηρεσίας, ο υπολογισμός της καταλληλότητας κάθε μεμονωμένης υποψήφιας υπηρεσίας $s_{j,l}$ γίνεται μέσω της *συνάρτησης οφέλους υπηρεσίας* (concrete service utility function), η οποία είναι $U(s_{j,i}) = \sum_{k=1}^{3} \frac{Q_{max}(j,k) - q_k(s_{j,i})}{Q_{max'}(k) - Q_{min'}(k)} * w_k$, όπου $q_k(s_{j,i})$ είναι η τιμή του κ-οστού χαρακτηριστικού ποιότητας υπηρεσίας για την υπηρεσίας $s_{j,i}$ (π.χ. χρόνος απόκρισης, κόστος κ.λπ.), w_k το βάρος που έχει αναθέσει ο χρήστης στο κ-οστό χαρακτηριστικό ποιότητας υπηρεσίας, $Q_{max}(j,k) = \max_{s \in QPA(j)} q_k(s)$ (δηλαδή τη μέγιστη τιμή του χαρακτηριστικού ποιότητας υπηρεσίας κ μεταξύ των πιθανών αναθέσεων υπηρεσίας για τη λειτουργικότητα *j*), και $Q_{max'}(k)$ (αντίστοιχα. $Q_{min'}(k)$) την συνολικά μεγαλύτερη (αντίστοιχα μικρότερη) τιμή του χαρακτηριστικού ποιότητας υπηρεσίας κ στη βάση δεδομένων.

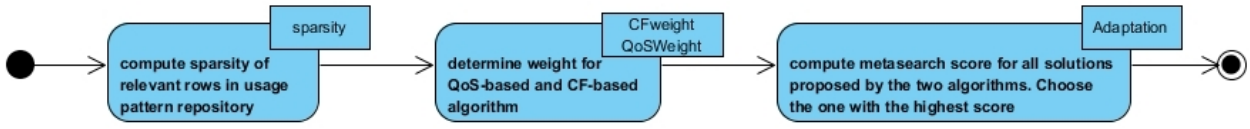
Δεδομένης της συνάρτησης οφέλους, ο υπολογισμός των m-καλύτερων λύσεων μπορεί να οριστεί ως ένα πρόβλημα βελτιστοποίησης ακεραίου προγραμματισμού, στο οποίο ζητείται να ελαχιστοποιηθεί η τιμή της συνολικής συνάρτησης οφέλους που δίνεται από τον τύπο $OUV_{QoS} = \sum_{i=1}^{F} \sum_{j=1}^{T(i)} U(s_{j,i}) * x_{j,i}$, όπου είναι F το πλήθος των λειτουργιών $funct_k$ που χρειάζονται προσαρμογή, και κάθε $x_{j,i}$ παίρνει είτε την τιμή 1 αν η υπηρεσία $s_{j,i}$ επιλέγεται για την υλοποίηση της λειτουργικότητας $funct_j$, ενώ την τιμή 0 αν δεν επιλέγεται. Από τη στιγμή που κάθε λειτουργικότητα $funct_j$ θα υλοποιείται στο τελικό

σχέδιο εκτέλεσης από μία ακριβώς υπηρεσία, η ελαχιστοποίηση της τιμής της συνολικής συνάρτησης οφέλους υπόκειται στον περιορισμό $\sum_{i=1}^{T(j)} x_{j,i} = 1, 1 \leq j \leq F$. Με τον τρόπο αυτό παράγονται τα m-καλύτερα σχέδια εκτέλεσης, και το καθένα από αυτά συνοδεύεται από την τιμή της συνολικής συνάρτησης οφέλους.

Στη συνέχεια, οι τιμές της συνολικής συνάρτησης οφέλους που υπολογίστηκαν, κανονικοποιούνται στο διάστημα [0,1]. Για να επιτευχθεί αυτό πρέπει να υπολογιστεί η ελάχιστη και η μέγιστη τιμή μεταξύ όλων των υποψήφιων λύσεων που βρέθηκαν προηγουμένως και να εφαρμοστεί ο τύπος $normedQoS_{Score_i} = 1 - \frac{QoSscore_i - minQoS_{Score}}{maxQoS_{Score} - minQoSScore}$.

Όσον αφορά τον τρόπο υπολογισμού των πιο πρόσφορων σεναρίων εκτέλεσης με βάση την τεχνική του συνεργατικού φιλτραρίσματος, αρχικά υπολογίζεται ο βαθμός καταλληλότητας της κάθε υποψήφιας υπηρεσίας για υλοποίηση των υπό προσαρμογή λειτουργικοτήτων, βάσει του βαθμού ομοιότητας υποψήφιων σεναρίων εκτέλεσης με τις παρελθούσες εκτελέσεις, όπως περιγράφηκε στο τρίτο στάδιο, και στη συνέχεια διαμορφώνεται ένα πρόβλημα ακέραιου προγραμματισμού για την παραγωγή πλήρων σχεδίων εκτέλεσης, με τον συνδυασμό των επί μέρους υποψήφιών υπηρεσιών. Το πρόβλημα ακέραιου προγραμματισμού ανάγεται στη βελτιστοποίηση της τιμής της συνολικής συνάρτησης οφέλους $OUV_{CF} = \sum_{i=1}^{F} \sum_{j=1}^{L(i)} CFS(s_{i,j}) * x_{i,j}$, όπου CFS($s_{i,j}$) είναι ο βαθμός καταλληλότητας της υποψήφιας υπηρεσίας $s_{i,j}$ για την υλοποίηση της λειτουργικότητας i. Για το πρόβλημα αυτό, υπολογίζονται τα 20 καλύτερα σχέδια εκτέλεσης, υπό τον περιορισμό στον περιορισμό $\sum_{i=1}^{T(j)} x_{j,i} = 1, 1 \leq j \leq F.$, και το καθένα από αυτά συνοδεύεται από την τιμή της συνολικής συνάρτησης οφέλους. Τέλος, η τιμές της συνολικής συνάρτησης οφέλους για τις παραχθείσες λύσεις κανονικοποιούνται στο διάστημα [0,1] βάσει του τύπου $normedCF\_Score_i = \frac{CF\_Score_i - minCF\_Score}{maxCF\_Score - minCF\_Score}$

Η τελευταία φάση σε αυτό το στάδιο είναι η σύνθεση των λύσεων που έχουν προταθεί από τους δύο αλγορίθμους δηλ. (α) τον αλγόριθμο που βασίζεται στα χαρακτηριστικά ποιότητας υπηρεσίας και (β) τον αλγόριθμο που βασίζεται στο συνεργατικό φιλτράρισμα. Η σύνθεση λαμβάνει υπ' όψιν τον βαθμό βεβαιότητας που αποδίδουμε στην πρόταση του αλγόριθμου που βασίζεται στο συνεργατικό φιλτράρισμα, ο οποίος βαθμός εξαρτάται από το πλήθος των δεδομένων από τον πίνακα παρελθουσών εκτελέσεων που είναι χρήσιμα στη συγκεκριμένη προσαρμογή: μικρό πλήθος δεδομένων οδηγεί σε μικρή αξιοπιστία, και συνακόλουθα σε ελάττωση του συντελεστή σημαντικότητας αυτής της πρότασης.



Εικόνα 5. Διάγραμμα δραστηριότητας για το βήμα συνδυασμού

Για τον συνδυασμό των επιμέρους βαθμολογιών, χρησιμοποιείται ο αλγόριθμος υπολογισμού αποτελεσμάτων μετα-αναζήτησης *WCombMNZ*, μιας και παρουσιάζει την καλύτερη απόδοση ανάμεσα σε άλλους αντίστοιχους αλγορίθμους. Πιο συγκεκριμένα, αρχικά θεωρούμε την ποσότητα του σταθμισμένου αθροίσματος βαρών

$$WCombSUM_i = \sum_{j=1}^{m_i} w_j * NormalizedScore_{i,j}$$

όπου το $WCombSUM_i$ είναι η τελική βαθμολογία για ένα αποτέλεσμα i, w_j είναι προκαθορισμένο βάρος σχετικό με τον προτεινόμενο αλγόριθμο j, m_i είναι το πλήθος των μη μηδενικών τιμών του αποτελέσματος i (δηλαδή το πλήθος των αλγορίθμων που προτείνουν αυτό το αποτέλεσμα), και $NormalizedScore_{i,j}$ είναι η κανονικοποιημένη βαθμολογία του αποτελέσματος i που παράγεται από τον αλγόριθμο j). Ακολούθως, υπολογίζουμε την ποσότητα

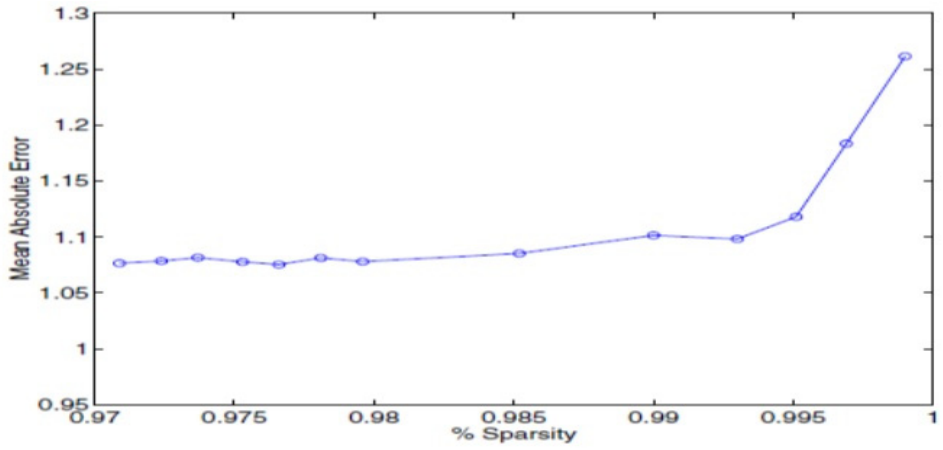
$$WCombMNZ_i = WCombSUM_i * m_i$$

η οποία δίνει και την τελική βαθμολογία.

Λαμβάνοντας υπ' όψιν τα ερευνητικά αποτελέσματα που αποδεικνύουν ότι το συνεργατικό φιλτράρισμα είναι επιρρεπές σε παραγωγή αποτελεσμάτων με χαμηλή ακρίβεια όταν έχουμε υψηλή αραιότητα (sparsity) δεδομένων αναφοράς, κατά τον υπολογισμό της ποσότητας $WCombSUM_i$ χρησιμοποιούμε μεταβλητά βάρη για τους δύο αλγορίθμους, χρησιμοποιώντας τον τύπο που προτείνεται στο [76]:

$$CFweight = \begin{cases} 0{,}40, \text{εάν sparsity} \leq 0.995 \\ \dfrac{0{,}40 * (0{,}999 - sparsity)}{0{,}004}, \text{εάν } 0{,}995 < \text{sparsity} \leq 0{,}999 \\ 0, \text{εάν sparsity} > 0{,}999 \end{cases}$$

$$QoSweight = 1 - CFweight$$

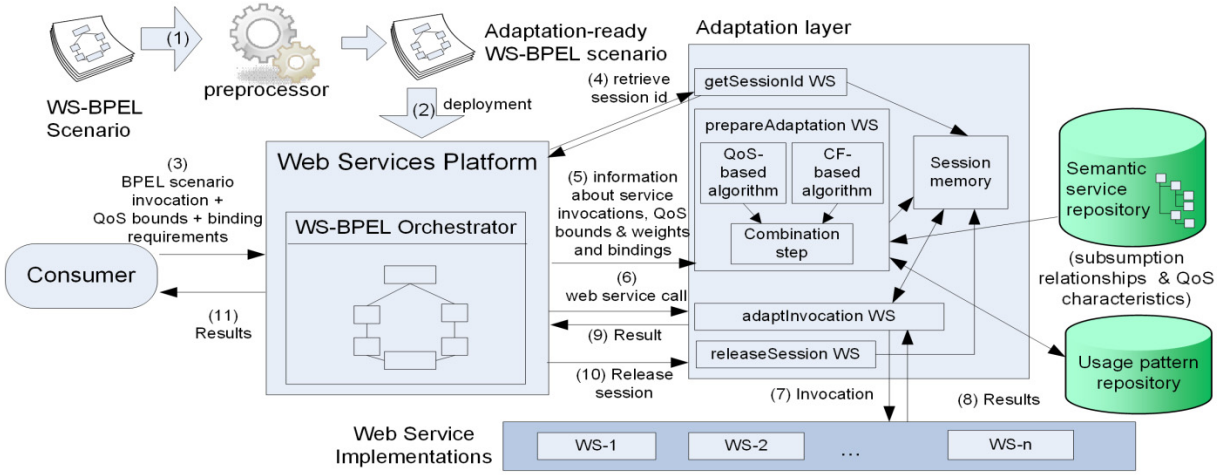


Εικόνα 6. Επίδραση της αραιότητας στην ακρίβεια πρόβλεψης

Παρατηρούμε στην εικόνα 6 ότι για τιμή της αραιότητας μικρότερη του 0,995 το μέσο απόλυτο σφάλμα είναι σχετικά μικρό και ελεγχόμενο, αυξάνεται σχεδόν γραμμικά για τιμές μεταξύ του 0,995 και 0,999, ενώ για τιμή μεγαλύτερη του 0,999 είναι πολύ μεγάλο και θα οδηγήσει σε αναξιόπιστα αποτελέσματα.

Το σχέδιο εκτέλεσης με την μεγαλύτερη τιμή $WCombMNZ_i$ είναι και αυτό που επιλέγεται.

Για την εκτέλεση σεναρίων με βάση τον ανωτέρω αλγόριθμο, εφαρμόζεται η γενική αρχιτεκτονική που εμφαίνεται στην ακόλουθη εικόνα, ενώ οι φάσεις εκτέλεσης του σεναρίου είναι παρόμοιες με αυτές που έχουν περιγραφεί στις παραγράφους που αφορούν στο δεύτερο και το τρίτο στάδιο εκπόνησης της διατριβής.



Εικόνα 7. Αρχιτεκτονική συνδυαστικού πλαισίου

Στο πέμπτο στάδιο, στο προτεινόμενο πλαίσιο εισάγονται μηχανισμοί για την παρακολούθηση της συμπεριφοράς των καλούμενων υπηρεσιών, όσον αφορά τις

πτυχές των χαρακτηριστικών ποιότητας υπηρεσίας τους και τη συλλογή του επιπέδου της ικανοποίησης των χρηστών σχετικά με τις υπηρεσίες που χρησιμοποίησαν. Παράλληλα, οι αλγόριθμοι επιλογής υπηρεσιών επεκτείνονται ώστε να λαμβάνουν υπ' όψιν αυτά τα δεδομένα κατά τη δημιουργία συστάσεων, μεριμνώντας για τη διατήρηση της σημασιολογίας «εκτέλεσης δοσοληψιών» που υπονοούνται από τον προσδιορισμό πολλαπλών κλήσεων υπηρεσιών διαδικτύου προς τον ίδιο πάροχο υπηρεσιών.

Σε αυτήν την προσέγγιση, η βάση δεδομένων με παρελθούσες εκτελέσεις σεναρίων επεκτείνεται ώστε να συμπεριλάβει επιπλέον στήλες, οι οποίες δείχνουν την αξιολόγηση της προτεινόμενης υπηρεσίας από τον ίδιο τον χρήστη. Ο Πίνακας 3 παρουσιάζει ένα παράδειγμα επεκτεταμένου πίνακα.

Πίνακας 3. Επεκτεταμένος πίνακας παρελθουσών εκτελέσεων σεναρίων, ο οποίος περιλαμβάνει στήλες με αξιολόγηση των υπηρεσιών από τους χρήστες

#exec	Travel	R_{travel}	Hotel	R_{Hotel}	Event	R_{Event}
1	OlympicAirways	8	YouthHostel	3	ChampionsLeague	7
2	HighSpeedVessels	null	YouthHostel	null		
3	LuxuryBuses	4		null	EuroleagueFinals	9
4	AirFrance	null	Hilton	null		

Η διαφορά σε σχέση με την προηγούμενη προσέγγιση είναι ότι στο κομμάτι του συνεργατικού φιλτραρίσματος, χρησιμοποιείται η συνάρτηση ομοιότητας συνημίτονου για την ομοιότητα μιας γραμμής $\vec{X}$ που υπάρχει στη βάση (παρελθούσα εκτέλεση) με το τρέχον σενάριο του χρήστη $\vec{Y}$. Η μετρική ομοιότητας μεταξύ των $\vec{X}$ και $\vec{Y}$ διαμορφώνεται έτσι σε

$$r(\vec{X},\vec{Y}) = \frac{\sum_{k=1}^{n}\left(\vec{X}[k] * \vec{Y}[k] * d\left(\vec{X}[k],\vec{Y}[k]\right)\right)}{\|\vec{X}\| * \|\vec{Y}\|}$$

και η πρόβλεψη (βαθμολογία) κάθε σεναρίου βάσης υπολογίζεται από τον συνήθη τύπο πρόβλεψης αξιολόγησης

$$p\left(\vec{R}[k]\right) = \underset{m}{mean}(\vec{R}[m]) + \frac{\sum_{\vec{N}\in raters\left(\vec{R}[k]\right)}\left(\vec{N}[k]\right) * r\left(\vec{R},\vec{N}\right)}{\sum_{\vec{N}\in raters\left(\vec{R}[k]\right)} r\left(\vec{R},\vec{N}\right)}$$

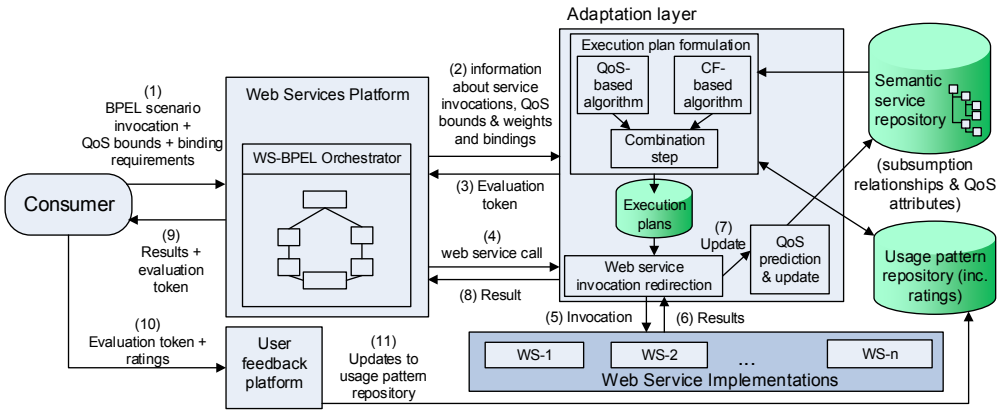
Να σημειωθεί ότι αν κάποιος χρήστης δεν έχει αξιολογήσει τις υπηρεσίες του σεναρίου που κάλεσε, θεωρούμε ως default τιμή το 8 (ίση με το 80% του μέγιστου επιτρεπόμενου), όπως αναφέρεται στο [45] και είναι πλήρως σύμφωνο με τα ευρήματα

του [70], σύμφωνα με το οποίο η πιθανότητα παροχής βαθμολογίας από κάποιον δυσαρεστημένο χρήστη είναι πολύ υψηλή (≥89%).

Η δυνατότητα αξιολόγησης από τους χρήστες μόνο των υπηρεσιών που έχουν χρησιμοποιήσει, διασφαλίζεται με την αποστολή σε κάθε χρήστη που ζητά πρόταση, μετά την εκτέλεση του σεναρίου, ενός *ηλεκτρονικού κουπονιού*, μέσω του οποίου του παρέχεται πρόσβαση στο υποσύστημα της αξιολόγησης, και το οποίο κουπόνι επιτρέπει μόνο την αξιολόγηση των υπηρεσιών που το σύστημα προσαρμογής γνωρίζει ότι ο χρήστης χρησιμοποίησε.

Τέλος, βασιζόμενοι στο [66], ο χρόνος απόκρισης και η διαθεσιμότητα των υπηρεσιών δεν θεωρείται δεδομένος και σταθερός, ίσος με αυτόν που δηλώνεται αρχικά από τους παρόχους των υπηρεσιών. Αντίθετα, το σύστημα παρακολουθεί τις επιδόσεις των υπηρεσιών σε ό,τι αφορά τις ανωτέρω παραμέτρους ποιότητας υπηρεσίας, και χρησιμοποιεί τους τύπους που αναφέρονται στα [66] και [68], για να εκτιμήσει την μελλοντική συμπεριφορά της κάθε υπηρεσίας. Σύμφωνα με αυτά, αν οι παρελθόντες χρόνοι απόκρισης ακολουθούν πρότυπο σταθερής συνάρτησης (με μικρή απόκλιση μεταξύ τους), ο εκτιμώμενος χρόνος ισούται με τον μέσο όρο των παρελθόντων Ν χρόνων απόκρισης. Αν ακολουθούν πρότυπο βαθμιαίας αύξησης/μείωσης, τότε ο εκτιμώμενος χρόνος ισούται με τον μέγιστο/ελάχιστο των αντιστοίχων παρελθόντων χρόνων, ενώ αν το πρότυπο δεν κατατάσσεται στις παραπάνω δύο κατηγορίες, ο εκτιμώμενος χρόνος ισούται με τον μέσο όρο των παρελθόντων Ν χρόνων απόκρισης. Με τον τρόπο αυτό επιτυγχάνεται χρήση δεδομένων με μεγαλύτερη ακρίβεια, με αποτέλεσμα την αντίστοιχη βελτίωση στην ποιότητα των συστάσεων.

Για την εκτέλεση σεναρίων με βάση τον ανωτέρω αλγόριθμο, εφαρμόζεται η γενική αρχιτεκτονική που εμφαίνεται στην ακόλουθη εικόνα, ενώ οι φάσεις εκτέλεσης του σεναρίου είναι παρόμοιες με αυτές που έχουν περιγραφεί στις παραγράφους που αφορούν στο δεύτερο και το τρίτο στάδιο εκπόνησης της διατριβής.



Εικόνα 8. Αρχιτεκτονική πλαισίου που περιλαμβάνει μηχανισμούς ανατροφοδότησης

1. INTRODUCTION

Web Services are considered a dominant standard for distributed application communication over the internet. Consumer applications can locate and invoke complex functionality, through widespread XML-based protocols, without any concern about technological decisions or implementation details on the side of the service provider. Web Services Business Process Execution Language (WS-BPEL) [1] allows designers to orchestrate individual services so as to construct higher level business processes; the orchestration specification is expressed in an XML-based language, and it is deployed in a BPEL execution engine, made thus available for invocation by consumers.

WS-BPEL has been designed to model business processes that are fairly stable, and thus involve the invocation of web services that are known beforehand. Therefore, the BPEL scenario designer specifies, at the time the scenario is crafted, the exact services to be invoked for the realization of the business process. This setting is however considered inadequate in the context of the current web: many functionalities offered by the services invoked within the scenario (e.g. checking for free rooms in a hotel or booking an air flight) are typically offered by numerous providers (different hotels and flight companies, respectively), and each providers offers its service under different quality of service (QoS) parameters. In this environment, it would be highly desirable for consumers to be able to tailor the WS-BPEL scenario execution according to their QoS requirements. Indeed, [2] lists governance for compliance with QoS and policy requirements as an open issue for the SOA architecture.

To tackle this shortcoming, numerous approaches have been proposed, following two main strategies: (i) horizontal adaptation, where the composition logic remains intact and the main adaptation task is to select the service and invoke the service best matching the client's QoS requirements; the selected services are substituted for either abstract tasks (e.g. [3]) or concrete service invocations (e.g. [4]) and (ii) vertical adaptation, where the composition logic may be modified. The incorporation of run-time adaptation introduces the need for service selection affinity maintenance [4]: service selection affinity refers to cases where a service selection in the context of adaptation implies the binding of subsequent

selections (e.g. selecting a hotel reservation from a travel agency dictates that the payment will be made to the same travel agency), to cater for preserving the transactional semantics that invocations to a specific service provider may bear.

QoS-based service selection, however, limits the adaptation criteria to aspects such as cost, availability and performance, not being able to take into account the satisfaction of service users “in the real world”; for instance, an airline company may offer low fares and a short trip duration, but the actual traveling experience may be very poor, an aspect not reflected in QoS attributes and therefore unavailable for the purposes of adaptation. On the other hand, collaborative filtering combines the informed opinions of humans (i.e. opinions taking into account the aspect of satisfaction), to make personalized, accurate predictions and recommendations [5]. In the context of collaborating filtering, personalization is achieved by considering ratings of "similar users" (in our case the a user is considered to rate a service favorably if she actually uses it), under the collaborative filtering’s fundamental assumption that if users X and Y have similar behaviors (e.g., buying, watching, listening – in our case, selecting the same services) on some items, they will act on other items similarly [6].

Under this light, a prominent approach would be to combine QoS-based service selection with collaborative filtering to perform adaptation based on both objective data (QoS attributes) and subjective ratings (quality of experience). Similarly, the combination of content-based filtering with collaborative filtering has been proposed in a number of works (e.g. [7][8]).

However, personalization and adaptation needs may extend beyond the specification of QoS requirements. In some cases, users may desire to select the exact services to be invoked for some cases and ask for recommendations on other services; for instance, in a holiday planning application, the user may require that reservation is made in a particular hotel, while at the same time asking for a recommendation about the airline. Once a user has made some explicit service selections, the combined approach can be used to make recommendations on the services that the user has not bound to specific providers.

In this dissertation a five step approach for adapting the execution of WS-BPEL scenarios taking into account both QoS-based criteria and collaborative filtering techniques is presented, introducing both an adaptation algorithm and an associated execution framework.

Initially, the international scientific bibliography in the field of web services was studied, focusing in the area of dynamic adaptation of business process execution. From the proposed approaches, particular emphasis was given to Dr. Christos Kareliotis' doctoral dissertation, entitled "BPEL Scenario Execution: Dynamic adaptation and exception resolution", which included both the design and implementation of a framework for dynamic adaptation of BPEL scenarios execution. More specifically, the Alternative Service Operation Binding (ASOB) framework was studied ([4], [73], [74] and [75]), a middleware-based framework for system exception resolution, which undertakes the tasks of failure interception, discovery of alternate services and their invocation, was studied.

At the second step, focus was given to QoS requirements, service selection affinity, automatically resolution of system-level exceptions in a QoS requirement-adhering manner, while consideration was given to sequential and parallel execution structures of WS-BPEL (<sequence> and <flow> tags, respectively), allowing for successful adaptation of any WS-BPEL scenario, while at the third step, an approach for integrating semantic-based collaborative filtering techniques into the WS-BPEL execution adaptation procedure with provisions for specifying QoS requirements for invocations of web services within a WS-BPEL scenario, selecting exact services to be invoked and adapting the WS-BPEL scenario executions according to the recommendations of the algorithm, was performed.

At step four, the combination of the two previous steps was performed, by smoothly synthesizing the results given to produce a single result.

Finally, the last step extends the previous one by monitoring, in order to follow the variations of QoS attribute values and by taking into account the users' opinions regarding services they have used.

Each step presents the adaptation algorithm, a thorough example, the proposed framework architecture, as well as a quantitative (in terms of performance) evaluation of the proposed approach.

The final combined adaptation algorithm uses both QoS specifications and semantic-based collaborative filtering personalization techniques to decide which offered services best fit the client's profile. To achieve this goal, the metasearch algorithm paradigm [10] is followed, using two different candidate adaptation ranking algorithms (step one and two), the first examining the QoS aspects only and the second being based on collaborative filtering techniques. The adaptation rankings produced by these two algorithms are combined to generate the overall ranking, which then drives the adaptation. The combination of the results is performed using a weighted metasearch score combination algorithm ([10][11]), however varying weights are used to address issues associated with collaborative filtering, such as cold start (i.e. few entries recorded in the rating database, thus no good matches can be obtained) and gray sheep (i.e. unusual users, which cannot be matched with other users even after the database has been adequately populated) [9].

The adaptation is based on (a) quality of service parameters of available web services (b) quality of service policies specified by users and (c) collaborative filtering techniques, allowing clients to further refine the adaptation process by considering service selections made by other clients.

The final combined proposed BPEL execution framework includes provisions for

(a) specifying QoS requirements for invocations of web services within a WS-BPEL scenario

(b) specifying specific bindings for selecting services and designating which services are subject to adaptation,

(c) adapting the WS-BPEL scenario execution according to the results of the service selection algorithm,

(d) monitoring the behavior of the invoked services regarding their QoS aspects,

(e) collecting user satisfaction feedback about the invoked services and taking these data into account when formulating recommendations and

(f) caters maintaining the transactional semantics that invocations to multiple services offered by the same provider may bear.

This approach follows the horizontal adaptation paradigm since, as noted in [4], horizontal adaptation preserves the execution flow which has been crafted by the designer to reflect particularities of the business process, while it also allows the exploitation of specialized exception handlers.

The rest of the dissertation is structured as follows: In chapter 2 web services execution and adaptation related work is presented, while in chapter 3 QoS concepts and collaborative filtering foundations are presented and analyzed. Chapter 4, 5 and 6 present the QoS-based algorithm and framework, the collaborative filtering-based algorithm and framework and the hybrid algorithm and framework respectively, as well as performance evaluation and qualitative metrics which validate its applicability to operational environments. Chapter 7 presents the final algorithm and framework with provisions for monitoring the QoS parameters of the services and adjusting accordingly the values of the services' QoS attributes, as well as accepting user ratings for the services they have used, which are taken into account by the CF-based algorithm. Finally, chapter 8 concludes this dissertation and outlines future work.

2. WEB Services Execution and Adaptation

As stated above, existing adaptation approaches follow either the horizontal or the vertical adaptation approach. VieDAME [24] performs adaptation of BPEL scenario execution considering QoS parameters; these parameters, as well as the selection strategy are determined beforehand, through pluggable modules. It allows monitoring of BPEL processes according to Quality of Service (QoS) attributes and replacement of existing partner services based on various (pluggable) replacement strategies. The chosen replacement services can be syntactically or semantically equivalent to the BPEL interface. Services can be automatically replaced at runtime without any downtime of the overall system. The solution is implemented with an aspect-oriented approach by intercepting SOAP messages and allows services to be exchanged during runtime with little performance penalty costs, making it suitable for high-availability BPEL environments. In case of interface mismatches, a set of transformers can be specified to handle these mismatches on a SOAP message level, as well. These mechanisms allow a non-intrusive adaptation of partner services within a BPEL process without any downtime of the overall system. VieDAME is also platform-dependent since it relies on extensions of the ActiveBPEL engine.

[53] presents MOSES, a methodology and a software tool implementing it to support QoS-driven adaptation of a service-oriented system. It works in a specific region of the identified problem space, corresponding to the scenario where a service-oriented system architected as a composite service needs to sustain a traffic of requests generated by several users. MOSES integrates within a unified framework different adaptation mechanisms. In this way it achieves a greater flexibility in facing various operating environments and the possibly conflicting QoS requirements of several concurrent users. The basic guideline followed in its definition has been to devise an adaptation methodology that is flexible, to cope with QoS requirements that may come from different classes of users, and (as much as possible) efficient, to make it suitable for runtime operations. To achieve flexibility, a novel approach is presented which allows to integrating within these framework different adaptation mechanisms (service selection and coordination pattern selection) that can be simultaneously used to serve the requests of different users, or even different requests from the same user. To achieve efficiency, a per-flow granularity which also allowed us to formulate the optimal adaptation problem as an LP problem has been considered. Because of the distributed

nature of the SOA environment, the QoS perceived by a user of the composite service can be affected by the performance of the networking infrastructure used to access the selected component services. MOSES, which performs web service selection by formulating and solving a linear programming problem, considering different patterns [par_or and par_and (i.e. concurrent execution of atomic services, with the construct successfully concluding when either one or all service executions finish successfully, respectively) etc.], and monitoring QoS execution during runtime. MOSES assumes that business processes are written as abstract compositions (contrary to our approach where business processes are specified through actual BPEL scenarios, enabling the use of existing ones without any modification), while QoS requirements are stated through an SLA, giving the average value of QoS attributes, not allowing distinct QoS specifications per web service invocation.

[23] presents AgFlow, a QoS-aware middleware supporting quality driven Web service composition, which revises the execution plan in order to conform the user's QoS constraints. AgFlow may operate either using global planning, in which the execution plan is revised in order to conform the user's QoS constraints, or using local optimization, in which optimization is made on individual task basis, using the Simple Additive technique Weighting [4] to select the optimal service for a given task. The main features of the AgFlow system are a service quality model to evaluate overall quality of Web services and alternative service selection approaches for executing composite services. AgFlow has been implemented as a platform that provide tools for defining service ontologies, specifying composite services using statecharts and assigning services to the tasks of a composite service.

Several works exist which optimize business processes, but also implement exception handling mechanisms, to provide solutions in volatile run-time environments.

[77] presents a framework which achieves dynamic service selection from statically composed Web Services in a manner that considers the optimum execution path regarding the QoS parameter: response time. It uses autonomic computing concepts to formulate execution plans for business processes, taking into account QoS parameters, monitoring QoS violations at runtime and also handling these violations and tries to optimize existing Web Services composition structure with the help of the Controller Agent Web Service.

[45] performs horizontal QoS-based adaptation, taking into account the sequential and parallel execution structures within the BPEL scenario. It presents a framework enabling the WS-BPEL designers to specify the QoS requirements for the service invocations included in the WS-BPEL scenarios and the subsequent adaptation of the scenarios' execution to these specifications. The proposed framework also supports the automatic resolution of system-level exceptions, while it also caters for the maintenance of service selection affinity, ascertaining that transactional semantics of service invocations are preserved. The proposed framework includes a scenario preprocessing step, before the scenario is deployed and made available for invocations, and an *adaptation layer*, which undertakes the tasks of optimizing the execution plan for the WS-BPEL scenario, adapting the execution and resolving exceptions.

0 considers adaptation in the context of exception resolution (i.e. automatically substituting a failed service by an equivalent one, in order to guarantee successful completion of the business process) in BPEL scenarios by locating and invoking web services having the same skills as the failed ones. The code for intercepting faults and invoking alternate web services is automatically generated and injected into the BPEL scenario by a preprocessor. Identification of same skilled web services is based on both functional and qualitative attributes, where functional attributes are required to be equivalent, while the comparison between quantitative attributes is policy-driven. The proposed approach exploits the exception handling mechanisms of BPEL and can thus be used with any available BPEL orchestrator.

Work in 0 considers service selection in the presence of QoS constraints and aiming to minimize an objective function for the entire orchestration employing both brute force (OPTIM_S) and heuristic (OPTIM_HWEIGHT) algorithms. More specifically, *QSSAC* algorithm is proposed to solve the problem of service selection which is based on service clustering. Service clustering, which is able to cluster candidate atomic services with similar functionality into classes, has a great effect on the efficiency and effect of this algorithm. According to the clustering results recorded in the service clustering information, *QSSAC* algorithm can reduce the number of atomic services of each task by choosing the best services from each class at first. Moreover, characteristic of each candidate can be obtained through the service clustering information. Then improvement of the result of each step of combination can be seen, by choosing a certain number of suitable services which are measured by utility values and characteristics. Finally, it provides three strategies to re-select services for tolerating

service changes and failures. This work presents the service selection algorithms but does not propose an architecture on top of which the adaptation can be realized, while it additionally does not consider service selection affinity.

Regarding services' functionality, some approaches need only record which services are *equivalent,* such as [16] which proposes a multi-tier architecture, TailorBPEL, which supports the tailoring of personalized BPEL-based workflow compositions, enabling end-users to tailor personalized BPEL-based workflow compositions at runtime, while others use more elaborate schemes, and such as the one described in [26] according to which a service A may be related a service B through one of the following subsumption relationships: *exact* (services have identical functionality, e.g. they both book a flight), *plugin* (service A is more specific than B and can thus be used in its place; for instance, if service A is "book a flight" and service B is "book transport", then we can use A in the place of B since A actually books a transport), *subsume* (service A is more general than B and therefore cannot always be used in its place; e.g. if A books a transport and B books a flight, we cannot always use A in the place of B since A may lead to booking a trip by ship instead of a trip by plane) and *fail* (none of the *exact*, *plugin* and *subsume* relationships holds). The subsumption relationships effectively broaden the pool of services that can be used in the context of adaptation (a service A can be unconditionally substituted by a service B if *A exact B* or *A plugin B*, as opposed to strict equivalence where substitution is only possible when *A exact B*), providing thus more flexibility in the formulation of the execution plan.

METEOR-S [43] is a suitable infrastructure for such service discovery activities, employing ontologies where service inputs, outputs and QoS aspects are described. Execution under the METEOR-S framework is also monitored to allow for updating of QoS attributes such as response time and failure rate. METEOR-S can be used for storing and querying both service functionalities and QoS characteristics. Since METEOR-S adopts ontologies for representing information about services, it is powerful enough to express both the service equivalence notions and the subsumption relationships. Also, [43] presents a methodology and a set of algorithms for Web service discovery based on three dimensions: syntax, operational metrics, and semantics. This approach allows for web service discovery not only based on functional requirements, but also on operational metrics. The development of mechanisms for the discovery of web services is based on operational metrics allows organizations to translate their vision into their business processes more efficiently, since e-workflows can be designed

according to QoS requirements, goals, and objectives. To facilitate the discovery and posteriori integration of Web service into workflows, an approach based on the use of ontologies to describe workflow tasks and Web service interfaces is proposed and an algorithm has been devised and a prototype implemented to discover and facilitate the resolution of structural and semantic differences during the integration process with an e-workflow. The algorithm uses a feature-based model to find similarities across workflow tasks and Web service interfaces and finally the system determines and evaluates the best mapping between the outputs

WSMO [15] may provide the foundations for modeling, storing and reasoning on the relevant web service functional and non-functional aspects. Its development process is divided into three steps: (i) the creation of a Platform Independent Model (PIM), expressed in UML; this model describes business rules and functionalities of the application, and exhibits a high degree of platform independence; (ii) then, the production of a Platform Specific Model (PSM), mapping the PIM into a specific platform; (iii) finally, the application's generation.

[16] presents WSMoD (Web Services MOdeling Design) a Web Service Modeling Design methodology for quality of service (QoS)-based Web Services. The approach of the methodology consists in considering non-functional aspects derived from business requirements, as well as functional requirements, and incorporating and refining them throughout the design process. WSMoD extends the model driven architecture proposed by [15], in two directions. The first one, according to the well-known software engineering principle "divide et impera", adds in the definition of the platform independent model a specific methodological step for the definition of the non-functional requirements. Users and channels, which support the interaction between web services and users, should be considered first-class concepts in the analysis and design of new services. The second extension concerns the platform independent model to represent quality aspects and the specific context related to user profile and channel constraints. This kind of context is still independent from a specific deployment platform, so that it can be adapted to different providers offering new web services. The goal of this extension of Platform Independent Model is to improve the design of software applications by avoiding modelling choices that are not deployable in real provisioning environments.

Note that none of above approaches incorporates CF techniques to enhance the quality of the adaptation. In the collaborative filtering domain, several methods have been

proposed, however their incorporation in the WS-BPEL execution adaptation process has not been considered.

[29] surveys collaborative filtering, focusing on is use within the adaptive web. Interestingly, [29] lists the basic properties of domains suitable for collaborative filtering classified under three major categories (data distribution; underlying meaning; data persistence) and all the listed properties hold for the context of WS-BPEL scenario adaptation.

[46] presents an approach for integrating collaborative filtering techniques into the WS-BPEL execution adaptation procedure, introducing both an adaptation algorithm and an associated execution framework. The adaptation algorithm uses both QoS specifications and semantic-based collaborative filtering personalization techniques to decide on which offered services best fit the client's profile, while the BPEL execution framework includes provisions for specifying QoS requirements for invocations of web services within a WS-BPEL scenario, selecting exact services to be invoked and adapting the WS-BPEL scenario executions according to the recommendations of the algorithm. This approach follows the horizontal adaptation paradigm, as well, while it also allows the exploitation of specialized exception handlers. The novel features of this proposal lie in the use of the collaborative filtering in the adaptation procedure, as well as in the creation of the execution framework. However, [46] uses very limited QoS-based criteria (only a lower and an upper bound for each QoS attribute), hence it runs the risk of formulating solutions whose QoS is much inferior to the optimal composition QoS that can be attained, especially in cases that CF has known issues (e.g. cold start and gray sheep).

0 presents an evaluation of collaborative filtering algorithms. Collaborative filtering is in many cases combined with another personalization technique, namely content based filtering; [82] and [83] are examples of such approaches. However, content-based filtering needs items with content to analyze [29], and in the context of WS-BPEL scenario adaptation we cannot use the items' content (responses of individual web services) since the responses of equivalent web services are identical (apart from invocation-specific data; this is mandatory or they would not be equivalent), hence they are not useful for choosing between different service implementations. Moreover, the responses contain personal data (and some of them sensitive data, e.g. credit card numbers or health-related data), and therefore they cannot be retained. Thus, in this

work we use only the collaborative filtering approach, retaining only service usage patterns, in an anonymized form.

In order to perform hybrid QoS/CF-based adaptation or exception resolution, [76] presents an approach, which adapts the execution of WS-BPEL scenarios taking into account both QoS-based criteria and collaborative filtering techniques, introducing both an adaptation algorithm and an associated execution framework. The adaptation algorithm uses both QoS specifications and semantic-based collaborative filtering personalization techniques to decide on which offered services best fit the client's profile. To achieve this goal, the metasearch algorithm paradigm [10] is followed, using two different candidate adaptation ranking algorithms, the first examining the QoS aspects only and the second being based on collaborative filtering techniques. The adaptation rankings produced by these two algorithms are combined to generate the overall ranking, which then drives the adaptation. The combination of the results is performed using a weighted metasearch score combination algorithm [10][11], however varying weights are used to address issues associated with collaborative filtering, such as cold start (i.e. few entries recorded in the rating database, thus no good matches can be obtained) and gray sheep (i.e. unusual users, which cannot be matched with other users even after the database has been adequately populated) [9].

As far as the combination of results deriving from different algorithms is concerned, [57] and [58] outline the IBM ILOG CPLEX, a tool for solving linear optimization problems, commonly referred to as Linear Programming (LP) problems (Maximize/Minimize - subject to - with upper/lower bounds). ILOG CPLEX also can solve several extensions to LP (a) Network Flow problems, a special case of LP that CPLEX can solve much faster by exploiting the problem structure, (b) Quadratic Programming (QP) problems, where the LP objective function is expanded to include quadratic terms and (c) Mixed Integer Programming (MIP) problems, where any or all of the LP or QP variables are further restricted to take integer values in the optimal solution and where MIP itself is extended to include constructs like Special Ordered Sets (SOS) and semi-continuous variables.

Finally, none of above approaches take into account an important aspect of QoS attributes, that their values may vary, according to server load, network conditions or other relevant factors. To this end, prediction models have been developed e.g. [66] and [68], to allow a more accurate estimation of QoS attribute values; this increased accuracy can be used in adaptation systems to improve the quality of the adaptation. In

order to perform adaptation and/or exception resolution, all approaches employ some means to formally specify the necessary properties of the services and importantly (a) their functionality and (b) their QoS attribute values.

Work in [67] includes mechanisms for monitoring the behavior of the invoked services regarding their QoS aspects, collecting user satisfaction feedback about the invoked services and taking these data into account when formulating recommendations. This approach follows the horizontal adaptation paradigm, as well, while it also allows the exploitation of specialized exception handlers. [67], also proposes a framework which provides means for monitoring the QoS parameters of the services and adjusting accordingly the values of the services' QoS attributes, as well as accepting user ratings for the services they have used, which are taken into account by the CF-based algorithm. The proposed framework is complemented with an execution architecture for enacting the adaptation, which adopts the middleware approach, with an adaptation layer intervening between the BPEL execution platform and the web services and arranging for redirecting service invocations to the services selected by the adaptation algorithm.

3. QOS CONCEPTS AND COLLABORATIVE FILTERING FOUNDATIONS

In the following subsections the concepts and underpinnings, from the areas of QoS and collaborative filtering, which are used, are summarized.

3.1 QoS concepts and definitions

QoS may be defined in terms of attributes [12][13], while typical attributes considered are cost, response time, availability, reputation, security etc. [14]. For conciseness purposes, in this chapter we will consider only the attributes responseTime (rt), cost (c) and availability (av), adopting their definitions from [15]. Extension of the framework to include additional attributes is straightforward, thus we have no loss of generality.

In the proposed frameworks, the QoS specifications for a service within the BPEL scenario may include an upper bound and a lower bound for each QoS attribute, i.e. for service s_j included in a WS-BPEL scenario, the designer formulates two vectors $MIN_j(min_{rt,j}, min_{c,j}, min_{av,j})$ and $MAX_j(max_{rt,j}, max_{c,j}, max_{av,j})$. Additionally the designer formulates a weight vector $W = (rt_w, c_w, av_w)$, indicating how important each QoS attribute is considered by the designer in the context of the particular operation invocation (effectively, weight element values are multiplied by the value of the respective QoS dimension of the service composition, and these products are then summed to produce a total score for the composition). The values of the QoS attributes are assumed to be expressed in a “larger values are better” encoding, hence a service having response time = 7 actually responds in less time than a service with response time = 3 (better response time). Note that weights apply to the whole composition, rather than to individual services, since they reflect the perceived importance of each QoS attribute dimension on the process as a whole, and not its constituent parts [23].

For convenience reasons, we assume that all QoS attributes are normalized in the range [0, 1]; value range normalization is a typical approach in works considering QoS-based adaptation (e.g. [23][24]). Note that this implies that a total ordering relationship is required in the range of the QoS attributes; this is always the case with attributes such as response time and availability, however some QoS attributes are more complex: for instance [25] identifies seven dimensions related to security, and some security mechanism S1 may outperform some other security mechanism S2 in some

dimensions but lag behind S2 in other dimensions. This could be tackled by decomposing the security QoS attribute in seven distinct attributes, one for each dimension, ensuring thus the existence of the total ordering relationship in the range of each attribute. In the rest of this chapter, we will only consider cases that the total ordering relationship exists.

In order to compute the QoS of services composed through sequential or parallel execution from constituent services s_1, …, s_n having QoS attributes equal to (rt_1, c_1, av_1), …, (rt_n, c_n, av_n), respectively, the formulas given in Table 1 [16] can be used. These computation formulas are in line with the Q-algebra operations introduced in [17] and generalized in [18].

As we can see from Table1, the response time of a sequential composition is equal to the sum of its components' response time, while the response time of a parallel composition is equal to the maximum value. This difference is important in the adaptation process, since different search strategies should be employed to optimally adapt the scenario to the client's QoS specification. Consider for example the case of a BPEL scenario includes sequential invocations to *A* and *B*, which is invoked with the setting *W*=(1, 1, 0) for both service invocations. If the repository of available services were as listed in Table2, then the adaptation engine should select services (A_2, B_2), with this composition scoring 2 = (sum (rt_{A2}, rt_{B2})∗1 +sum(c_{A2}, c_{B2})∗1), a score higher than any other composition. In a parallel composition however, the adaptation engine should select (A_2, B_1), since these provide an overall score of 1.7 =(*max*(rt_{A2}, rt_{B1})∗1 +sum(c_{A2}, c_{B1})∗1), as opposed to 1.3 of (A_2, B_2).

Table 1: QoS of composite services

	QoS attribute		
	responseTime	cost	availability
Sequential composition	$\sum_{i=1}^{n} rt_i$	$\sum_{i=1}^{n} c_i$	$\prod_{i=1}^{n} av_i$
Parallel composition	$\max_i(rt_i)$	$\sum_{i=1}^{n} c_i$	$\prod_{i=1}^{n} av_i$

Table 2: Sample repository contents

Service	responseTime	cost	availability
A_1	0.6	0.5	0.8
A_2	0.8	0.4	0.7
B_1	0.2	0.5	0.9
B_2	0.7	0.1	0.7

3.2 Subsumption relation representation

In order to perform adaptation, either QoS-based or collaborative filtering-based, which services implement the same functionality must be represented, and are thus candidate for invocation when this particular functionality is needed. In this work, subsumption relations between services [18] to represent this information is used. In [26] the following subsumption relations are defined:

A exact B, iff A provides the same functionality with B.

A plugin B, iff A provides more specific functionality than B. For instance B could provide travel, whereas A could provide air travel; in this case A could be used whenever the functionality of B is needed, since it delivers (a specialization of) the functionality delivered by B.

A subsume B, iff A provides more generic functionality than B. In this case A cannot unconditionally be used whenever the functionality of B is needed. For instance B could provide air travel while A provides travel, and using a travel service (instead of an air travel one) could result to transportation by car, which does not comply with the functionality of B.

A fail B, in all other cases; in this case, A cannot be substituted for B.

The rationale for adopting the subsumption relations are as follows:

subsumption relations effectively broaden the pool of services that can be used in the context of adaptation (a service A can be unconditionally substituted by a service B if A exact B or A plugin B, as opposed to strict equivalence where substitution is only

possible when A exact B), providing thus more flexibility in the formulation of the execution plan.

Subsumption relations are suitable for supporting the task of similarity metric computation, which is essential for collaborative filtering-based adaptation.

[26] and [27] address the representation of subsumption relations between service categories (or abstract tasks, in horizontal adaptation terminology) using trees, with generic service categories being located towards the tree root and specific service categories being placed towards the leafs (the references above apply this arrangement also to concepts corresponding to service parameters). Since in this work we are interested not only in service categories but in concrete services also (because these will be actually invoked in the context of WS-BPEL scenario execution), the tree scheme used in [26] and [27] is extended by considering not only is-a arcs in the tree (general/specific categories) but also instance-of arcs: an arc is drawn in the tree between service category C and concrete service S, if and only if S implements exactly the functionality specified by category C. To illustrate this representation, let us consider the case of a travel planning WS-BPEL scenario containing the following activities: ticket booking, hotel booking, and event attendance. In this case the subsumption relations, including categories and concrete services could be arranged as shown in figure 1 (categories are denoted using a folder icon; concrete services are denoted using a bullet mark).

Below, how the exact and plugin subsumption relations (the ones sufficient for unconditional service substitution) can be computed using the tree representation, are listed. Since the goal of the adaptation is to select the concrete services to be invoked in place of an abstract task or a concrete service, only the rules for the cases where the right-hand side operand is a concrete service are given. In the following, c represents a category, while s1 and s2 represent services:

- Rule C_{ext}: c exact s_1 iff c is the immediate parent of s_1 (e.g. Air travel and SwissAir in figure 4-1).
- Rule C_{plg}: c plugin s_1 iff c is an ancestor of s_1 (e.g. Ticket and SwissAir in figure 4-1).
- Rule S_{ext}: s_1 exact s_2 iff $\exists$ c: c is the immediate parent of s_1 and c is the immediate parent of s_2 (e.g. AirFrance and SwissAir in figure 4-1).
- Rule S_{plg}: s_1 plugin s_2 iff $\exists$ c: c is the immediate parent of s_1 and c plugin s_2 (e.g. SportsTicketBooker and NBAFinals in figure 4-1).

In all other cases, unconditional substitution cannot occur hence a fail result for the operands is computed. Listing 1 and Listing 2 illustrate the computation of the relevant subsumption relation between a service category and a service or between two services, respectively, given the tree representation suggested in [26] and [27]. The representation of subsumption relations illustrated in figure 1 can be trivially extended to accommodate the QoS attribute values of concrete services, by simply attaching to each concrete service node s the vector $QoS_S=(rt_s, c_s, av_s)$ corresponding to the particular service's QoS metrics.

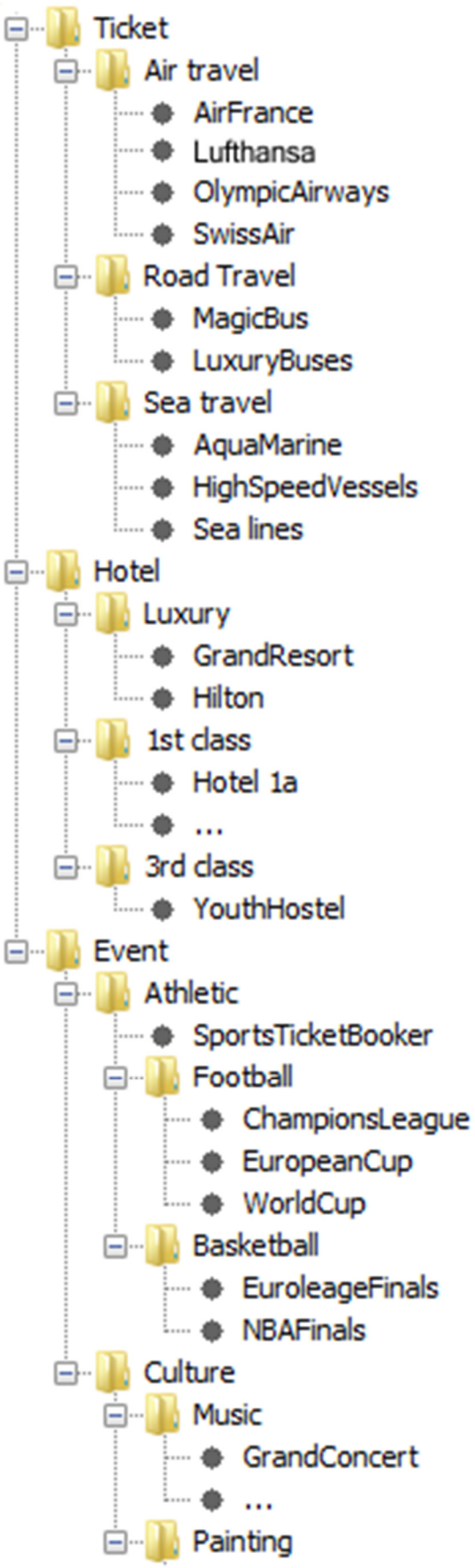


Figure 1: Subsumption relations for the travel planning WS-BPEL scenario

The exact subsumption relation is equivalent to contract equivalence [26]. Further [37] uses the subcontract preorder concept, denoted as a ⩽b, to denote that b offers “more capabilities than a, and therefore b can be substituted for a”; the preorder concept is therefore analogous to the plugin subsumption relation.

```
subsumptionRelCatSvc(ServiceCategory c, ConcreteService s, SubsumptionTree st) {
    SubsumptionTreeNode categNode = findNodeInTree(c, st);
    SubsumptionTreeNode svcNode = findNodeInTree(s, st);
    if (svcNode.parent == categNode)
        return exact;
    end if
    while (svcNode.parent ≠ null)
        svcNode = svcNode.parent;
        if (svcNode == categNode)
            return plugin;
        end if
    end while
    return fail;
}
```

Listing 1: Computing the subsumption relation between a service category and a concrete service

```
subsumptionRelSvcSvc(ConcreteService s1, ConcreteService s2, SubsumptionTree st)
{
    SubsumptionTreeNode svc1Node = findNodeInTree(s1, st);
    SubsumptionTreeNode svc2Node = findNodeInTree(s2, st);
    if (svc1Node.parent == svc2Node.parent)
        return exact;
```

```
    end if
    while (svc2Node.parent ≠ null)
        svcNode = svcNode.parent;
        if (svcNode2 == svc1Node.parent)
            return plugin;
        end if
    end while
    return fail;}
```

Listing 2 : Computing the subsumption relation between two concrete services

The information regarding the QoS data and the subsumption relations can be stored in repository structures like the ones provided by OPUCE [37]. The population of the repository with QoS values can be performed either by using existing web service QoS datasets (e.g. the QWS dataset [38][39][40] or the WS-DREAM dataset [41][42]), or by using the methodologies described in [38][39][41] to compute the QoS attribute values for any desired set of target web services. In all cases, the execution framework can monitor the actual quality of service delivered by the individual web services upon their invocation, and update the QoS attribute values in the repository accordingly, as described in [43].

3.3 Designations on specific service bindings

In the considered environment, the WS-BPEL consumer is allowed to make specific service bindings, i.e. request that some particular operation is performed using a designated service. For instance, in the travel planning WS-BPEL scenario discussed above, the consumer may request that ticket reservation is performed through the SwissAir service. The consumer may also designate that some functionality included in the WS-BPEL scenario is not executed; for instance, a tourist may not want to attend any event. Typically, the WS-BPEL code will examine input parameters and decide using a conditional execution construct (<switch>) whether to invoke the functionality or not.

Finally, functionalities that are neither explicitly bound to a specific service implementation, nor are designated as “not to be executed” are subject to adaptation, using the algorithm described in section 3.

3.4 Usage patterns repository

In order to perform collaborative filtering, a system needs to record evaluations (ratings) or choices (actions taken) made by users. This information is typically stored in a ratings matrix [29], where each row corresponds to a user and each column corresponds to an item, and the value of the (i, j) cell of this matrix indicates how user i has rated item j (or whether user i has taken action j). Since our aim is to adapt WS-BPEL scenario executions based on past user choices, each row of the ratings matrix in the proposed system corresponds to a particular WS-BPEL scenario execution, while each column corresponds to a concrete web service implementation. The value of a cell (i, j) is 1 if the service corresponding to column j were executed in the i^{th} execution, and null otherwise. We will call this matrix usage patterns repository. The execution framework can maintain this usage patterns repository by recording in an appropriate store which services were used in any particular execution of a BPEL scenario.

For notational convenience, in the rest of this dissertation the usage patterns repository in a more compact form will be represented: taking into account that each functionality included in the WS-BPEL scenario will result to at most one invocation of some service that implements it (it may result to zero invocations, if the consumer has designated that this piece of functionality should not be executed. Note that the adaptation procedure presented here does not take into account loops, an aspect which will be addressed as part of our future work.), for each row, at most one of the columns corresponding to the services implementing the functionality will have a value, while the other columns will be null. Thus the information within the usage patterns repository using one column for each distinct functionality of the WS-BPEL scenario can be equivalently denoted, with the value of a cell (i, j) designating the service used to deliver the specific functionality if the functionality corresponding to column j were delivered in the context of the ith execution, and null otherwise. Table 3 illustrates a usage patterns repository for the travel planning WS-BPEL scenario, using the compact notation. Note that the sparse representation is used internally, as is the case with all collaborative filtering algorithms. In executions 1, 2, 5 and 7 all functionalities were invoked, while in the remaining executions some functionalities were omitted, as per user request.

Table 3: Example usage patterns repository

# exec	Travel	Hotel	Event
1	OlympicAirways	YouthHostel	ChampionsLeague
2	SwissAir	Hilton	GrandConcert
3	HighSpeedVessels	YouthHostel	
4	LuxuryBuses		EuroleagueFinals
5	Lufthansa	YouthHostel	GrandConcert
6	AirFrance	Hilton	
7	SwissAir	YouthHostel	ChampionsLeague

The usage patterns repository is populated by the execution adaptation architecture, described in section 5-3. When the execution adaptation architecture formulates the execution plan for a particular invocation (i.e. selects the concrete services to be selected to deliver the requested functionalities), the corresponding row is inserted in the usage patterns repository.

4. QOS-BASED APPROACH

In this chapter a framework that extends BPEL execution with provisions for (a) specifying QoS requirements for invocations of web services within a WS-BPEL scenario (b) adapting the WS-BPEL scenario execution according to the QoS requirements while maintaining service selection affinity and (c) automatically resolving system-level exceptions in a QoS requirement-adhering manner is presented. The system architecture is compliant with the SOA paradigm, while all additional information needed for adaptation purposes (i.e. the QoS specifications for individual web service invocations) are expressed in standard WS-BPEL syntax. The proposed framework also considers both sequential and parallel execution structures of WS-BPEL (<sequence> and <flow> tags, respectively), allowing for successful adaptation of any WS-BPEL scenario.

4.1 The QoS-based algorithm

The QoS-based algorithm initially identifies the services which are candidate to be used for delivering functionalities in the context of the current WS-BPEL scenario, according to their QoS attribute values, as well as the QoS bounds specified by the user for the particular invocation, and subsequently computes a score for each candidate composition. Figure 2 illustrates the steps of the QoS-based algorithm in the form of an activity diagram and figure 3 describes the algorithm in pseudocode, while the following paragraphs provide details on the actions taken within each step.

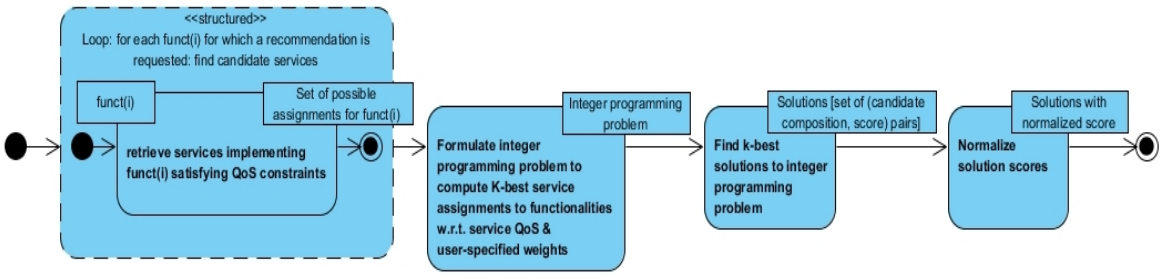


Figure 2: Activity diagram for the QoS-based algorithm

/* QoS-based adaptation algorithm pseudocode

Assumption:

- Scenario includes functionalities (i.e. invocations to services) f_1, f_2, …, f_n

Inputs:

- MIN and MAX (lower and upper bounds for QoS attributes) and weights
- Specification of bindings of functionalities to concrete services B=(b_1, b_2, …, b_n) [b_i == null if no binding provided, else id of service)
- Specifications of functionalities not to be invoked O=(o_1, o_2, …, o_n) [o_i == thrue if f_i should not be invoked, false otherwise]
- Specifications of functionalities for which recommendations are requested R=(r_1, r_2, …, r_n) [r_i == category of functionality if a recommendation is requested for f_i, null otherwise]
- Subsumption relation tree, including the QoS attribute value of services
- Scenario structure (information about which functionalities are executed sequentially and which in parallel)

Outputs:

- List of <execution plan, score> pairs, with “execution plan” binding services that recommendation is asked for to concrete services

*/

/* 1. For each functionality r_i that a recommendation is requested for, compute the set of services QPA(i) that deliver this functionality (as can be determined by the subsumption relations) and satisfying QoS thresholds. */

for each $f_i \in R$, $f_i \neq$ null: QPA(i) = {s $\in$ serviceRepository: (f_i exact s $\vee$ f_i plugin s) $\wedge$ MIN ≤ QoS(s) ≤ MAX

/* 2. formulate and solve integer programming problem, obtaining the top-20 execution plans, taking into account the QoS attribute values and the weights of the QoS attributes */

```
ipp = formulateIntegerProgrammingProblem(QPA)
QoS_solutions = obtainTop20Solutions(ipp)
/* 3. Finally, normalize scores */
QoS_proposal = normaliseScores(QoS_solutions);
```

Figure 3: Pseudocode for the QoS-based algorithm

1. For each distinct functionality $funct_i$ within the WS-BPEL scenario which is subject to adaptation (i.e. for each service invocation that the user (a) has not designated an exact service binding and (b) has not designated that the functionality should not be invoked), the service implementations that deliver this functionality are retrieved from the repository. Recall that service functionalities are represented using the subsumption relations tree, and in terms of this tree the retrieval of services implementing a specific functionality is equivalent to retrieving services satisfying either the Sext or the Splg rule (i.e. services with either identical or more specific functionality, compared to the requested one). Equivalent services retrieval is further constrained by the QoS thresholds specified in the respective invocation. Thus, for each functionality $funct_i$, this step computes a set of possible concrete service assignments QPA(i)={$s_{i,1}$, $s_{i,2}$, …, $s_{i,T(i)}$}. Formally, QPA(i) is computed as follows:

$$QPA(i) = \{s \in Repository: (funct_i\ exact\ s\ \vee funct_i\ plugin\ s)\ \wedge\ [(min_{rt,i} \leq rt_s \leq max_{rt,i})\ \wedge\ (min_{c,i} \leq c_s \leq max_{c,i}) \wedge\ (min_{rel,i} \leq rel_s \leq max_{rel,i})]\}$$

   If, for some service that is subject to adaptation, no candidates satisfying the thresholds are found (i.e. QPA(i)=∅), then the algorithm terminates producing no results, indicating thus that the constraints specified by the user cannot be satisfied.

   In our example, services Swiss Air, Olympic Airways, Air France and Lufthansa, satisfy the Sext rule, however service Olympic Airways fails to meet the QoS constraint regarding the maximum cost. Hence QPA(AirTravel)={Swiss Air, Air France, Lufthansa}.

2. The algorithm computes the m-best solutions, with respect to the QoS value, with each solution being a set of concrete service assignments to the functionalities for which a recommendation has been requested. Since the service consumer's QoS bounds per functionality are an input to the adaptation algorithm, service selection can proceed by exploiting these constraints, performing local selection so as to efficiently compute the optimal concrete service assignments for the functionalities under adaptation [47][37][48][49][50][51][52]. In this work, we adopt the concrete service utility function used in [47][49], which is:

$$U(s_{j,i}) = \sum_{k=1}^{3} \frac{Q_{max}(j,k) - q_k(s_{j,i})}{Q_{max'}(k) - Q_{min'}(k)} * w_k$$

here $q_k(s_{j,i})$ is the value of the k^{th} QoS attribute of concrete service $s_{j,i}$ (the first QoS attribute being response time, the second one being cost and the third one being availability), w_k being the weight assigned to the kth QoS attribute, $Q_{max}(j,k) = \max_{s \in QPA(j)} q_k(s)$ [i.e. the maximum value of QoS attribute k among possible concrete service assignments for functionality j], and $Q_{max'}(k)$ [resp. $Q_{min'}(k)$] being the overall maximum (resp. minimum) value of QoS attribute k within the repository. Note that services with high QoS values have low utility function values. Given the utility function, the computation of the m-best solutions can be formulated as an integer programming optimization problem as follows: minimize the overall utility value given by:

$$OUV_{QoS} = \sum_{i=1}^{F} \sum_{j=1}^{T(i)} U(s_{j,i}) * x_{j,i}$$

where F is the number of functionalities $funct_k$ requiring adaptation, and each $x_{j,i}$ is a binary variable taking the value 1 if $s_{j,i}$ is selected for delivering functionality $funct_j$, and the value 0, otherwise. Since each functionality $funct_j$ is delivered in the final execution plan by exactly one concrete service, the minimization of the overall utility value is subject to the constraint

$$\sum_{i=1}^{T(j)} x_{j,i} = 1, 1 \leq j \leq F$$

Note that in the above problem formulation an additive formula is used for all QoS attributes, while the formulas listed in Table 1 for composite QoS service computation are not all additive; in particular the availability attribute of a composite service is shown in table 4-1 to be computed as a product, while the response time of a parallel composition is computed as the maximum of the constituent services' response time. In order to transform the non-additive formulas to additive ones, suitable for use in the context of an integer programming problem, we have employed the following techniques:

- regarding the product functions used for computing the availability of a composite service, we have adopted the approach used in [23][53][54], according to which the logarithm of availability, rather than the availability itself is used when writing expressions. Through the use of the logarithm, the equality

$$log(\prod_i availability(i)) = \sum_i log(availability(i))$$

is exploited, and therefore the product function is transformed to a sum function. Observe also that in the computation of the $U(s_{j,i})$ function listed above, the literal value any QoS attribute (including availabiliity) is normalized taking into account the minimum and maximum values of the attribute, the range of the normalized attribute value used in the utility function is always in the range [0, 1].

- regarding the max function used in computing the response time of a parallel composition, it is possible to follow the method described in [55] to transform an objective minimization problem where the objective contains the max function (termed a *minimax objective*) to an integer programming problem. In brief, the transformation procedure of an objective function containing the term

$$\max_{k \in K} \sum_{j \in J} c_{kj} x_j$$

involves the introduction of a new variable z representing the above term, and the introduction of the following extra constraints:

$$\sum_{j \in J} c_{kj} x_j \leq z, \forall k \in K$$

A relevant example is given in [56]. It is however worth noting that the IBM ILOG CPLEX optimizer 12.6.0.0 [57] used in the experiments, directly supports the usage of the max operator within the objective function, even in a recursive structure (i.e. the operands of a max operator may be max operators themselves, as would be the case with nested flow constructs in a BPEL scenario), hence there was no need to apply the transformation described above; the transformations can be used, if the integer programming optimization software used does not support the direct use of the max operator in the objective function.

If service selection affinity needs to be maintained between functionalities $funct_i$ and $funct_j$, then an additional constraint is added to the problem as follows:

i. Let QPA(i)={$s_{i,1}$, $s_{i,2}$, …, $s_{i,L(i)}$} be the possible concrete service assignments for $funct_i$ and QPA(j)={$s_{j,1}$, $s_{j,2}$, …, $s_{j,L(j)}$} the possible concrete service assignments for $funct_j$, computed in step 1, above. Without loss of generality, it is assumed that the first k services in QPA(i) and QPA(j) are offered by the same service provider (i.e. services for which the host part of their endpoint address [1] is

identical), while the remaining services are offered by different providers, or more formally:

$$\begin{cases} provider(s_{i,a}) = provider(s_{j,a}), 1 \le a \le k \\ provider(s_{i,x}) \ne provider(s_{j,y}), 1 \le x \le k, 1 \le y \le k, x \ne y \end{cases}$$

In this case, implementing the functionalities ($funct_i$, $funct_j$) with any choice of services ($s_{i,a}$, $s_{j,a}$) for $1 \le a \le k$ leads to maintenance of service selection affinity, while any other choice (i.e. $a > k$, or implementing the functionalities with a choice of services ($s_{i,x}$, $s_{j,y}$) with $x \ne y$) leads to a failure in maintaining service selection affinity.

ii. The constraints

$$x_{i,a} - x_{j,a} = 0, 1 \le a \le k$$

are added to the problem; under the presence of these constraints, a service $x_{i,a}$ implementing functionality $funct_i$ and provided by service provider a will be selected if and only if for the realization of functionality $funct_j$ the service $x_{j.a}$ provided by the same provider is selected too. For optimization purposes, all elements corresponding to services $s_{i,b}$ and $s_{s,b}$: $b > k$ are removed from the model, since they cannot be part of any feasible solution.

This procedure can be generalized for cases that service selection affinity needs to be maintained between any number of functionalities: if service selection affinity needs to be maintained between functionalities $funct_{i1}$, $funct_{i2}$,…, $funct_{if}$, then the constraints

$$x_{i1,a} - x_{i2,a} = 0, 1 \le a \le k$$

…

$$x_{i1,a} - x_{if,a} = 0, 1 \le a \le k$$

Similarly to the case of maintaining service selection affinity between two functionalities, it is assumed without loss of generality that

$$\begin{cases} provider(s_{i1,a}) = provider(s_{i2,a}), 1 \le a \le k \\ \dots \\ provider(s_{i1,a}) = provider(s_{if,a}), 1 \le a \le k \\ provider(s_{p,x}) \ne provider(s_{q,y}), 1 \le p \le f, 1 \le q \le f, p \ne q, 1 \le x \le k, 1 \le y \le k, x \ne y \end{cases}$$

Subsequently, the integer programming problem is solved, and the k best solutions are obtained. Obtaining the k best solutions can be achieved by solving the problem once, and then adding a constraint rendering the obtained solution infeasible (for the IBM ILOG CPLEX optimizer [57] used in the experiments, this procedure is described in [58]). In this implementation, k was set to 20; the value of 20 has proven to be sufficient even for applying metasearch techniques to web search engines [59] (where the size of each individual list is considerably higher than the number of alternatives in our case), i.e. further increasing the number of solutions that each voting algorithm provides to the combination phase, does not improve the quality of the result computed by the combination step.

In our example, a single functionality requires adaptation (AirTravel); the utility function for this functionality is

$$U(s_{AirTravel,i}) = \sum_{k=1}^{3} \frac{Q_{max}(AirTravel,k) - q_k(s_{AirTravel,i})}{Q_{max'}(k) - Q_{min'}(k)} * w_k$$

Substituting the values for $Q_{max}(k)$, $Q_{max}(k)$, $Q_{min}(k)$ and w_k, the utility function can be rewritten as:

$$U(s_{AirTravel,i}) = \underbrace{\frac{5 - rt(s_{AirTravel,i})}{5-4} * 0.2}_{runtime} + \underbrace{\frac{8 - cost(s_{AirTravel,i})}{9-2} * 0.5}_{cost} + \underbrace{\frac{9 - av(s_{AirTravel,i})}{9-4} * 0.3}_{availability}$$

The utility function values for services in QPA(AirTravel) are as shown in Table 4. Since assignment is performed only for a single functionality, each possible solution obviously consists of selecting one of the candidates in QPA(AirTravel) to implement the AirTravel functionality. In terms of the integer programming problem solution, the variable $x_{AirTravel,j}$ for this service will be equal to 1 and the variables $x_{AirTravel,k}$ for the remaining services will be 0. The value of the overall utility function for a solution, coincides with this of the sole service selected in this solution. All possible solutions are kept, since their number is less than 20.

Table 4: Utility function values and IP problem solutions for services in QPA(AirTravel)

IP problem solution	Selected $s_{AirTravel,i}$	$U(s_{AirTravel,i})$	OUV_{QoS}
$x_{AirTravel,SwissAir}=1, x_{AirTravel,AirFrance}=0,$ $x_{AirTravel,Lufthansa}=0$	Swiss Air	0.231	0.231
$x_{AirTravel,SwissAir}=0, x_{AirTravel,AirFrance}=1,$ $x_{AirTravel,Lufthansa}=0$	Air France	0.100	0.100
$x_{AirTravel,SwissAir}=0, x_{AirTravel,AirFrance}=0,$ $x_{AirTravel,Lufthansa}=1$	Lufthansa	0.657	0.657

3. The scores of the solutions produced by step 2, above, are then normalized in the range [0,1], to enable the combination step to function correctly. Normalization is performed by first computing the minimum and the maximum execution plan scores among all solutions formulated in step 3, above, and then applying the formula

$$normedQoS_{Score_i} = 1 - \frac{QoSscore_i - minQoS_{Score}}{maxQoS_{Score} - minQoSScore}$$

[37] (adapted from [11]; since the target of step 2 was to minimize the overall utility function OUVCF, here the normalization formula suggested in [11] is subtracted from 1, in order to assign the normalized score 1 to the best solution and the normalized score 0 to the worst). The output of the algorithm is the set of candidate execution plans, with each execution plan being tagged with the relevant normalized score (QoS-score).

In this example, the normalized solution scores are as shown in table 5.

Table 5: Normalized solution scores

$s_{AirTravel,i}$	Normalized score
Swiss Air	0.764
Air France	1.000
Lufthansa	0.000

4.2 Proposed framework

The proposed framework extends the typical WS-BPEL execution scenario by accommodating two additional modules, namely the BPEL scenario preprocessor and an adaptation layer. The BPEL scenario preprocessor accepts as input the original BPEL scenario and produces as output a transformed BPEL scenario, which has been transformed to (i) transmit to the adaptation layer the QoS specifications and the necessary information regarding the scenario structure (i.e. sequential and parallel flows) to enable the adaptation of the scenario and (ii) redirect all service invocations to the adaptation layer, where they will be appropriately redirected to the most suitable service provider. The adaptation layer is deployed as a middleware, positioned between the BPEL execution engine and the web service providers and arranges for redirecting the web service invocations to the web service implementations best matching the client's QoS specifications, and intercepting and resolving system-level exceptions. The adaptation layer also offers two utility web services, the first one assigning session identifiers to BPEL scenario executions and the second one accepting the information regarding QoS specifications and the scenario structure. Figure 4 presents the overall architecture of the proposed framework; in the following section (a) the specification of QoS parameters in the BPEL scenario (b) the operation of the preprocessor and (c) the operation of the adaptation layer are elaborated.

4.3 Specifying QoS information in the scenario

The first step towards enabling the QoS-based adaptation is the specification of the required QoS for service invocations. In order to provide this feature in a WS-BPEL compliant fashion, the proposed framework adopts the following conventions:

1. the designer should include in each invoke construct in the WS-BPEL scenario the optional attribute name, assigning distinct names to the invoke constructs.
2. for the invoke construct having name invX, the designer should use the WS-BPEL variables QoSmax_invX, QoSmin_invX and QoSweight_invX, which define the respective QoS specifications for the particular invocation. The BPEL designer may set the values for these variables after inspecting input parameters to the

scenario (e.g. “choose=cheapest”), arranging thus for tailoring the QoS specification to the invoking user’s preferences.

Figure 5 presents an excerpt of a BPEL scenario setting QoS specifications for an invocation (named invoke1). Since variable QoSmin_invoke1 is not set, no lower bounds will be considered for the QoS attribute values in the process of adapting the particular web service invocation. Additionally, since the QoSmax_invoke1 does not include a setting for the availability QoS attribute, no upper bound for this attribute value will be considered.

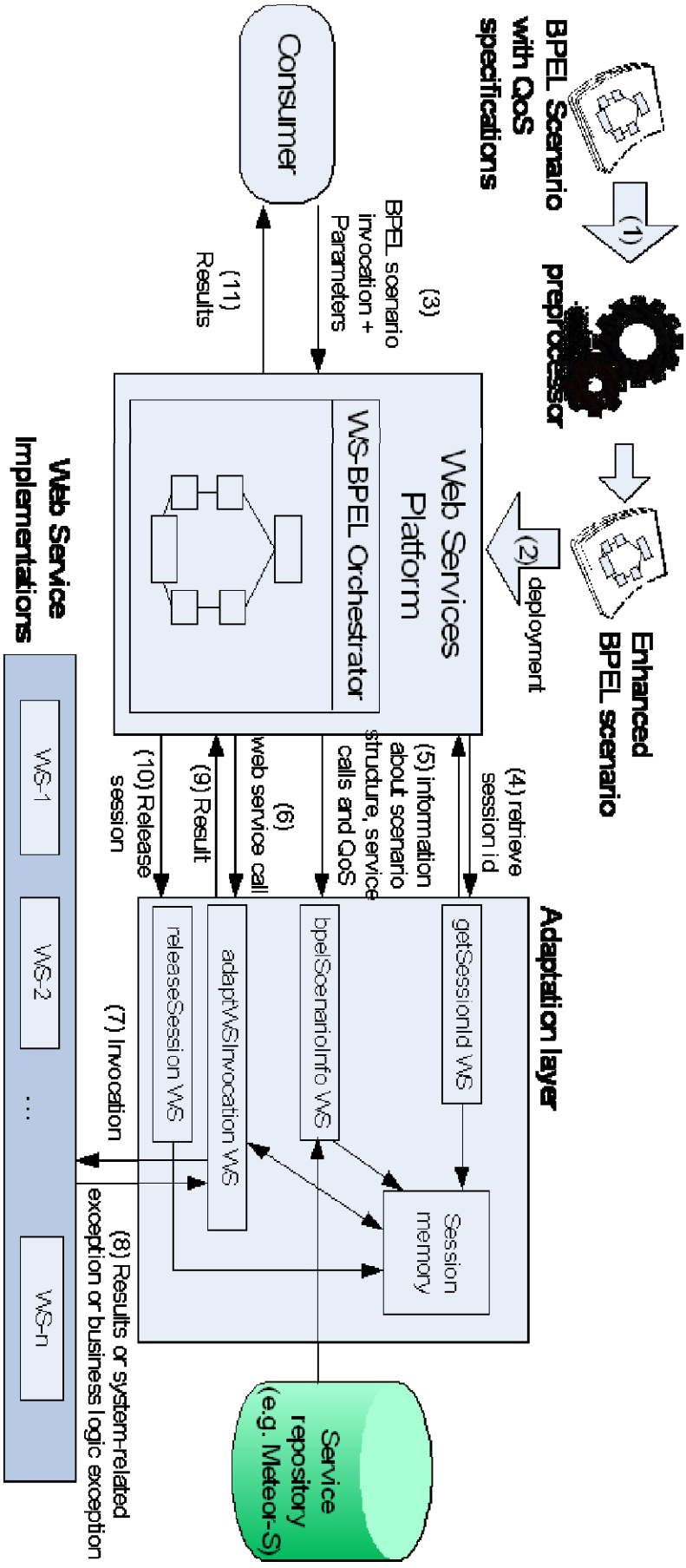


Figure 4: Proposed Framework Architecture

4.4 Preprocessing the BPEL scenario

As shown in figure 4, before the BPEL scenario is deployed on the web services platform, it is processed by the BPEL preprocessor, which creates an enhanced BPEL scenario as its output. The enhanced BPEL scenario differs from the original one in the following aspects:

1. it includes as its first operation an invocation to the web service getSessionId provided by the middleware; the result of the invocation is stored in a variable for later perusal.

```
<assign>
  <copy>
    <from><literal>respTime:5;cost:3</literal></from>
    <to variable="QoSmax_invoke1"/>
  </copy>
  <copy>
    <from><literal>respTime:-1;cost:-2;availability:1</literal></from>
    <to variable="QoSweight_invoke1"/>
  </copy>
</assign>
<invoke name="invoke1" partnerLink="lnk1" portType="port1" operation="op1" inputVariable="input1" outputVariable="output2"/>
```

Figure 5: QoS specification in the BPEL scenario

2. it includes an invocation to the bpelScenarioInfo web service provided by the middleware, through which the BPEL scenario transfers to the adaptation middleware (a) the current session identifier (b) the values of all QoS-related parameters (QoSmax_, QoSmin_ and QoSweight_) and (c) the information about the scenario structure. The latter effectively is represented as a simplified XML representation of the BPEL scenario including only the <sequence>, <flow> and <invoke> constructs of the original BPEL scenario; for the <invoke> constructs in particular, only the name, and operation attributes are transmitted, coupled with the service's endpoint

address, extracted from the relevant WSDL file. This invocation is inserted before the first <invoke> construct of the original WS-BPEL scenario, to ascertain that the adaptation-related information have been transferred to the adaptation layer before the first invocation is intercepted and adapted.

3. The preprocessor arranges that each web service invocation is complemented with a header including the session identifier for the current scenario execution (the value returned by the getSessionId WS) and the value of the name attribute of the particular invoke construct. While header manipulation not a standard WS-BPEL feature, most contemporary BPEL engines provide means to set request headers, e.g. [17][19].
4. The BPEL scenario includes as its final operation an invocation to the releaseSession web service provided by the middleware.

The enhanced BPEL scenario produced by the preprocessor is then deployed to the web services platform and made available for execution.

4.5 Executing the BPEL scenario

When the BPEL scenario starts executing, it will retrieve the session identifier from the adaptation layer and subsequently will transfer to the adaptation layer all information described in the previous subsection. The adaptation layer at this stage proceeds to the creation of the current session's execution plan as follows:

1. for each web service invocation within the WS-BPEL scenario, its equivalent services are retrieved from the service repository; note that the information retrieved from the service repository includes the values for the equivalent services' QoS attributes. Only equivalent services that satisfy the QoS thresholds specified in the respective invocations' QoSmax_ and QoSmin_ are retrieved. If, for some service in the initial WS-BPEL scenario, no candidates satisfying the thresholds are found, then the adaptation layer returns a QoS_PolicyFault to the web services platform. The WS-BPEL designer may intercept the fault using the standard WS-BPEL mechanisms (<catch> construct) and attempt to resolve it, e.g. by relaxing the constraints and restarting the scenario, or simply notify the requesting client of the error condition.

2. the adaptation layer formulates all candidate execution plans for the particular execution of the BPEL scenario. Assuming that the scenario contains N invocations $\{inv_1, inv_2, \ldots, inv_N\}$ and that for each invocation inv_j there is a set of equivalent services $EQ_j=\{s_{j,1}, s_{j,2}, \ldots, s_{j,k}\}$, the maximal set of candidate execution plans is $EQ_j \times EQ_2 \times \ldots \times EQ_N$. This set is however pruned by removing elements that violate the service selection affinity principle, i.e. if invocations inv_i and inv_j are directed to the same service provider (i.e. services for which the host part of their endpoint address is identical) in the original scenario, then all candidate execution plans in which replacements to inv_i and inv_j are not directed to the same service provider are removed from the candidate set. If the pruning step results in an empty candidate execution plan set, then a QoS_PolicyFault is returned to the web services platform.
3. for each execution plan within the candidate set formulated in step 2, an overall score is computed. The score computation procedure proceeds in a bottom-up fashion: initially, the score of individual invocations is computed using the formula $sc(inv_i) = rt_i * w_{rt,i} + c_i * w_{c,i} + av_i * w_{av,i}$, where rt_i, c_i and av_i are the QoS attribute values for the service replacing inv_i in the execution plan, while $w_{rt,i}$, $w_{c,i}$, $w_{av,i}$ are the weights specified in the QoS_weight_ variable for invocation inv_i. After all individual invocations' scores have been computed, the formulas in table 1 are used to compute the overall score of the execution plan. Finally, the execution plan with the highest score is selected, and the correspondences between the original invocations and the services used in the selected execution plan are stored in the session memory (cf. figure 1), coupled with the current session id. The correspondences are marked as unbound; this flag will be used for exception resolution (described below).

When an invoke construct is processed within the BPEL scenario, the outgoing request is redirected to the adaptWSInvocation web service provided by the adaptation layer; this can be accomplished either using a proxy setting in the web services platform or by using a transparent redirection router (both techniques are detailed in [4]). When the adaptWSInvocation intercepts a request, it processes it as follows:

1. it extracts from the request headers the session identifier and name of the invoke construct. Using these keys, it queries the session memory for the correspondence between the invoke construct and the actual service endpoint, selected in the execution plan formulation phase.
2. the request is forwarded to the endpoint retrieved in the previous step and the reply is received. If the reply is a normal response or a business logic-level fault (cf.

0), then the reply is forwarded back to the web services platform. Additionally, the host part of the endpoint to which the invocation was made (denoted as h_{inv} in the following) is extracted, and the session memory is updated setting the correspondence between invoke construct names and endpoint address to bound, for all invocations to endpoints offered by h_{inv}. This update will prevent the exception resolution process (described below) from breaking the service selection affinity.

3. if the reply received from the invocation is a system-level fault (e.g. “host unreachable” or “connection refused”; for a full discussion the interested reader is referred to [6]), then the adaptation layer will try to resolve the fault by invoking a service equivalent to the failed one. Note however that such a resolution is possible only if no prior successful invocation was made in the same session to a service offered by h_{inv}. This restriction is applied to maintain session affinity, since if a prior invocation was made to host h_{inv} and the current invocation is directed to another host to resolve the system fault, then the service selection affinity will be broken. Taking the above into account, the adaptation layer first queries the session memory to determine if the current invocation has been marked as bound (recall from step 2 above that this will be performed if any prior invocation to services offered by h_{inv} has concluded successfully). If it has been marked as bound, then the fault cannot be automatically resolved and is thus returned to the web services platform. If, however, the current invocation is marked as unbound, then the adaptation layer first locates in the current execution plan all services s_1. s_2, …, s_k offered by host h_{inv} and then retrieves from the repository all k-tuples (s'_1. s'_2, …, s'_k) such that:

   a. s_i is equivalent to s'_i, $\forall$ i=1, 2, …, k; this condition guarantees the functional equivalence of the initial execution plan to the candidate exception resolution plan.

   b. s'_i satisfies the QoS thresholds specified in the respective invocations’ QoSmax_ and QoSmin_ variables, $\forall$ i=1, 2, …, k; this condition guarantees that the candidate exception resolution plan adheres to the restrictions specified by the client.

   c. all services s'_i are offered by the same host, which must be different from h_{inv}; this condition guarantees that the candidate exception resolution plan maintains the service selection affinity and that the failed host will not be retried.

   Subsequently, for each k-tuple KT_j, an overall score is computed using the formula $sc(KT_j) = \sum_{i=1}^{k} rt_i * w_{rt,i} + c_i * w_{c,i} + av_i * w_{av,i}$ (rt_i, c_i and av_i denote the QoS attribute values

for service s'_i in KT_j and while $w_{rt,i}$, $w_{c,i}$, $w_{av,i}$ are the weights specified in the QoS_weight_ variable for the respective invocation). The k-tuple with the highest score is then chosen and all invocations in the execution plan to services offered by host h_{inv} are replaced by the corresponding invocations to services of the chosen k-tuple. Finally, the failed service invocation is restarted, being now directed to the newly chosen endpoint. If a system-level exception occurs at this point, the next-best k-tuple is selected, the execution plan is updated and the invocation is restarted again; this is be repeated until either a request succeeds or an administrator-defined limit is reached; in the latter case, the system-level exception is returned to the web services platform.

4.6 Experimental analysis

In order to assess the performance of our approach and validate this approach, a set of experiments, aiming to measure and quantify the overhead incurred due to the introduction of the middleware has being conducted. In these experiments (a) the overhead imposed by the use of the invocations to the getSessionId and bpelScenarioInfo web services (invoked once per execution of a WS-BPEL scenario) (b) the overhead imposed for each web service invocation within the BPEL scenario and (c) the overhead imposed when the exception resolution mechanism is activated are measured. The time taken by the preprocessor to transform the original WS-BPEL scenario into its enhanced form is not assessed, since preprocessing takes place in an offline fashion, not penalizing thus the production system performance. Finally, release session time has been found to be negligible and metrics are not presented here due to space limitations. Moreover, the “release session” invocation may be implemented as an asynchronous web service call, having thus minimal impact on the WS-BPEL scenario execution time.

For these experiments two machines were used: the first one (a workstation equipped with one Pentium 4@2.8GHz CPU and 512MB of RAM) hosted the preprocessor and the clients, while the second one (a workstation equipped with one Pentium i7@1.6 GHz and 4 GBytes of RAM) hosted the BPEL execution engine (a Glassfish application server [20], the middleware and the target web services. The repository was implemented as an HSQLDB server, which was hosted on the second workstation ([21]). The repository was populated with synthetic data with an overall size of 2.000 web services. The machines were connected through a 100Mbps local area network.

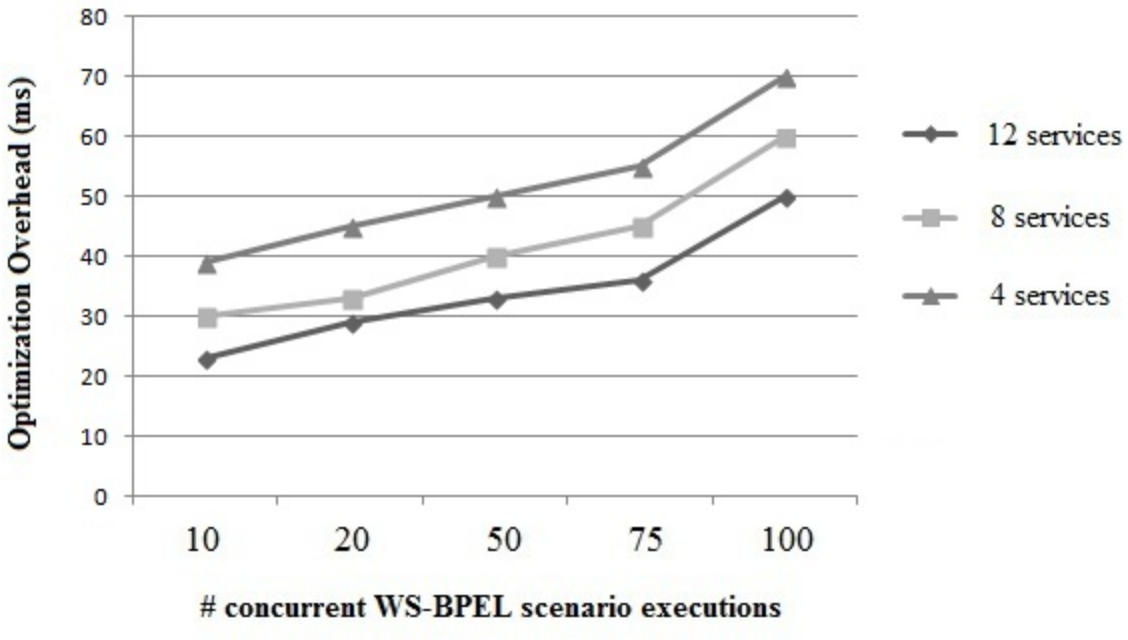


Figure 6: Optimization overhead

Figure 6 presents the optimization overhead (i.e. the overhead imposed by the use of the invocations to the getSessionId and bpelScenarioInfo web services) for varying number invocations present in the WS-BPEL scenario and different number of concurrent invocations (i.e. concurrent clients requesting the execution of the WS-BPEL scenario). The overhead increase has been found to be steeper when the number of concurrent invocations raises from 75 to 100 concurrent invocations; this is due to the depletion of the second workstation's resources at this load range; offloading specific tasks from that machine (e.g. hosting the adaptation layer and/or the target web services in a different machine than the WS-BPEL execution engine) is expected to provide smoother performance scaling.

Figure 7 presents the overhead incurred for the execution of a service invocation within the WS-BPEL scenario. This effectively accounts for (a) the two extra network messages required to transfer the request to the adaptation layer and return the reply from it and (b) the time taken to lookup in the session memory the correspondence between the particular service invocation and the endpoint determined in the optimization stage, and adjust the request message for forwarding to that endpoint. Even for high concurrency levels, the overhead for service execution is small (21 msec). Similarly to Figure 6, the overhead rises more steeply when the number of concurrent executions rises from 75 to 100, which is again due to the depletion of the second workstation's resources.

Figure 7: Service execution overhead

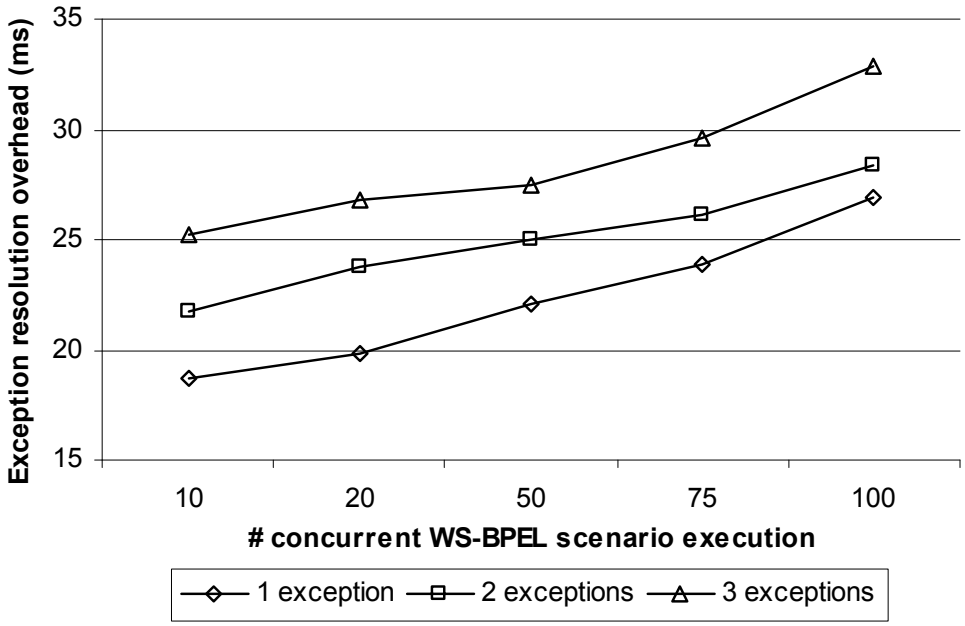


Figure 8: Exception resolution overhead

Finally, figure 8 presents the overhead incurred for resolving system level exceptions. This overhead accounts only for the time needed by the adaptation layer to perform the relevant tasks and does not include the time needed to invoke the failing services, since the latter varies significantly with the root cause of the failure (e.g. a fault owing to a network timeout leads to significantly higher delays than a fault owing to an invocation to a service that has been withdrawn), and therefore no meaningful statistics can be derived for the failing services' invocation times. Note also that the overheads illustrated

in figure 8 refer to the resolution of an exception occurring in the invocation of a single service; the “1 exception” data series refers to the case of the exception being resolved by the first alternative service, while the data series “2 exceptions” refers to the case that the first alternative service fails and the second one succeeds (similarly for the data series “3 exceptions”). As described above, only the first attempt to resolve an exception involves repository lookups and calculations of scores for alternative solutions, while subsequent attempts simply move to the “next best” solutions computed in the first attempt; this justifies the small time increments between the different data series in figure 8.

5. COLLABORATIVE FILTERING APPROACH

In this chapter an approach for integrating collaborative filtering techniques into the WS-BPEL execution adaptation procedure, introducing both an adaptation algorithm and an associated execution framework is presented. The adaptation algorithm uses both QoS specifications and semantic-based collaborative filtering personalization techniques to decide on which offered services best fit the client's profile, while the BPEL execution framework includes provisions for (a) specifying QoS requirements for invocations of web services within a WS-BPEL scenario (b) selecting exact services to be invoked and (c) adapting the WS-BPEL scenario executions according to the recommendations of the algorithm. This approach follows the horizontal adaptation paradigm, since horizontal adaptation preserves the execution flow which has been crafted by the designer to reflect particularities of the business process, while it also allows the exploitation of specialized exception handlers.

5.1 The service recommendation algorithm

As stated above, the adaptation algorithm chooses the services to be invoked in the context of the particular execution taking into account the following criteria:

i) The consumer's QoS specifications.

ii) Designations on which exact services should be invoked, if such bindings are requested by the consumer.

iii) The QoS characteristics of the available service implementations.

iv) The service subsumption relationships.

v) The usage patterns of services recorded in past WS-BPEL scenario executions.

Items (i) and (ii) are provided per WS-BPEL scenario execution by the consumer, while items (iii)-(v) are drawn from repositories maintained by the adaptation scheme. In the following paragraphs, we briefly describe the QoS concepts and we elaborate on the representation of the subsumption relationships and the usage patterns, as well as the overall adaptation algorithm.

5.2 The collaborating filtering-based algorithm

The collaborating filtering-based algorithm, upon each invocation of a WS-BPEL scenario, proposes sets of actual services that functionalities denoted as “to be recommended” could be bound to. Each proposal is tagged with a CF-score, which is determined through a collaborative filtering process. Figure 9 illustrates the steps of the CF-based algorithm in the form of an activity diagram and figure 10 describes the algorithm in pseudocode, while the following paragraphs provide details on the actions taken within each step.

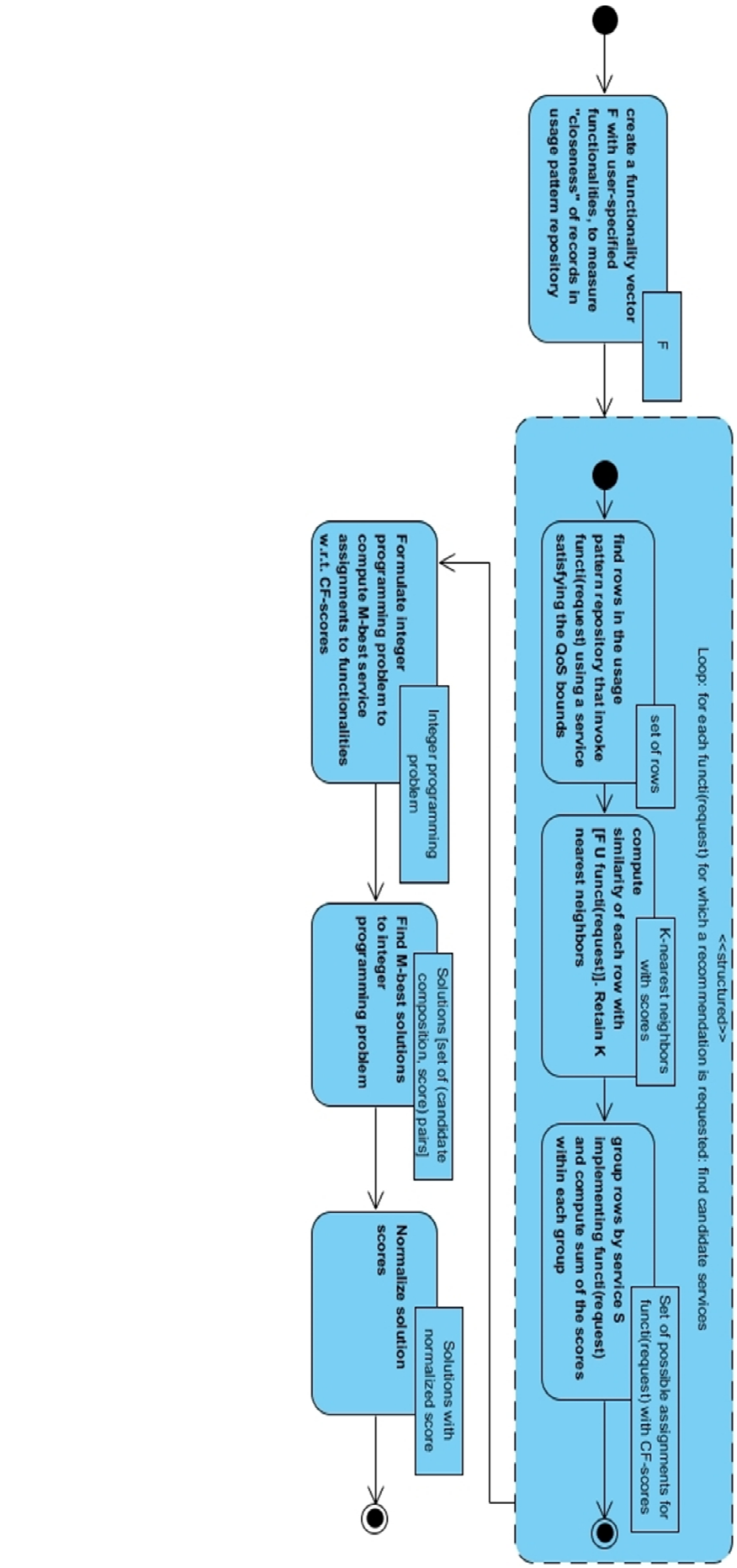


Figure 9: Activity diagram for the CF-based algorithm

/* CF-based adaptation algorithm pseudocode

Assumption:

- Scenario includes functionalities (i.e. invocations to services) f_1, f_2, …, f_n

Inputs:

- MIN and MAX (lower and upper bounds for QoS attributes)
- Specification of bindings of functionalities to concrete services B=(b_1, b_2, …, b_n) [b_i == null if no binding provided, else id of service)
- Specifications of functionalities not to be invoked O=(o_1, o_2, …, o_n) [o_i == thrue if f_i should not be invoked, false otherwise]
- Specifications of functionalities for which recommendations are requested R=(r_1, r_2, …, r_n) [r_i == category of functionality if a recommendation is requested for f_i, null otherwise]
- Subsumption relation tree, including the QoS attribute value of services

Outputs:

- List of <execution plan, score> pairs, with “execution plan” binding services that recommendation is asked for to concrete services

*/

```
/* 1. Build scenario-level functionality vector */
for (i = 1; i <= n; i++)
  if (B[i] != null) then
     F[i] = B[i]; /* set explicit binding information in the functionality vector */
  Else
     F[i] = null;
  end if
end for
/* 2. For each functionality that a recommendation is requested for, fetch matching scenario executions from the usage patterns repository */
for (i = 1; i <= n; i++)
  if (R[i] != null) /* recommendation requested */
```

```
    /* Fetch matching rows from the usage patterns repository */
    matchingRows = selectFromUsageRepository row: R[i] exact row[i] ∨ R[i] plugin row[i];
  /* Formulate FV[fi], to be used as the similarity yardstick for functionality fi */
    FV[fi] = F;
    FV[fi][i] = R[i]
/* 3, 4 drop rows not in bounds and compute score for remaining ones */
    for each (row in matchingRows)
      if (notInBounds(row, MIN, MAX) then
        removeElement(matchingRows, row);
      else
        scores[row] = computeSimilarity(row, FV[fi]);
    end for
/* 5. Retain k-nearest scores and compute group scores */
    qualifyingRows = top30_Scores(matchingRows, scores);
    /* formulate groupScores, containing the candidate services for delivering functionality fi and the respective scores */
    groupedScores[fi] = groupRowScores(qualifyingRows, scores, fi);
  end if /* recommendation requested */
end for
/* 6. formulate and solve integer programming problem, obtaining the top-20 execution plans */
ipp = formulateIntegerProgrammingProblem(groupedScores[fi]: R[i] != null)
CF_solutions = obtainTop20Solutions(ipp)
/* 7. Finally, normalize scores */
CF_proposal = normaliseScores(CF_solutions);
```

Figure 10: Pseudocode for the CF-based algorithm

1. it formulates a scenario-level functionality vector F=(f_1, f_2, …, f_n), where each f_i corresponds to a functionality that is part of the WS-BPEL scenario. The values of the elements f_i are determined as follows:
    - if functionality corresponding to element f_i is bound to a specific service, then the value of f_i is set to the identifier of this service.
    - in all other cases (i.e. if the corresponding functionality is not invoked in the context of the particular WS-BPEL scenario execution or a recommendation is requested for it), the value of f_i is set to null.

    Regarding this example, the functionality vector would be set to F=(null, GrandResort, NBA).
2. for each functionality $funct_i$(request) for which a recommendation is requested, the algorithm retrieves from the usage patterns repository the rows for which
    $funct_i$(request) exact $funct_i$(row) or $funct_i$(request) plugin $funct_i$(row)
    i.e. those patterns containing either an invocation to some identical functionality or a functionality to an invocation to a more specific one (figure 1). These are the only rows that are useful for formulating a recommendation for $funct_i$(request), since they involve services that deliver the requested functionality.
    Additionally, a request-level functionality vector F($funct_i$(request)) is formulated, by replacing the null value corresponding to $funct_i$ in vector F with the category corresponding to functionality $funct_i$. The new functionality vector will be used to calculate the similarity of the current request with the usage patterns in the repository, as explained below.
    In our example, rows 1, 2, 5, 6 and 7 of table 3 would be selected (rows 3 and 4 do not satisfy the subsumption relation criteria). The functionality being considered is $funct_i$=AirTravel and it corresponds to the first element of functionality vector F=(null, GrandResort, NBA) generated in step 1; hence, the functionality vector F(AirTravel) will be formulated with F(AirTravel)=(AirTravel, GrandResort, NBA).
3. the rows for which the QoS characteristics of service $funct_i$(row) do not satisfy the bounds set through vectors MIN($funct_i$) and MAX($funct_i$) are dropped; effectively, these rows cannot be used in the recommendation, since they involve services whose QoS characteristics do not satisfy the consumer's requirements.
    Regarding this example, row #1 would be dropped, since the OlympicAirways service implementing the functionality AirTravel does not satisfy the bounds related to the cost QoS attribute; therefore only rows 2, 5, 6 and 7 would be retained.

4. for each row retained by step 3, the algorithm computes a similarity score, indicating its similarity with the current request, as the latter is represented by the F(functᵢ(request)) functionality vector. The similarity score is calculated using the Sørensen similarity index [60] (alternatively known as Dice's coefficient [61]), according to which the similarity of two sets A={a_1, a_2, …, a_n}, B={b_1, b_2, …, b_m}, is equal to $S(A,B)=\frac{2|A\cap B|}{|A|+|B|}$, properly modified to suit a domain with semantic similarities. The modification follows the approach used in the fuzzy set similarity index calculation, where the cardinality of the intersection of two sets (i.e. the nominator in the Sørensen similarity index formula) is computed as the sum of the probabilities that a member belongs to both sets [62]. Correspondingly, when set member similarity is considered, the nominator of the fraction is replaced by $2*\sum_i sim(a_i,b_i)$, where sim(a_i, b_i) is a metric measuring the similarity between a_i and b_i; analogous approaches are adopted in ontology alignment and ontology matching domains, e.g. [31]. Effectively, the cardinality of the sets' intersection (i.e. the nominator of the fraction) used in cases that only the "equals" operator is available, is replaced by the sum of similarities of the corresponding elements. As a similarity metric between two functionalities (web services or categories), the one proposed in [27] is adopted:
   sim(s_1,s_2) = C–lw*PathLength – NumberOfDownDirection
   where:
   - C is a constant set to 8 [27][32].

lw is the level weight for each path in subsumption tree (cf. figure 1, chapter 3-2). The value of lw depends on the depth of the subsumption tree and the level of the node in it. To count the level weight lw, the formula $lw=\frac{subTreeDepth-(ln-1)}{subTreeDepth}$ [27] is used, where subTreeDepth is the depth of the subsumption tree, and ln is the level of node s_2 in the subsumption tree (the root node has a level equal to 1, its immediate children a level equal to 2 and so forth).

- PathLength is the number of edges counted from functionality s_1 to functionality s_2.
- NumberOfDownDirection is the number of edges counted in the directed path between functionality s_1 and s_2 and whose direction is towards a lower level in the subsumption tree.

This similarity metric dividing the result computed in the above formula by 8 is further normalized; this way, the similarity metric is always in the range [0, 1] and so is the value of the modified Sørensen similarity index, consistently with its original definition.

Regarding this example, the similarity measures between the F(AirTravel) functionality vector and the rows retained by the third step of the algorithm are:

sim(F(AirTravel), row2) = (19/24, 19/24, 10.5/24)

sim(F(AirTravel), row5) = (19/24, 14/24, 10.5/24)

sim(F(AirTravel), row6) = (19/24, 19/24, 0)

sim(F(AirTravel), row7) = (19/24, 14/24, 15/24)

and consequently the modified Sørensen similarity index values between F(AirTravel) and these rows are:

S(F(AirTravel), row2) = 2*(19/24+19/24+10.5/24) / 6 = 0.674

S(F(AirTravel), row5) = 2*(19/24 + 14/24 + 10.5/24) / 6 = 0.604

S(F(AirTravel), row6) = 2*(19/24+19/24+0) / 5 = 0.633

S(F(AirTravel), row7) = 2*(19/24+14/24+15/24) / 6 = 0.667

5. Finally, the algorithm retains only the K-nearest neighbors (i.e. the rows with the highest similarity scores), groups the retained rows by the value of the service implementing the $funct_i$(request) functionality and computes the sum of the scores within each group. Services implementing the $funct_i$(request) functionality that are associated with higher sums are deemed more suitable, in the sense of collaborative filtering, for being used in order to deliver the functionality $funct_i$(request). In this implementation, we have set K to 30. This value is higher than the commonly used value of 10 [33] (the value of 10 is also used in the original version of the algorithm [46]): the rationale behind choosing a higher value is to produce a longer list of services that are proposed by the collaborating filtering algorithm, providing thus step 6 that that follows with more options and allowing it to make a choice best suiting the user's request. Effectively, for functionality $funct_i$(request) this step will compute a set of possible concrete service assignments $CFPA(i)=\{s_{i,1}, s_{i,2}, \ldots, s_{i,L(i)}\}$, with the cardinality of $CFPA(i) = L(i) \leq 30$ (the number of proposed solutions may be less than 30, if less than 30 distinct services are recorded in the usage patterns repository to have been used for delivering $funct_i$(request)). For each possible concrete service assignment $s_{i,j}$, the respective similarity score $CFS(s_{i,j})$ has also been computed.

In this example, all rows would be retained, since only three rows exist. Since in row 2 and row 7 the same service (SwissAir) is used to deliver the AirTravel functionality, their individual scores would be summed up to formulate the solution SwissAir with a score of 1.341. The AirFrance service is the second solution with a score of 0.633 (only row 6 contributes to it) and the Lufthansa service is the third solution with a score of 0.604 (only row 5 contributes to it).

Steps 2-5 are repeated for each functionality $funct_i$(request) for which a recommendation is requested.

6. After the lists of candidates for each individual service that is subject to adaptation have been computed, the algorithm selects the top-20 execution plans with respect to their CF-score. The rationale behind choosing the value of 20 has been discussed in Section 3-1, above. Given an execution plan containing services ($s_{1,i}$, $s_{2,j}$, …, $s_{N,k}$) with the similarity scores of the services computed in step 5 being (CFS($s_{1,i}$), CFS($s_{2,j}$), …, CFS($s_{N,k}$)), then the CF-score of the execution plan is equal to CFS($s_{1,i}$)+CFS($s_{2,j}$)+…+CFS($s_{N,k}$). Using the sum function to aggregate the individual service scores into the execution plan CF-score ensures that the final result reflects the matching scores of all involved services. Contrary, the use of the max function would reflect the matching score of the best matching service only and therefore would render the algorithm prone to selecting an execution plan for which a single service has a very high score, but all other services have very poor ones.

   Producing the top-20 execution plans is modelled as a integer programming optimization problem, which is formulated as follows: maximize the overall utility value given by:

$$OUV_{CF} = \sum_{i=1}^{F} \sum_{j=1}^{L(i)} CFS(s_{i,j}) * x_{i,j}$$

   where F is the number of functionalities $funct_k$(request) requiring adaptation, and each $x_{i,j}$ is a binary variable taking the value 1 if $s_{i,j}$ is selected for delivering functionality $funct_i$(request), and the value 0, otherwise. Since each functionality $funct_i$(request) is delivered in the final execution plan by exactly one concrete service, the maximization of the utility value is subject to the constraint

$$\sum_{j=1}^{L(i)} x_{i,j} = 1, 1 \leq i \leq F$$

If service selection affinity needs to be maintained between functionalities $funct_i$(request) and $funct_j$(request), then an additional constraint is added to the problem, following the procedure described (step 2), above.

Subsequently, the integer programming problem is solved, and the k best solutions are obtained. Obtaining the k best solutions can be achieved by solving the problem once, and then adding a constraint rendering the obtained solution infeasible (for the IBM ILOG CPLEX optimizer [57] used in our experiments, this procedure is described in [58]). In this implementation, k was setequal to 20, as explained above.

In our example, adaptation has been requested for a single functionality, for which three possible solutions have been computed, therefore this step generates three solutions, as shown in Table 6.

Table 6: Utility function values and CF problem solutions for services in QPA(AirTravel)

CF problem solution	Selected $s_{AirTravel,i}$	$CFS(s_{AirTravel,i})$	OUV_{CF}
$x_{AirTravel,SwissAir}=1$, $x_{AirTravel,AirFrance}=0$, $x_{AirTravel,Lufthansa}=0$	Swiss Air	1.341	1.341
$x_{AirTravel,SwissAir}=0$, $x_{AirTravel,AirFrance}=1$, $x_{AirTravel,Lufthansa}=0$	Air France	0.633	0.633
$x_{AirTravel,SwissAir}=0$, $x_{AirTravel,AirFrance}=0$, $x_{AirTravel,Lufthansa}=1$	Lufthansa	0.604	0.604

7. The final step in this algorithm is to normalize the range of the solution's CF-scores, expressed through the respective values of the overall utility function, to the range [0, 1], in order for the combination step to function correctly. Normalization is again performed using the formula

$$normedCF\_Score_i = \frac{CF\_Score_i - minCF\_Score}{maxCF\_Score - minCF\_Score}$$

[37], where minCF_Score and maxCF_Score are the minimum and maximum scores, respectively, of the solutions produced in step 6. The output of the algorithm is the set of candidate execution plans, with each execution plan being tagged with the relevant normalized score (CF-score) (Table 7).

Table 7: Normalized solution scores

$s_{AirTravel,i}$	Normalized score
Swiss Air	1.000
Air France	0.039
Lufthansa	0.000

Note that the elimination of rows not satisfying the QoS bounds (step 3) is in this case merely an optimization step: indeed, these rows could be normally processed having their similarity scores computed, and would then be dropped in the combination step; however, performing the filtering at this stage saves unnecessary work (and thus improves performance), while it also facilitates the operation of the combination step, since it will not need to check whether the solutions proposed by each algorithm are conferment to the constraints set by the user.

5.3 The Execution Adaptation Architecture

The execution adaptation architecture adopts the middleware-based approach, typically used in QoS-based adaptation frameworks (e.g. [4][24][34]). An adaptation layer intervenes between the BPEL execution engine and the actual service providers, selecting the services that will actually be invoked in the context of any single WS-BPEL scenario execution using the algorithm described in section 5.2 and arranging for redirecting the actual invocations to the selected services. Redirection is performed through a specially crafted web service, namely adaptInvocation. The adaptation layer implements three utility web services, namely getSessionId (assigning unique session identifiers to individual WS-BPEL scenario executions), prepareAdatation (accepting QoS bounds and service binding information, and executing the algorithm of section III to select services for the functionalities that adaptation is requested for) and releaseSession (cleaning up the information regarding the WS-BPEL scenario execution, upon its termination).

Furthermore, the execution adaptation architecture includes a preprocessor module which transforms “ordinary” (i.e. non-adaptive) WS-BPEL scenarios to a form that their execution can be readily adapted by the middleware. The preprocessor module relieves the burden of having to redesign the BPEL scenarios, limiting the necessary changes to (a) allowing the user to specify the QoS requirements and the service bindings (including requests for recommendation) at the front-end application (e.g. a web form) and (b) preprocessing and redeploying the WS-BPEL scenarios. Using existing WS-BPEL scenario, where functionality invocations are specified by means of concrete services, allows for using standard WS-BPEL scenario editors, instead of specialized software that would be required if functionalities were specified by means of abstract tasks.

Figure 11 presents the overall execution adaptation architecture. In the following, we will elaborate on the operation of the preprocessor and the operation of the adaptation layer.

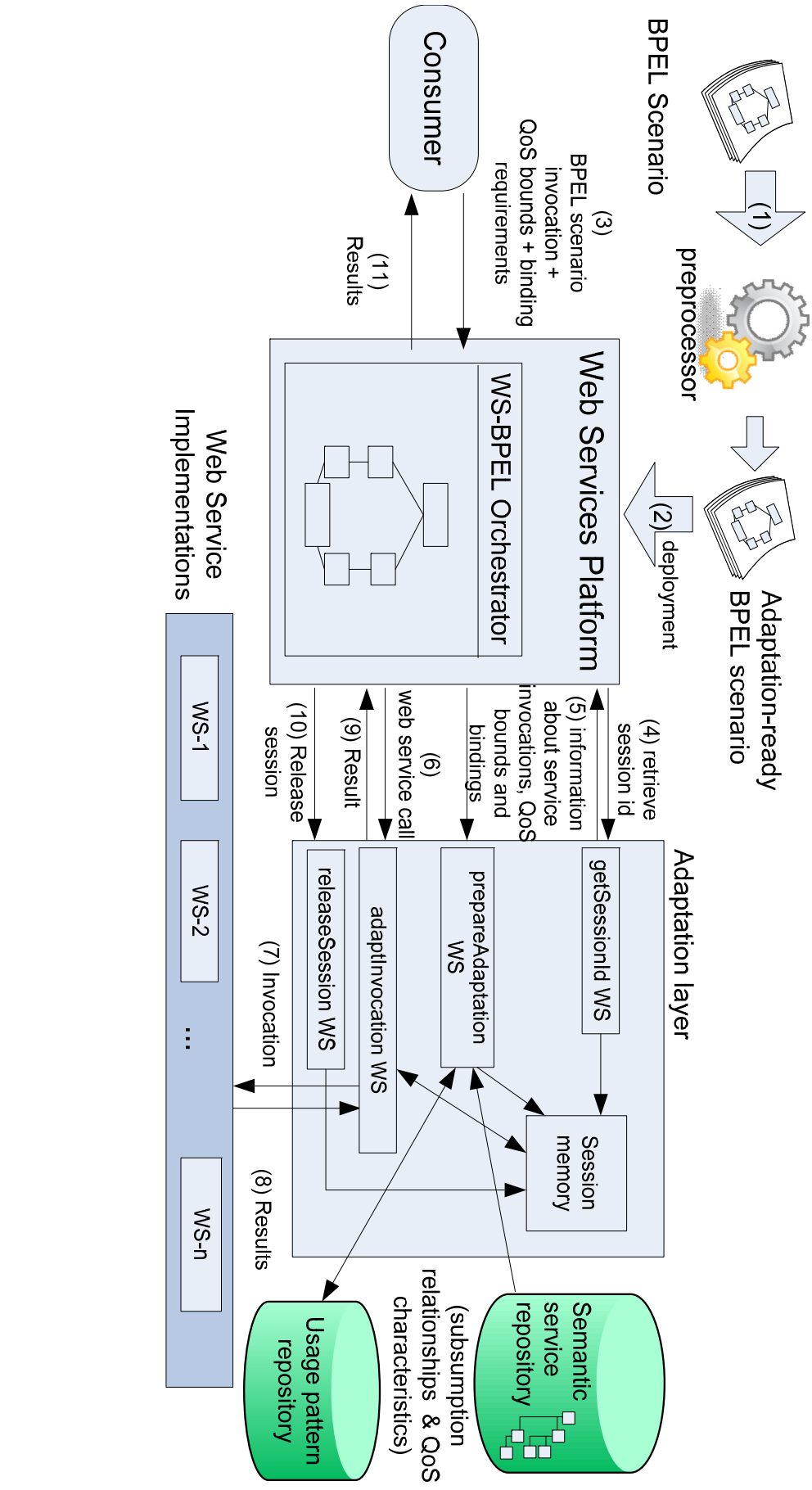


Figure 11: The Execution Adaptation Architecture

5.4 Preprocessing the BPEL scenario

As stated above, the preprocessor accepts as input a WS-BPEL scenario and transforms it into an “adaptation-ready” form. More specifically, the transformed scenario differs from the original one into the following respects:

1. it includes, as its first operation an invocation to the web service getSessionId provided by the middleware; the result of the invocation is stored in a variable and used in subsequent operations.
2. it extracts from the incoming request the information regarding (a) the QoS bounds for the functionalities and (b) the designations regarding which functionalities should be bound to specific services, which will not be invoked and for which a recommendation is requested. This information is then transmitted, together with the result of the getSessionId invocation, to the adaptation layer, through an invocation to the prepareAdaptation WS. For simplicity purposes, we will assume that the QoS bounds for a functionality funct used in the WS-BPEL scenario are represented using input parameters QOSMAXinvName and QOSMAXinvName, where invName is the value of the name attribute in the <invoke> construct realizing the functionality (<invoke name="invName" partnerLink="lnk1” …>). Similarly, the designations regarding functionality bindings are represented using input parameters BINDINGinvName, whose value may be one of (i) the id of the service to which the functionality should be bound, (ii) the literal SKIP if the functionality should not be invoked and (iii) the literal RECOMMEND, if a recommendation is requested for the specific functionality.

   Regarding the example in section 5.2, we will assume that the names of the invoke constructs corresponding to the ticket booking, hotel booking, and event attendance functionalities are bookTicket, bookHotel and attendEvent, respectively. Since the example invocation discussed in section III contained no QoS bounds, no QOSMIN* and QOSMAX* input parameters need to be set; the settings of the BINDING* variables will be BINDINGbookTicket=RECOMMEND, BINDINGbookHOTEL=GrandResort, BINDINGattendEvent=NBAFinals.

   The code realizing this functionality is injected immediately after the invocation to getSessionId.

3. each service invocation is redirected to the adaptInvocation, complemented with a header which includes the session id for the current WS-BPEL scenario execution (the value returned by the getSessionId WS) and the value of the name attribute of the particular invoke construct. Although header manipulation not a standard WS-BPEL feature, most contemporary WS-BPEL orchestration engines provide means to set request headers, e.g. [19][35]
4. an invocation to the releaseSession service of the adaptation layer is included as a final operation in the transformed scenario.

The adaptation-ready WS-BPEL scenario, as produced by the preprocessor, is deployed to the WS-BPEL orchestration engine and made available for execution.

5.5 Executing the BPEL scenario

When a WS-BPEL scenario commences execution, its first action will be to invoke the getSessionId web service hosted in the adaptation layer, so as to retrieve a unique session identifier. Afterwards, it invokes the prepareAdaptation web service of the adaptation layer, transmitting to it (i) the session identifier (b) the information regarding the QoS bounds for each functionality and (c) the functionality binding and the recommendation request information.

At this point, the adaptation layer has all the information needed to execute the algorithm described in Section 5.2, so as to produce any recommendations requested. After the algorithm has been applied, all service bindings (both those specified by the consumer and were provided as input to the prepareAdaptation web service, as well as those produced as output of the recommendation algorithm) are stored into the session memory, tagged with the session identifier. Effectively, the consumer session memory stores, for each session, the mappings between functionality invocations within the particular sessions and the concrete services that these invocations should be directed to. In the example presented in Section 5.2, the information that would be inserted into the session memory (assuming that the session identifier would be equal to s1) would be as shown in figure 12.

(Session: s1;
Bindings: (bookTicket: Swissair;
bookHotel: GrandResort;
attendEvent: NBAFinals))

Figure 12: Information inserted in session memory

The prepareAdaptation web service finally stores the service bindings it received as input parameters into the usage patterns repository, making thus the usage information available for future recommendation formulations.

When the WS-BPEL orchestration engine executes an invoke construct, the invocation is directed to the adaptInvocation service of the adaptation layer, due to the transformations made by the preprocessor (cf. item 3 in subsection IV.A, above). Upon reception of an incoming request, the adaptInvocation service proceeds as follows:

1. it extracts from the request headers the session identifier and name of the invoke construct.
2. Using the session identifier it retrieves from the session memory the service bindings pertinent to the particular session, and then it uses the name of the invoke construct to extract the binding of the specific functionality.
3. The request is then forwarded to the service indicated by the binding, the result is collected and finally it is returned to the WS-BPEL orchestration engine, as a reply to the original invocation.

Finally, when the WS-BPEL scenario reaches its end, it invokes the releaseSession web service, providing the session identifier as a parameter. The releaseSession service will then remove from the session memory all information pertaining to this session.

5.6 Performance Evaluation

In order to assess the performance of our approach and validate its feasibility, a set of experiments had being conducted, aiming to measure and quantify the overhead incurred due to the introduction of the middleware, the semantic trees and the BPEL scenarios stored in the database. The extent to which users are satisfied by the recommendations generated by the proposed algorithm is also of importance, and it is scheduled as part of future work.

In these experiments (a) the overhead imposed by the use of the invocations to the getSessionId, prepareAdaptation and releaseSession web services (invoked once per execution of a WS-BPEL scenario), (b) the overhead imposed for each web service invocation due to the intervention of the adaptInvocation service are measured. In the experiment the following parameters have been varied:

1. the number of concurrent invocations,
2. the size of the usage patterns repository,
3. the number of functionalities in the WS-BPEL scenario and
4. the number of recommendations requested per invocation.

The time needed by the preprocessor to transform the original WS-BPEL scenario into its “adaptation-ready” form has not been evaluated, because the preprocessor operates in an offline fashion, not imposing thus any overhead to the performance of the production system.

For these experiments two machines were used: the first one (a workstation equipped with one Intel Xeon E5-2620@2.0GHz / 6 cores CPU and 16 GB of RAM) hosted the preprocessor and the clients, while the second one (a workstation with identical configuration to the first, except for the memory which was 64GBytes) hosted the BPEL orchestration engine (a Glassfish application server [20] using the Metro web service stack [63]), the adaptation layer, the target web services, the service repository (which included the tree representation of the subsumption relations and the services’ QoS characteristics) and the usage patterns repository. The machines were connected through a 100Mbps local area network. Both repositories (the semantic service repository and the usage patterns repository) were implemented as in-memory hash-based structures, which proved more efficient that using a separate, in-memory database engine (e.g. HSQLDB [21] used in [45] and [46]). The new rows of the usage

patterns repository are written periodically to the disk to ensure persistence, while provisions have been made to allow for reloading the semantic service repository in case it changes (e.g. addition or deletion of services, or changes in the QoS values).

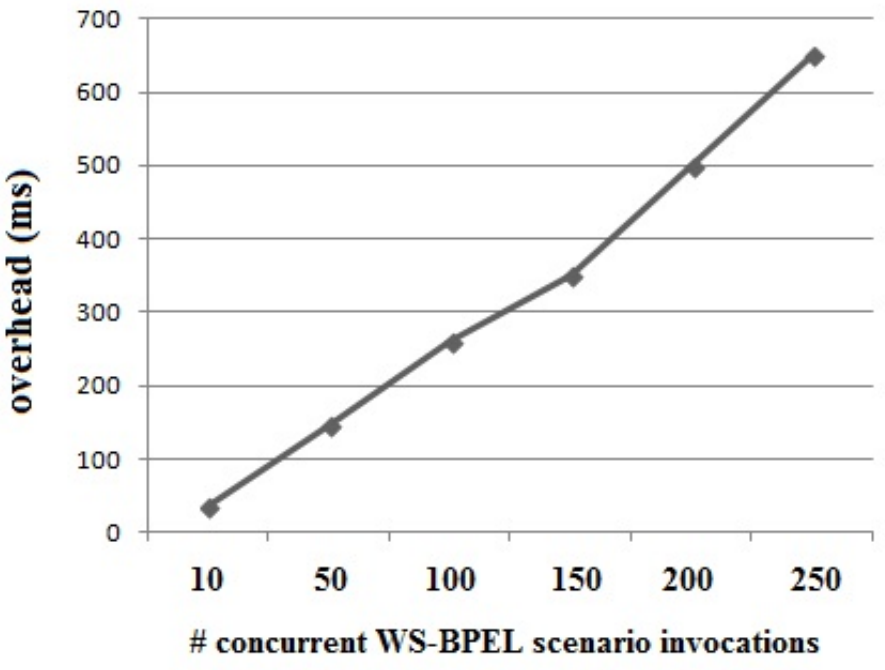


Figure 13: Recommendation and housekeping overhead for varying degrees of concurrency

Figure 13 presents the recommendation and housekeeping overhead, i.e. the overhead imposed by the use of the invocations to the getSessionId, adaptInvocation and releaseSession web services, for a varying degree of concurrent WS-BPEL scenario invocation requests arriving to the WS-BPEL orchestration engine. In this experiment, the size of the usage patterns repository was set to 1,000 qualifying entries (i.e. the usage patterns repository contained 1,000 entries matching the functionality for which a recommendation was requested; cf. step 2 in subsection 5-2), the number of functionalities in the WS-BPEL scenario was set to 10 and one recommendation was requested (i.e. 9 service bindings were fixed by the consumer). We can notice that the overhead is increasing linearly with the number of concurrent invocations, while we can also observe that even with 250 concurrent invocations the overhead is less than one second.

Figure 14 illustrates the time needed to perform recommendation and housekeeping tasks under a varying number of functionalities within the WS-BPEL scenario and for different counts of qualifying records in the usage patterns repository (250, 500 and 1000 records). In all cases, the concurrency level was set to one (a single request was submitted and processed) and one recommendation was requested. The recorded times

were found to increase by about 10%-12% when the number of functionalities increases by one; this is owing to the time needed for the extra processing to compute the semantic distances for the extra service.

In figure 14 we can also notice that the recommendation and housekeeping overhead is increasing linearly with the number of qualifying usage patterns repository entries. This was to be expected, since the current version of the algorithm considers and processes all entries fetched from the repository. The pruning of unpromising entries, e.g. entries whose cardinality is significantly greater than the cardinality of the functionality vector (increasing thus the denominator in the Sørensen's similarity index formula without a prospective increase in the nominator, hence leading to a poorer score) could contribute towards achieving better scalability with respect to the number of qualifying usage patterns repository entries.

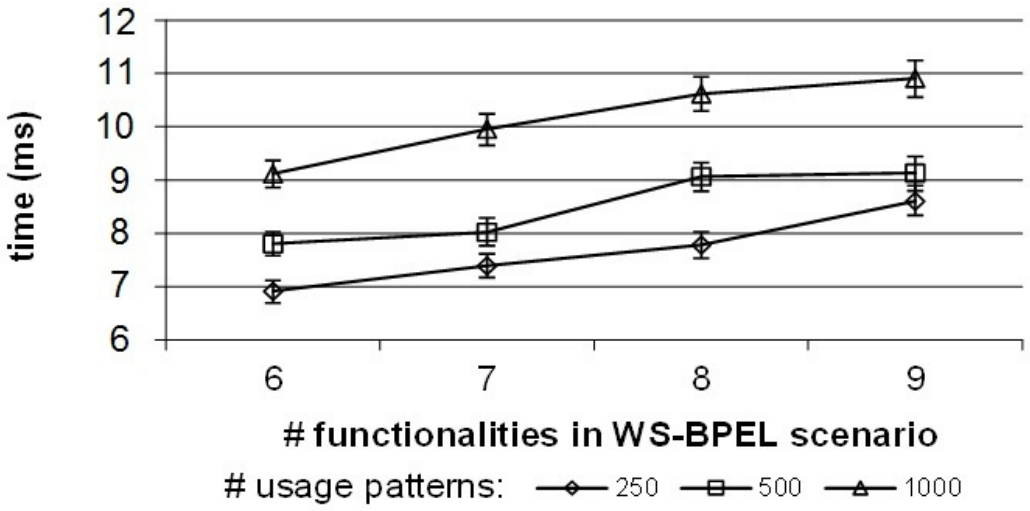


Figure 14: Recommendation and housekeping overhead for varying number of functionalities within the WS-BPEL scenario and qualifying usage patterns

Additionally, maintaining pre-computed similarity metrics between all pairs of functionalities in the subsumption relationship tree, would help achieve better absolute times, trading off time for space, albeit it would not contribute towards improving scalability.

Figure 15 illustrates the time needed to generate recommendations, with respect to the number of qualifying records in the usage patterns repository and the number of recommendations requested in a single WS-BPEL scenario invocation. In all cases, the concurrency level was set to one (a single request was submitted and processed) and

the WS-BPEL scenario whose execution was requested contained five functionalities. We can observe that the time needed for each recommendation is fairly stable, e.g. the time needed for making two recommendations is approximately double the time needed for making one recommendation, and similarly for making three recommendations. This is to be expected, since each recommendation is made individually, by repeating the same steps of the algorithm. The behavior regarding the number of qualifying usage patterns is analogous to the one observed in figure 14.

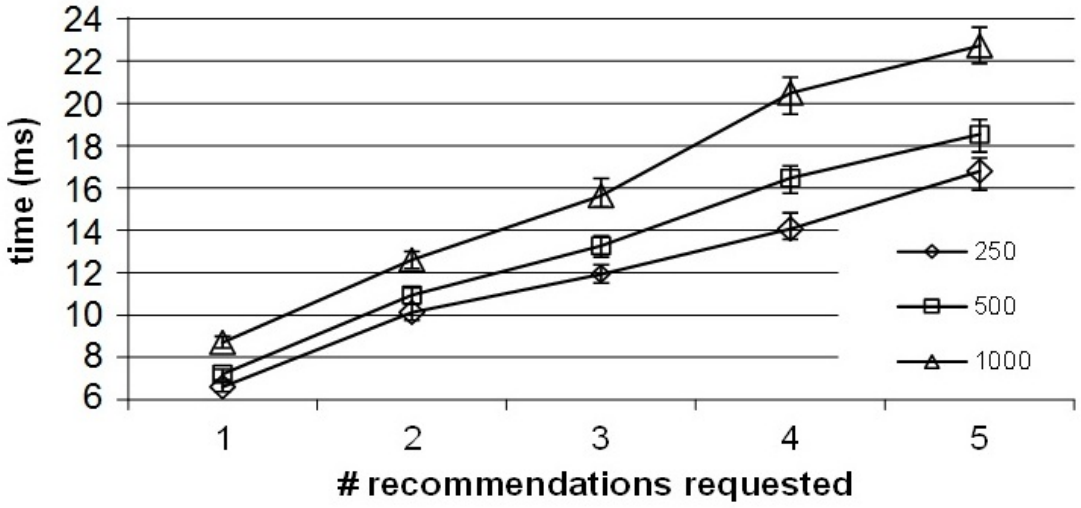


Figure 15: Recommendation and housekeping overhead for varying number of qualifying usage patterns and number of requested recommendations

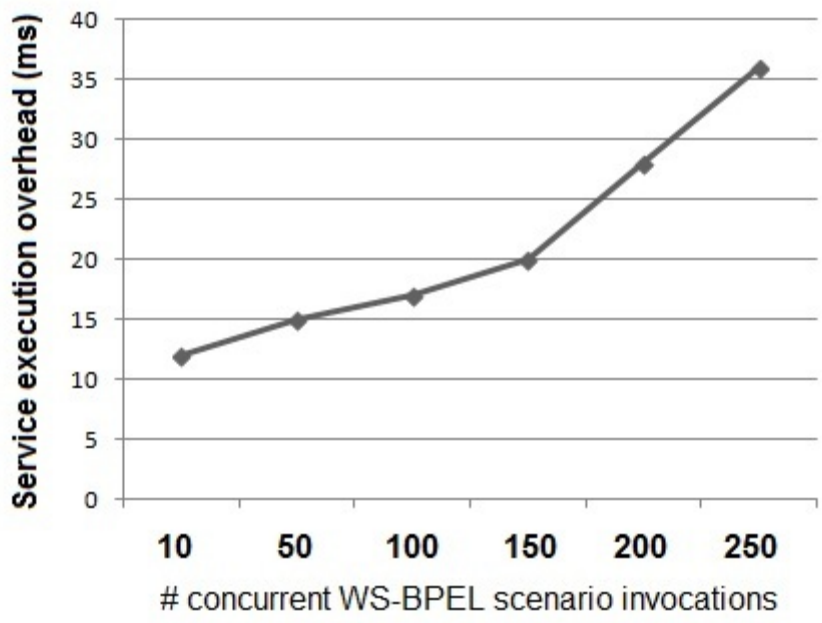


Figure 16: Service execution overhead for varying degrees of concurrency

Finally, figure 16 depicts the overhead imposed for each service execution. This overhead corresponds to the two extra network messages required per service invocation (four network messages are needed when the adaptation layer intervenes as opposed to two messages only when services are directly invoked), plus the time needed for the adaptation layer to access the session memory, retrieve the service mappings and redirect the request appropriately. We can notice that the overhead is increasing almost linearly with the number of concurrent invocations. The overhead is small, therefore once the prepareAdaptation service has concluded, the WS-BPEL scenario's performance is only minimally affected.

6. HYBRID APPROACH

In this chapter, a framework which incorporates runtime adaptation for BPEL scenarios is presented. The adaptation is based on (a) the quality of service parameters of available services, allowing for tailoring their execution to the diverse needs of individual users and (b) on collaborative filtering techniques, allowing clients to further refine the adaptation process by considering service selections made by other clients, in the context of the same business processes. The proposed framework also caters maintaining the transactional semantics that invocations to multiple services offered by the same provider may bear.

6.1 The service recommendation algorithm

As stated in sections 4.1 and 5.1, this approach follows the horizontal adaptation algorithm, i.e. it leaves the composition logic intact and adapts the execution by selecting which concrete service implementation will be used in each specific invocation. In order to perform this task, the algorithm takes into account the following criteria:

i) The consumer's QoS specifications (bounds and weights).
ii) Designations on which exact services should be invoked, if such bindings are requested by the consumer.
iii) Designations on which functionalities should not be invoked (e.g. in the example within section 4-1, the user does not want to attend any event).
iv) The QoS characteristics of the available service implementations.
v) The service subsumption relations.
vi) The usage patterns of services recorded in past WS-BPEL scenario executions.
vii) Statistics on how well-populated the usage patterns repository is; these are used to adjust the weights assigned to the results of the collaborative filtering-based adaptation, considering the fact that a poorly populated usage patterns repository is bound to produce results of poorer quality. To this end, the concept of user-item sparsity [44] is exploited; details on this aspect are given below.

Items (i)-(iii) are provided per WS-BPEL scenario execution by the consumer, while items (iv)-(vii) are drawn from repositories maintained by the adaptation scheme. Note that it is not necessary for the end-user to directly specify the QoS bounds or weights for each individual service as an input parameter to the scenario: the user could specify the bounds and weights in a more generic form (e.g. choose=cheapest), which would then be mapped to specific bounds for the involved services' QoS attributes through code provided by either the WS-BPEL scenario designer or an intermediate entity (e.g. the code behind a web page collecting the necessary data from the user, in order to invoke the BPEL scenario).

Recall also that the approach adopted in this chapter incorporates two different candidate service ranking algorithms, the first examining the QoS aspects only and the second being based on collaborative filtering techniques. The algorithms run in parallel to formulate their suggestions regarding the services that should be used in the adapted execution, and subsequently their suggestions are combined, through a metasearch score combination algorithm with varying weights. In the following subsections, the two algorithms (the QoS-based and the collaborating filtering-based) as well as the combination step will be described. Note that, for the metasearch score combination algorithm to work properly, the scores produced by each individual algorithm should be normalized [10][11][37]. To this end, the individual algorithms adopted from [45] and [46] have been extended to include a score normalization step; additionally the algorithms have been modified to allow them to function as "voters" in the metasearch process adopted in this chapter. Figure 17 depicts the operation of the algorithm in the form of an activity diagram.

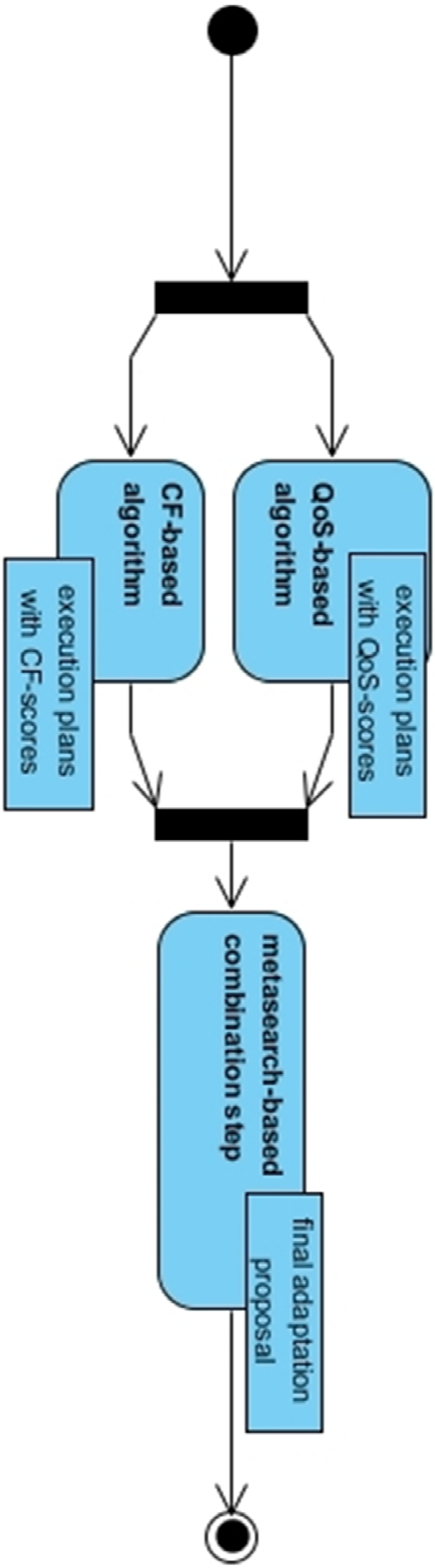


Figure 17: Activity diagram for overall algorithm operation

To illustrate the operation of the algorithms, an example execution request will be used:

TravelPlanner(bindings=(AirTravel(R), GrandResort, NBA), QoSLimits={$MIN_{AirTravel}$(rt: 4, cost: 3, av: 3), $MAX_{AirTravel}$(rt: 7, cost: null, av: null)}, QoSWeights={rt: 0.2, cost: 0.5, av: 0.3})

which can be effectively read as “I want to stay in Grand Resort, I want to watch an NBA match and I want a recommendation for an air travel; response time for the AirTravel service should be at least 4 and at most 7, cost should be at least 3 (recall that QoS attribute values are expressed in a “larger values are better” encoding scheme, therefore setting a lower bound for the cost excludes the most expensive services). It is assumed that the values of the QoS attributes for the services implementing the AirTravel functionality are as shown in table 8.

Table 8: QoS attribute values for services

AirTravel Company	Response_Time	Cost	Availability
Swiss Air	5	7	8
Olympic Airways	4	1	6
Air France	5	8	9
Lufthansa	6	3	4

6.2 The combination step

The role of the combination step is to synthesize the results given by individual algorithms to produce a single result. Recall from the previous two subsections that each algorithm produces a set of candidate execution plans, with each execution plan being tagged with the relevant normalized score (QoS-score or CF-score). Figure 18 presents the steps taken into the combination step in the form of an activity diagram, while the following paragraphs provide details on the actions taken within each individual step.

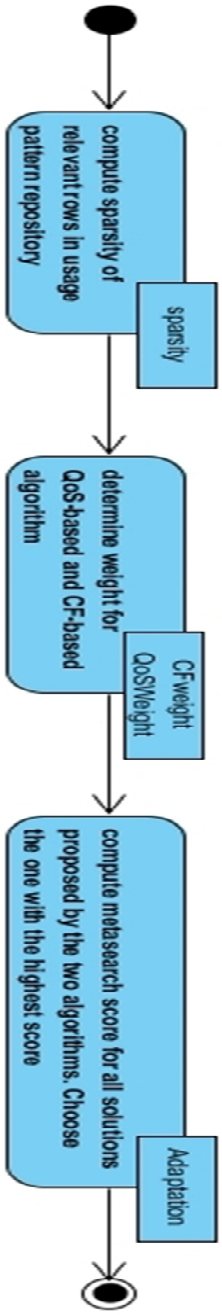


Figure 18: Activity diagram for the combination step

In order to combine the scores, one could use any standard metasearch score combination algorithms, such as CombMIN (minimum of individual scores), CombSUM (sum of individual results), CombMNZ (CombSUM, multiplied by the number of individual algorithms actually suggesting a particular solution) and so forth, with CombMNZ being identified as having the best performance [11]; He and Wu [37] however suggest that CombSUM and CombMNZ are best suitable for combining results from individual algorithms with relative similar performance. He and Wu [37] argue that if the individual algorithms have diverse performance, it is best to specify a certain belief about the quality of the recommendations from individual algorithms, proposing thus WCombSUM and WCombMNZ, which extend the standard CombSUM and CombMNZ schemes by introducing a weight for the individual ratings produced by the algorithms. More specifically, $\mathrm{WCombSUM_i} = \sum_{j=1}^{m_i} w_j * \mathrm{NormalizedScore_{i,j}}$

where $\mathrm{WCombSUM_i}$ is the final score for a particular result i, w_j is a predefined weight associated with the proposing algorithm j, m_i is the number of nonzero scores of result i (i.e. number of algorithms proposing the particular result), and $\mathrm{NormalizedScore_{i,j}}$ is the normalized score for result i produced by algorithm j. Also,

$$\mathrm{WCombMNZ_i} = \mathrm{WCombSUM_i} * m_i$$

[44] assert that collaborating filtering is prone to producing results with low prediction accuracy when the sparsity (i.e. the ratio of empty cells to the total number of cells) of the rating matrix exceeds 99.5%. In more detail, [44] demonstrates that in the sparsity range [99.5%, 99.9%] the mean absolute rate of collaborative filtering increases sharply (and hence the recommendation quality drops) in a linear fashion, while beyond the sparsity limit of 99.9% there is no point in taking into account the results of collaborative filtering-based algorithms. For the sparsity range [0, 99.5%], figure 19, based on [44] shows that collaborating filtering has inferior performance as compared to content-based collaborating filtering, exhibiting a mean absolute error rate higher by 6%-9%, with an average of 7.8%.

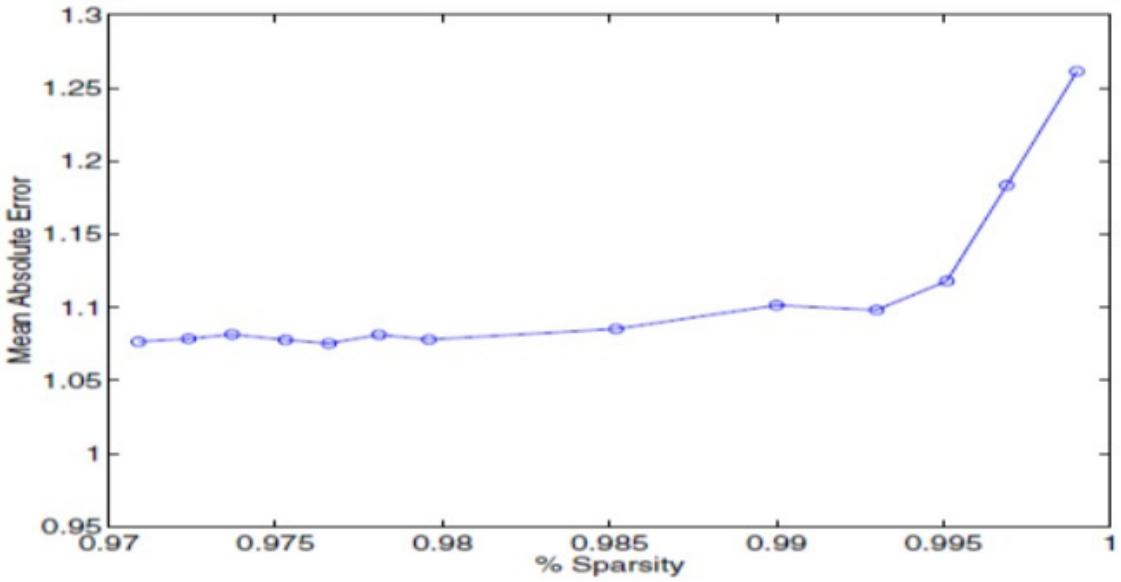


Figure 19: Mean absolute error for varying number of the recommenders' table sparsity

Taking the above into account, we adopt a modified version of the WCombMNZ metasearch score combination algorithm, in which the weight assigned to each of the algorithms (the QoS-based one and the collaborating filtering-based one) varies according to the sparsity of the rating matrix (i.e. the usage patterns repository). More specifically, the weights are calculated as follows:

$$CFweight = \begin{cases} 0.40, \text{if sparsity} \le 0.995 \\ \dfrac{0.40 * (0.999 - \text{sparsity})}{0.004}, \text{if } 0.995 < \text{sparsity} \le 0.999 \\ 0, \text{if sparsity} > 0.999 \end{cases}$$

$$QoSweight = 1 - CFweight$$

Figure 20: QoS and CF weights

Note here that the sparsity metric is calculated against the rows of the usage patterns repository that are retained by step (3) of the CF-based algorithm described in Section 5-2 therefore the weights are individualized for each distinct adaptation. For efficiency purposes, the calculation of the sparsity is performed during the execution of the CF-based algorithm described in Section 5-2, where the pertinent data are readily available, and is forwarded to the combination step, along with the list of the execution plans.

In this weight assignment setting, the QoS-based algorithm is considered as a counterpart of the content-based collaborating filtering algorithm examined in [44], under the analogy that the QoS-based algorithm examines the values of item metadata (QoS attribute value of services), instead of the item contents. The value of 0.40 for

CFweight in the range [0, 0.995] has been selected due to the fact that collaborating filtering exhibits higher mean absolute error rate as compared to the content-based collaborating filtering algorithm, and hence its weight should be set lower than 0.5, according to the rationale of the WCombMNZ metasearch score combination algorithm. On the other hand, the value of 0.4 is large enough to allow the collaborative filtering algorithm to influence the final result. The appropriateness of value 0.4 has been also experimentally verified, as described above.

After computing the WCombMNZ metasearch for all candidate execution plans, the combination step selects the execution plan with the highest score, which will be used to drive the adaptation process. If more than one candidate execution plans have the same score, one of them is selected randomly.

In this example, the sparsity of the table consisting of rows 2, 5, 6 and 7 is equal to 0.891. This stems from the facts that (a) each row in the usage patterns repository has 23 columns, i.e. equal to the number of concrete services in the subsumption tree, (b) rows 2 and 7 have 3 cells equal to “1” and (c) rows 5 and 6 have 2 cells equal to “1”, making thus a total of 10 cells having a value of “1” out of a total of 92, or, inversely, 82/92 cells are empty, leading to a sparsity of 0.891. Using the rules for computing CFweight and QoSweight given above, we get CFweight=0.40 and QoSweight = 0.6.

Using these weights, we may now compute the WCombMNZ value for each solution, as shown in table 9.

Table 9: Computing the WCombMNZ score for the solutions

Solution	QoS-based algorithm score	CF-based algorithm score	WCombMNZ formula	WCombMNZ score
Swiss Air	0.764	1.000	2 * (0.6 * 0.764 + 0.4 * 1.000)	1.717
Air France	1.000	0.039	2 * (0.6 * 1.000 + 0.4 * 0.039)	1.231
Lufthansa	0.000	0.000	2 * (0.6 * 0.000 + 0.5 * 0.000)	0

6.3 The execution adaptation architecture

The execution adaptation architecture follows the middleware-based approach, which is common in QoS-based adaptation frameworks (e.g. [4][24][34]). In this approach, an adaptation layer intervenes between the BPEL execution engine and the actual service providers, selecting the services that will actually be invoked in the context of any single WS-BPEL scenario execution, using the algorithm described in the previous section and arranging for redirecting the actual invocations to the selected services. Redirection is performed through a specially crafted web service, namely adaptInvocation. The adaptation layer implements three additional utility web services, namely getSessionId (assigning unique session identifiers to individual WS-BPEL scenario executions), prepareAdaptation (accepting information about the invocations that should be adapted, QoS bounds and service binding information, and executing the algorithm of section 3 to determine the execution plan that should be followed in the particular WS-BPEL scenario execution) and releaseSession (cleaning up the information regarding the WS-BPEL scenario execution, upon its termination). Furthermore, the execution adaptation architecture includes a preprocessor module which transforms “ordinary” (i.e. nonadaptive) WS-BPEL scenarios to a form that their execution can be readily adapted by the middleware. The preprocessor module relieves the burden of having to redesign the BPEL scenarios, limiting the necessary changes to (a) allowing the user to specify the QoS requirements and the service bindings (including requests for recommendation) at the front-end application (e.g. a web form) and (b) preprocessing and redeploying the WS-BPEL scenarios. Using existing WS-BPEL scenario, where functionality invocations are specified by means of concrete services, allows for using standard WS-BPEL scenario editors, instead of specialized software that would be required if functionalities were specified by means of abstract tasks.

Figure 21 presents the overall execution adaptation architecture. In the following, the operation of the preprocessor and the operation of the adaptation layer will be elaborated.

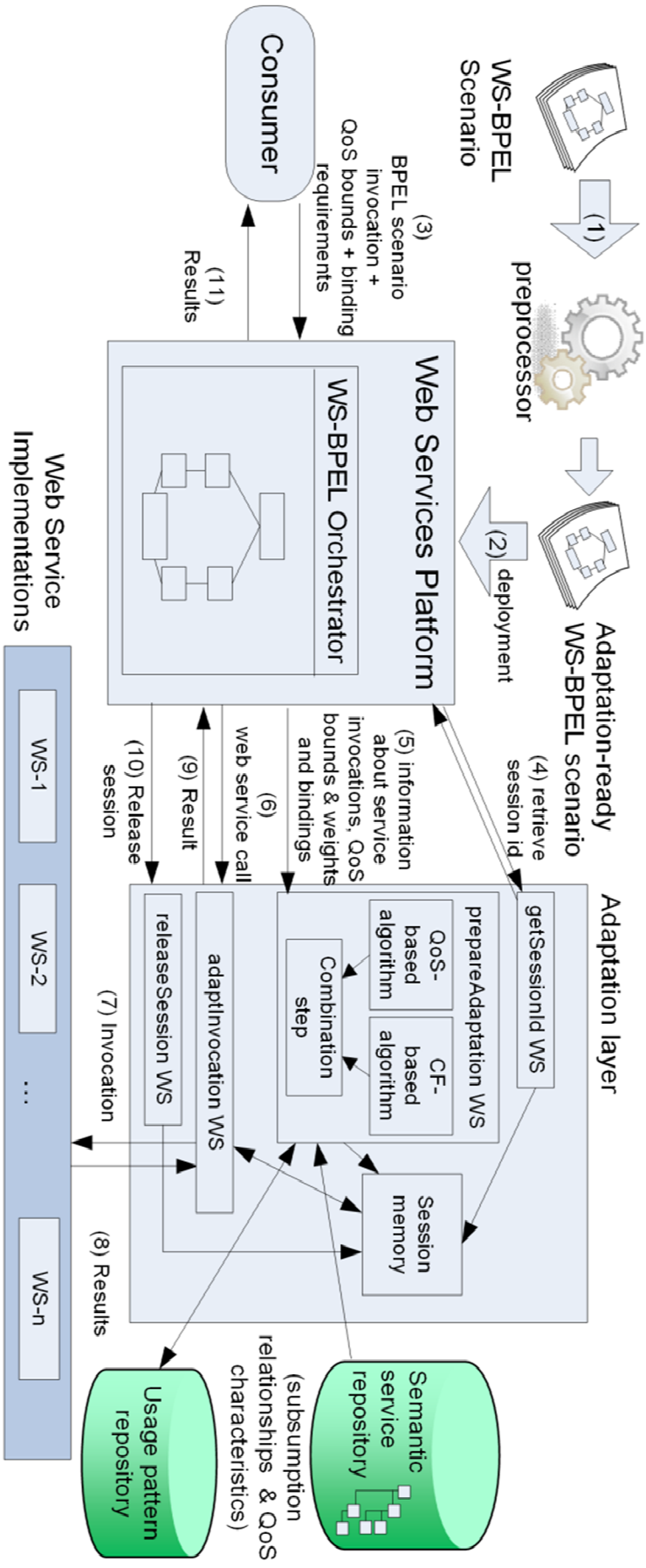


Figure 21: The Execution Adaptation Architecture

6.4 Specifying the QoS information in the scenario

The first step towards enabling the QoS-based adaptation is the specification of the required QoS bounds for service invocations and the specification of the weights of each QoS attribute. In order to provide this feature in a WS-BPEL compliant fashion, the proposed framework adopts the following conventions:

1. the designer should use the WS-BPEL variable QoS_weights to designate the weight vector W, i.e. the vector designating the weights of all QoS attributes (cf. Section 3.1). The BPEL designer may set the value of these variables after inspecting input parameters to the scenario (e.g. “choose=cheapest”), arranging thus for tailoring the QoS specification to the invoking user’s preferences.
2. the designer should include in each invoke construct in the WS-BPEL scenario the optional attribute name [1], assigning distinct names to the invoke constructs.
3. for the invoke construct having name invX, the designer should use the WS-BPEL variables QoSmax_invX and QoSmin_invX, which define the respective QoS specifications for the particular invocation. Similarly to the QoS_weights variable, the designer may set the values of the QoSmax_invX and QoSmin_invX variables after inspecting input parameters to the scenario.

Note that the setting above allows the designer to set different QoS bounds for each distinct invocation. This provides more flexibility with the adaptation of the scenario, e.g. in the travel planning scenario considered in Section 3.2, the bounds for the event attendance functionality may be set to high values (e.g. min=0.8, max=1), while the bounds for ticket booking and hotel booking may be set to low values, to indicate that the client wants good seats for the event to be attended, but economy travel tickets and an inexpensive hotel to limit the overall cost. On the other hand, the weights apply to the whole composition, rather than to individual services, since they reflect the client’s perceived importance of each QoS attribute dimension on the process as a whole, and not its constituent parts [23].

Listing 4-2 presents an excerpt of a BPEL scenario setting QoS specifications for an invocation (named invoke1). Since variable QoSmin_bookTicket does not include a setting for respTime and cost, no lower bounds will be considered for these attribute values in the process of adapting the service invocation bookTicket. Similarly, since the QoSmax_bookTicket does not include a setting for the availability QoS attribute, no

upper bound for this attribute value will be considered in the process of adapting the service invocation bookTicket.

```
<!-- assign global weights -->
<assign>
  <copy>
    <from><literal>respTime: 0.3; cost:-0.5; availability: 0.2</literal></from>
    <to variable="QoSweights"/>
  </copy>
</assign>
<!-- assign invocation-specific bounds for invocation “bookTicket”-->
<assign>
  <copy>
    <from><literal>respTime:0.5;cost:0.3</literal></from>
    <to variable="QoSmax_bookTicket"/>
  </copy>
  <copy>
    <from><literal> availability: 0.6</literal></from>
    <to variable="QoSmin_bookTicket"/>
  </copy>
</assign>
<invoke name="bookTicket" partnerLink="lnk1" portType="port1" operation="op1"
   inputVariable="input1" outputVariable="output2"/>
```

Listing 4-2 QoS specification in the BPEL scenario

6.5 Preprocessing the WS-BPEL scenario

As stated above, the preprocessor accepts as input a WS-BPEL scenario and transforms it into an "adaptation-ready" form. More specifically, the transformed scenario differs from the original one into the following respects:

1. it includes, as its first operation an invocation to the web service getSessionId provided by the middleware; the result of the invocation is stored in a variable and used in subsequent operations.
2. it collects the information regarding (a) the QoS bounds for the functionalities, (b) the QoS attribute weights and (c) which functionalities should be bound to specific services, which will not be invoked and for which a recommendation is requested. This information is then transmitted, together with the result of the getSessionId invocation, to the adaptation layer, through an invocation to the prepareAdaptation WS.

   As stated in Section 4.1, the QoS bounds for a functionality used in the WS-BPEL scenario are represented using input parameters QoSmax_invName and QOSmin_invName, where invName is the value of the name attribute in the <invoke> construct realizing the functionality (<invoke name="invName" partnerLink="lnk1" …>), while the QoS attribute weights are stored in variable QoSweights. Similarly, the designations regarding functionality bindings are represented using variables BINDING_invName, which will again be set by the designer, probably after examining some input parameters. The value of a BINDING_invName variable may be one of (i) the id of the service to which the functionality should be bound, (ii) the literal SKIP if the functionality should not be invoked and (iii) the literal RECOMMEND, if a recommendation is requested for the specific functionality.

   Regarding the example in section 5.2 (where a user wants to stay in Grand Resort and attend the NBA Finals, and requests a recommendation for ticket booking), we will assume that the names of the invoke constructs corresponding to the ticket booking, hotel booking, and event attendance functionalities are bookTicket, bookHotel and attendEvent, respectively. If the QoS attribute weights and QoS bounds for the bookTicket service were as shown in Listing 4-2, the information transmitted to the middleware through the prepareAdaptation WS would be as follows:

SessionId=<as returned by getSessionId>
QoSweights=respTime: 0.3; cost:-0.5; availability: 0.2
QoSmax_bookTicket=respTime:0.5;cost:0.3
QoSmin_bookTicket=availability:0.6
BINDING_bookTicket=RECOMMEND
BINDING_bookHOTEL=GrandResort
BINDING_attendEvent=NBAFinals

This information, together with the data contained in the semantic service repository and the usage patterns repository are adequate for the middleware layer to execute the algorithm described in section 5 and determine the execution plan to be used. The implementation arranges for executing the QoS-based and the CF-based parts in parallel, so that the optimization step can benefit from the presence of multiple processors and/or cores (a commodity in contemporary hardware).

The invocation to the prepareAdaptation WS is inserted before the first <invoke> construct of the original WS-BPEL scenario, to ascertain that the adaptation-related information have been transferred to the adaptation layer (and processed by it) before the first invocation is intercepted and adapted.

3. each service invocation is redirected to the adaptInvocation service, complemented with a header which includes the session id for the current WS-BPEL scenario execution (the value returned by the getSessionId WS) and the value of the name attribute of the particular invoke construct. Although header manipulation not a standard WS-BPEL feature, contemporary WS-BPEL orchestration engines provide means to set request headers, e.g. [19][35].
4. an invocation to the releaseSession service of the adaptation layer is included as a final operation in the transformed scenario.

The adaptation-ready WS-BPEL scenario, as produced by the preprocessor, is then deployed to the WS-BPEL orchestration engine and made available for execution.

6.6 Executing the WS-BPEL scenario

When a WS-BPEL scenario commences execution, its first action will be to invoke the getSessionId web service hosted in the adaptation layer, so as to retrieve a unique session identifier. Afterwards, it invokes the prepareAdaptation web service of the adaptation layer, transmitting to it (i) the session identifier (b) the information regarding the QoS bounds for each functionality, (c) the QoS weights and (d) the functionality binding and the recommendation request information. At this point, the adaptation layer has all the information needed to execute the algorithm described in Section 6.1, so as to produce the execution plan to be followed in the current instance of the WS-BPEL scenario. After the algorithm has been applied, all service bindings (both those specified by the consumer and which were provided as input to the prepareAdaptation web service, as well as those produced as output of the adaptation algorithm) are stored into the session memory, tagged with the session identifier. Effectively, the consumer session memory stores, for each session, the mappings between functionality invocations within the particular session and the concrete services that these invocations should be directed to. Assuming that in the example presented in Section 6.5 the algorithm would determine that the bookTicket functionality should be bound to the Swissair service, the information that would be inserted into the session memory would be as shown in figure 21.

```
(Session: <as returned by getSessionId>;
 Bindings: (    bookTicket: Swissair;
                bookHotel: GrandResort;
                attendEvent: NBAFinals) )
```

Listing 4-3 Information inserted in session memory

The prepareAdaptation web service finally stores the service bindings it received as input parameters into the usage patterns repository, making thus the usage information available for future recommendation formulations.

When the WS-BPEL orchestration engine executes an invoke construct, the invocation is directed to the adaptInvocation service of the adaptation layer, due to the transformations made by the preprocessor (cf. item 3 in Section 6.5, above). Upon reception of an incoming request, the adaptInvocation service proceeds as follows:

1. it extracts from the request headers the session identifier and name of the invoke construct.
2. Using the session identifier, it retrieves from the session memory the service bindings pertinent to the particular session, and then it uses the name of the invoke construct to extract the binding of the specific functionality.
3. The request is then forwarded to the service indicated by the binding, the result is collected and finally it is returned to the WS-BPEL orchestration engine, as a reply to the original invocation.

Finally, when the WS-BPEL scenario reaches its end, it invokes the releaseSession web service, providing the session identifier as a parameter. The releaseSession service will then remove from the session memory all information pertaining to this session.

6.7 Experimental evaluation

In order to determine the optimal value for parameter CFweight and assess the performance of our approach and the quality of the adaptations it produces, a set of experiments have been conducted. The first experiment aimed to offer insight on the effect of the CFweight parameter on the formulation of the solution chosen by the algorithm and the solution's quality. The performance-related experiments aim to measure and quantify the overhead incurred due to the introduction of the middleware. On the other hand, the experiments assessing the quality of the adaptations aim to provide insight on how the QoS of the execution plan proposed by the algorithm in Section 6.2 compares with the QoS of the execution plans proposed by the plain QoS-based algorithm (which is optimal) and the plain CF-based algorithm. In this experiment it is also included a "random" algorithm (i.e. randomly select a service fulfilling the QoS constraints specified by the user), in order to determine whether how the QoS of the execution plans formulated by the algorithm in Section 6.2 compares with the average execution plan.

In more detail, in the performance-related experiments the overhead imposed due to the following activities executed in the proposed scheme have been evaluated: (a) execution plan formulation (invocation of the prepareAdaptation service) (b) invocation redirection (the extra network messages to and from the middleware during each service invocation and the querying of the session memory) and (c) housekeeping

activities (i.e. invocations to getSessionId and releaseSession services, as well as update of the usage patterns repository). In these experiments the following parameters have been varied:

1. the number of concurrent invocations,
2. the size of the usage patterns repository,
3. the number of functionalities in the WS-BPEL scenario and
4. the number of recommendations requested per invocation.

The time needed by the preprocessor to transform the original WS-BPEL scenario into its "adaptation-ready" form has not been included in this evaluation, because the preprocessor operates in an offline fashion, not imposing thus any overhead to the performance of the production system. In all experiments, the semantic service repository was populated with synthetic data having an overall size of 2.000 web services[1]. This arrangement reduced the number deployed services to 20, and does not affect the adaptation algorithm operation, since endpoints are only considered in the execution phase, after the adaptation is performed. Note also that in a real-world setting, these services would be deployed on different machines (the alternative providers' machines)); these services account for 20 different functionalities, with each functionality having 100 alternative providers. The QoS attribute values in this repository were uniformly drawn from the domain [0, 1]. When conducting a test for a particular number of functionalities, 20 BPEL scenarios were synthetically generated, randomly drawing implementations of distinct functionalities from the repository, and the performance evaluation tests were run for each of the generated scenarios. In the scenario generation process, two consecutive functionality invocations were selected to be executed sequentially (<sequence> construct) with a probability of 0.8 and in parallel

[1] since in this experiment a single[1] machine to host the target web services was used, and deploying 2,000 target web services on a single machine would degrade its performance, for the experiment purposes it was arranged that within the semantic service repository all services delivering exactly the same functionality were mapped to the same implementation on the machine hosting the services. DNS aliases were used to make the service endpoints different at the repository level (e.g. a service implementation *svc1* deployed on the second machine would appear in 100 entries in repository as http://exp1.sdbs.uop.gr/svc1/endpoint ,…, http://exp100.sdbs.uop.gr/svc1/endpoint, with all addresses exp1.testdom.uop.gr, …, exp100.testdom.uop.gr resolving to the same IP address.

(<flow> construct), with a probability of 0.2. Therefore, in all cases, there existed BPEL scenarios involving parallel invocations.

The differences observed, regarding the execution time, when the performance evaluation tests were conducted on different BPEL scenarios involving the same number of functionalities were negligible (less than 2% of the overall time). Some differences were observed in the quality of the proposed solution, which depend on the functionalities that were actually included in the services and the contents of the usage patterns repository; this is to be expected, since the score computed by the collaborative filtering algorithm . Each unique performance evaluation test was run 100 times, and the average value was computed and is shown in the following diagrams; error bars are used in the diagrams to show the minimum and maximum values. The lower QoS bounds for the functionalities were randomly drawn from the domain [0,0.4], while the upper QoS bounds for the functionalities were randomly drawn from the domain [0.6,1]. The weights of the QoS attributes were again randomly selected from the domain [0,1]. In all cases, a uniform distribution was used.

For these experiments two machines were used: the first one (a workstation equipped with one Intel Xeon E5-2620@2.0GHz / 6 cores CPU and 16 GB of RAM) hosted the preprocessor and the clients, while the second one (a workstation with identical configuration to the first, except for the memory which was 64GBytes) hosted the BPEL orchestration engine (a Glassfish application server [20] using the Metro web service stack [63]), the adaptation layer, the target web services, the service repository (which included the tree representation of the subsumption relations and the services' QoS characteristics) and the usage patterns repository. The machines were connected through a 100Mbps local area network. Both repositories (the semantic service repository and the usage patterns repository) were implemented as in-memory hash-based structures, which proved more efficient that using a separate, in-memory database engine (e.g. HSQLDB [21] used in [45] and [46]). The new rows of the usage patterns repository are written periodically to the disk to ensure persistence, while provisions have been made to allow for reloading the semantic service repository in case it changes (e.g. addition or deletion of services, or changes in the QoS values).

6.7.1 Determination of CFweight

In this experiment, 1,000 BPEL scenarios were randomly generated, and for each scenario the adaptation algorithm varying the values of the CFweight parameter from 0% to 60% were run. In the scenario generation process, two consecutive functionality invocations were selected to be executed sequentially (<sequence> construct) with a probability of 0.8 and in parallel (<flow> construct), with a probability of 0.2. For each value of the CFweight parameter, the following metrics were collected:

1. percentage of solutions affected by the CF-based dimension: this metric is computed as the ratio of solutions where the final solution chosen was different than the one proposed by the QoS-based algorithm to the number of examined cases, and it is a measure of the effect that the CF-based component has on the formulation of the final solution. Since the introduction of the CF-based component aimed to allow for taking into account the users' subjective ratings (quality of experience), we would like to increase this effect.
2. Normalized quality of the solution: this metric is computed as the ratio of QoS-based overall utility value (OUV_{QoS}) of the chosen solution to the maximum QoS-based utility value computed by the QoS-based algorithm. Effectively, this metric represents how close the finally chosen solution is to the optimal QoS. Naturally, we would like this metric to be as high as possible.

Figure 22 presents the findings of our experiment. For values of CFweight ≤ 20%, the normalized quality of the solution is very close to the optimal (97.3% for CFweight=20%), however only up to 18.7% of the solutions are affected by the CF-based dimension (or, equivalently, the proposal of the QoS-based algorithm is adopted in the 81.3% of the adaptations); this indicates that when the value of CFweight falls in this range, the result of the CF-based algorithm is not adequately taken into account.

For values of CFweight ≥ 50%, the majority of decisions is affected by the QoS-based dimension (53.1% for CFweight=50% and 60. 3% for CFweight=60%) however the normalized quality of the solution drops to 76.20% for CFweight=50% and to 67.47% for CFweight=60%.

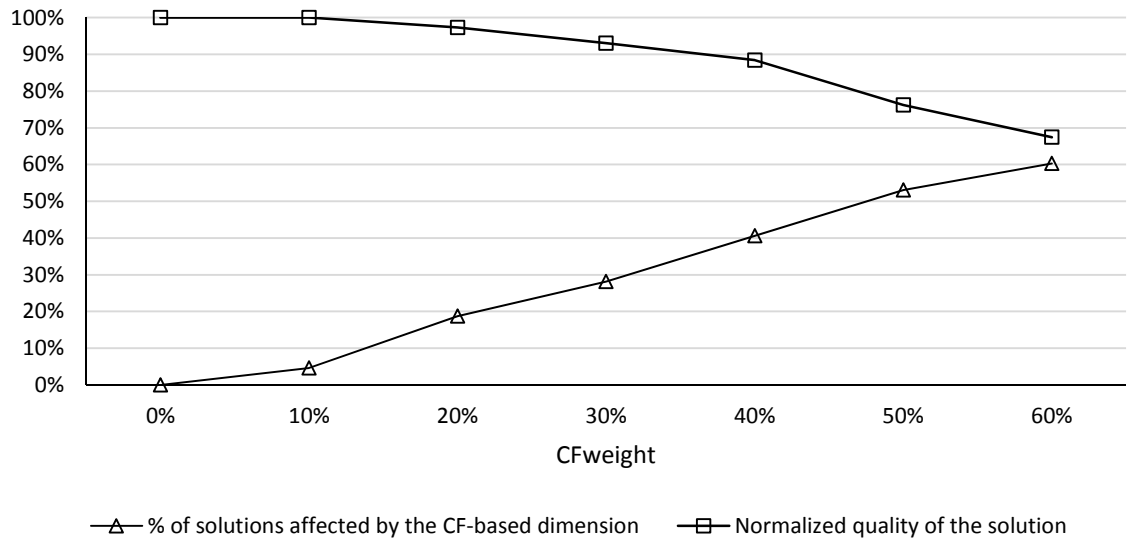


Figure 22: The effect of the CFweight parameter on the on the formulation of the solution and the solution's quality

Setting CFweight to one of the values (30%, 40%) appears to provide a good balance between the goals of maximizing the effect of the CF-based dimension and maintaining a high normalized quality of the solution. If a value of 30% is used, 28.1% of the solutions are affected by the CF-based dimension and the normalized quality of the solution is 93.07, while the respective numbers for CFweight=40% are 40.60% and 88.43% respectively. Considering that a value of 88.43% for the normalized quality is acceptable, CFweight was set to 40%, in order to maximize the effect of the CF-based dimension.

6.7.2 Execution time

Figure 23 presents the execution plan formulation and housekeeping overhead, i.e. the overhead imposed by the use of the invocations to the getSessionId, prepareAdaptation and releaseSession web services, for a varying degree of concurrent WS-BPEL scenario invocation requests arriving to the WS-BPEL orchestration engine. In this experiment, the usage patterns repository was set to include 1,000 qualifying entries (i.e. the usage patterns repository contained 1,000 entries matching the functionality for which a recommendation was requested), the number of functionalities in the WS-BPEL scenario was set to 10 and one recommendation was requested (i.e. 9 service bindings were fixed by the consumer). We can notice that the overhead is

increasing linearly with the number of concurrent invocations, while we can also observe that even with 250 concurrent invocations the overhead is less than one second.

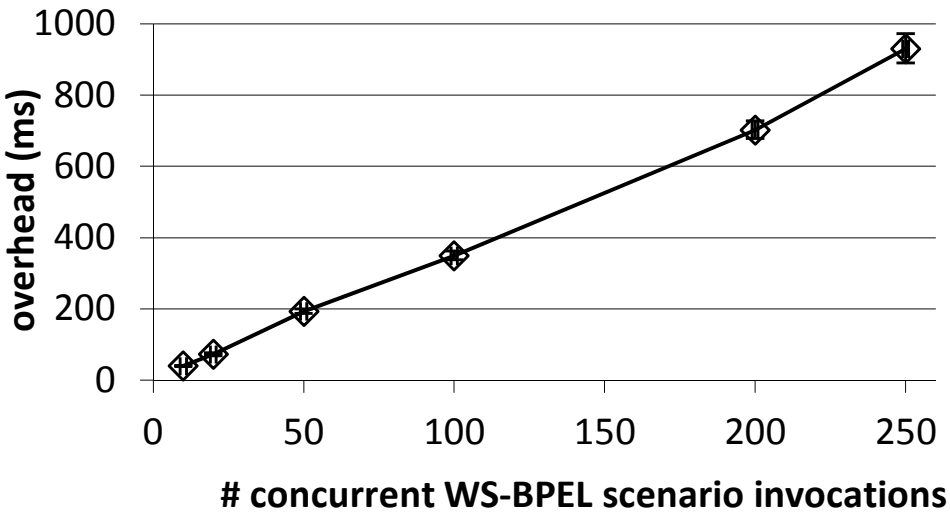


Figure 23: Execution plan formulation and housekeeping overhead for varying degrees of concurrency

Figure 24 illustrates the time needed to perform the computation of the execution plan and the housekeeping tasks under a varying number of functionalities within the WS-BPEL scenario and for different counts of qualifying records in the usage patterns repository (250, 500 and 1000 records). In all cases, the concurrency level was set to one (a single request was submitted and processed) and one recommendation was requested. The recorded times were found to increase between 8% and 14% when the number of functionalities increases by one; this is owing to the time needed for the extra processing to compute the similarity score for the extra service, since the integer programming problems formulated in both cases are identical and therefore their solution time is not affected. The time to compute the similarity score has been optimized, by keeping the similarity metrics between all service/category pairs pre-computed in a hash table and looking them up as needed, instead of computing them on demand. The hash table is populated when the middleware bootstraps and when the service repository is reloaded.

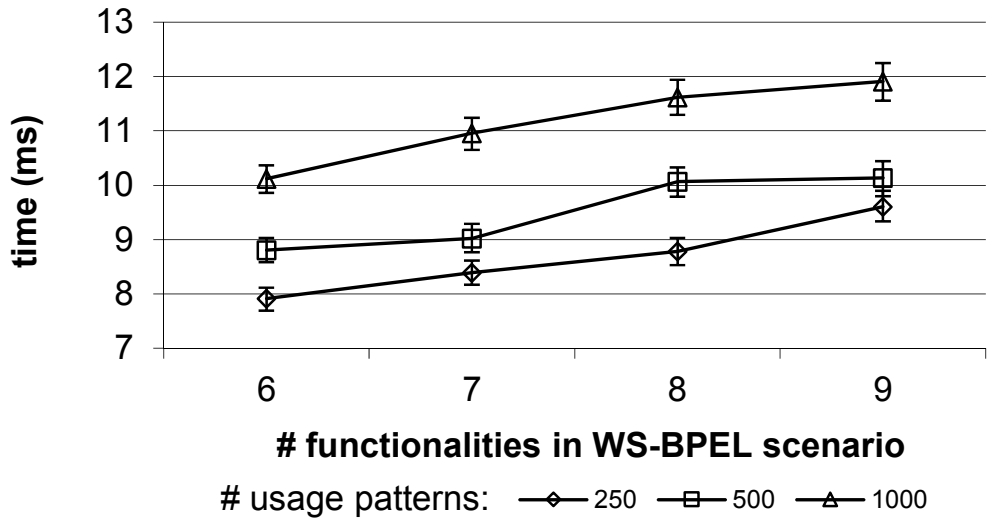


Figure 24: Execution plan formulation overhead for varying number of functionalities within the WS-BPEL scenario and qualifying usage patterns

In figure 24 we can also notice that the execution plan formulation overhead is increasing at a slower rate with respect to the number of the qualifying usage patterns (on average, an increase of 45.68% when the number of qualifying usage patterns increases by 4). This indicates that the algorithm scales well with the size of the usage patterns repository, however for large usage patterns repository sizes, techniques reducing the execution time, such as the one proposed in [64] can be employed.

Finally, figure 25 illustrates the time needed to formulate the execution plan, with respect to the number of qualifying records in the usage patterns repository and the number of recommendations requested in a single WS-BPEL scenario invocation. In all cases, the concurrency level was set to one (a single request was submitted and processed) and the WS-BPEL scenario whose execution (and adaptation) was requested contained 10 functionalities. We can observe that the time needed for each recommendation is fairly stable, e.g. the time needed for making three recommendations is approximately triple the time needed for making one recommendation when the size of the usage patterns repository remains stable. This is due to the fact that each recommendation in the CF-based is made individually by repeating the same steps of the algorithm, and this extra time is merely added to the overall algorithm execution time. The behavior regarding the number of qualifying usage patterns is analogous to the one observed in figure 14. In summary, the recommendation overhead increases linearly with respect to the number of recommendations requested, hence the proposed approach is scalable.

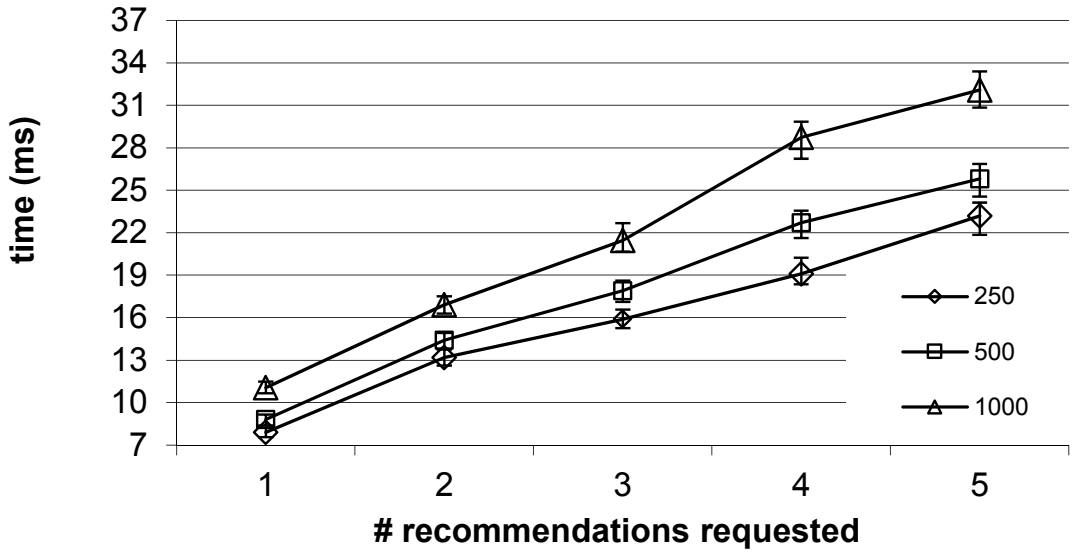


Figure 25: Recommendation overhead for varying number of qualifying usage patterns and number of requested recommendations

6.7.3 Execution plan QoS

Figure 26 depicts the QoS of the execution plan formulated by different algorithms for a number of trial cases, aiming to provide insight on how the QoS of the execution plan proposed by the algorithm in Section 6.2 compares with the QoS of the execution plans proposed by the plain QoS-based algorithm described in [45] and the plain CF-based algorithm described in [46]; as stated above, a “random” algorithm is also included in order to determine whether how the QoS of the execution plans formulated by the algorithm in Section 6.2 compares with the average execution plan.

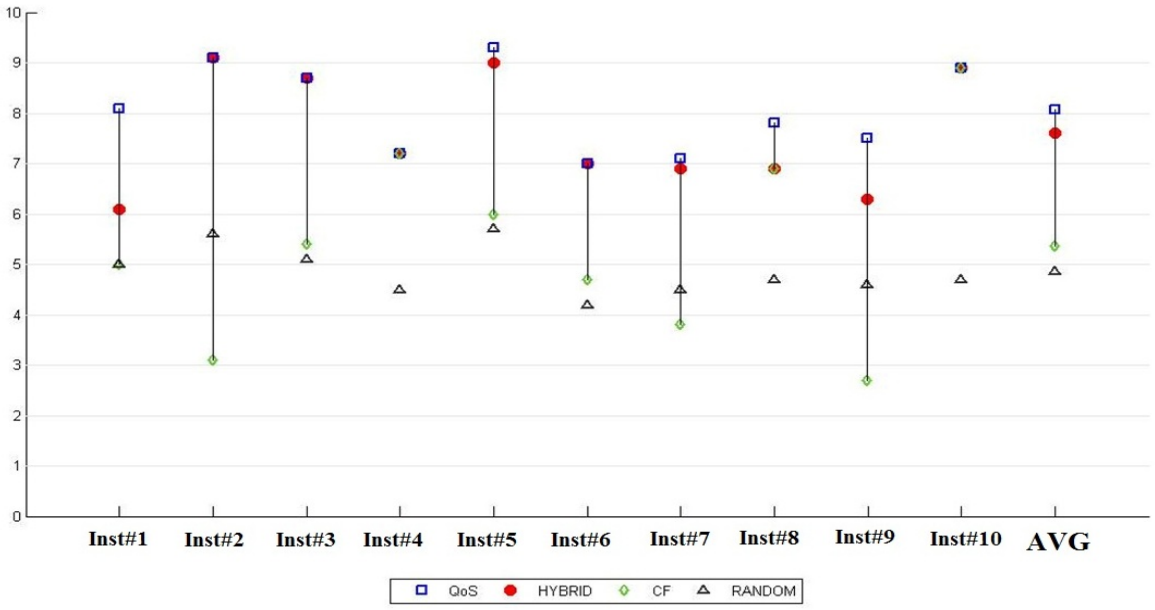


Figure 26: QoS of the execution plans proposed by different algorithms for ten trial cases

The trial cases in this diagram correspond to the invocation of a WS-BPEL scenario containing 10 functionalities in total and requesting 1 recommendation (the remaining 9 functionalities were bound to specific services). The lower QoS bounds for the functionalities were randomly drawn from the domain [0,0.4], while the upper QoS bounds for the functionalities were randomly drawn from the domain [0.6,1]. The weights of the QoS attributes were again randomly selected from the domain [0,1]. In all cases, a uniform distribution was used.

Note that since the QoS-based and the CF-based algorithms, as described in Sections 4.1 and 5.2 respectively, formulate lists of execution plans, the QoS value shown in the diagram for these algorithms is the one corresponding to the execution plan that has attained the biggest score, i.e. the execution plan that would be selected by each algorithm, if it were to decide on its own for the adaptation. Note also that since the random algorithm is non-deterministic, each test case for this algorithm was run 100 times and the mean execution plan QoS was used in the diagram.

In the diagram we can notice that the combined algorithm chooses execution plans whose QoS are very close to the optimal ones: the ratio (combinedQos / optimalQoS) varies between 0.75 and 1, with an average of 0.92. This is considerably higher that the corresponding ratio achieved by the CF-based algorithm (min: 0.34, max: 1, average: 0.65; this is in-line with the results presented in [44]) and the ratio attained by the random algorithm (min: 0.62, max: 0.75, average: 0.65). In all cases, the combined algorithm achieves a QoS equal or higher than pure CF-based one, indicating its ability

to tailor the execution considering the proposals of the CF-based algorithm and at the same time maintain a high QoS.

Of particular interest in figure 26 are trials 2, 5 and 6, in which the deviation between the QoS computed by the CF-based and the QoS-based algorithms is high, however the QoS of the composition is equal to the one proposed by the QoS-based algorithm: these cases correspond to occasions that the usage patterns repository sparsity is high, and therefore the confidence to the proposal of the CF-based algorithm is low, leading the combination step to practically disregard the CF-based algorithm's proposals. In trial 3 sparsity was low, however the execution plan proposed by the QoS-based algorithm was finally adopted: this is because the same execution plan was ranked third by the CF-based algorithm, and hence it achieved the highest WCombMNZ score in the combination step. Conversely in trial 8 the proposal of the CF-based algorithm was adopted, being the fourth runner up in the QoS-based algorithm list and achieving the highest WCombMNZ score in the combination step. In trial 4, both algorithms ranked first the same execution plan, and hence it was selected. In all other trials, an execution plan that was high in the list of both algorithms, but not first in either list, was selected.

7. MONITORING AND FEEDBACK DATA

In this chapter, a framework which provides runtime adaptation for BPEL scenarios is presented. The adaptation is based on (a) quality of service parameters of available web services (b) quality of service policies specified by users (c) collaborative filtering techniques, allowing clients to further refine the adaptation process by considering service selections made by other clients, (d) monitoring, in order to follow the variations of QoS attribute values and (e) on users' opinions services they have used.

7.1 Prerequisites

As far as Qos Concepts and subsumption relationship representation is concerned, the ones presented in chapter 3 are adopted. Furthermore, in order to perform CF-based adaptation, a usage patterns repository with user ratings for services is required. In this chapter, the representation used in [46] is adopted, where the ratings repository is modelled as a table having a number of columns equal to the functionalities present in the BPEL scenario, and one row for each BPEL scenario execution. Cell i,j is filled with value S if during the i^{th} execution of the BPEL scenario, service S was used to implement functionality j; cell (i, j) may be also blank, if during the i^{th} execution of the BPEL scenario functionality j was omitted. In order to accommodate user ratings, we extend this repository by adding one column per functionality. This column stores an integer value from the domain [1, 10], corresponding to the rating given by the user that executed the particular scenario instance, as described in [84]. For the cases that the user has not provided a rating, a null value is stored and the CF-based algorithm uses a default value, as explained in section 5.2. The BPEL scenario adaptation unit inserts new records to the usage patterns repository, when the concrete services that will be invoked in the context of a particular BPEL scenario execution are decided, while the user evaluation collection module arranges for storing the user rankings in the relevant columns.

Table 10: Example usage patterns repository

#exec	Travel	R_{travel}	Hotel	R_{Hotel}	Event	R_{Event}
1	OlympicAirways	8	YouthHostel	3	ChampionsLeague	7
2	SwissAir	6	Hilton	9	GrandConcert	6
3	HighSpeedVessels	null	YouthHostel	null		
4	LuxuryBuses	4		null	EuroleagueFinals	9
5	Lufthansa	7	GrandResort	8	OperaPerformance	6
6	AirFrance	null	Hilton	null		
7	LuxuryBuses	null	YouthHostel	null	ChampionsLeague	null

7.2 The service recommendation algorithm

As stated in sections 4.1, 5.1 and 6.1, this approach follows the horizontal adaptation algorithm, i.e. it leaves the composition logic intact and adapts the execution by selecting which concrete service implementation will be used in each specific invocation. In order to perform this task, the algorithm takes into account the following criteria:

- The consumer's QoS specifications (bounds and weights).
- Designations on which exact services should be invoked, if such bindings are requested by the consumer (e.g. a user wanting to travel using Air France).
- Designations on which functionalities should not be invoked (e.g. a user wanting to book a trip without scheduling any event attendance).
- The QoS characteristics of the available service implementations, including monitored values of the QoS attributes of the services.
- The service subsumption relationships.
- The usage patterns repository, including ratings entered by the users.

The approach proposed in this chapter incorporates two different candidate service ranking algorithms, the first examining the QoS aspects only ([45]) and the second being based on CF techniques ([46]). The algorithms run in parallel to formulate their suggestions regarding the services that should be used in the adapted execution, and subsequently their suggestions are combined, through a metasearch score combination algorithm with varying weights, as described in [84].

7.3 The modified QoS-based adaptation algorithm

The QoS-based adaptation algorithm initially identifies the services which are candidate to be used for delivering functionalities in the context of the current BPEL scenario, respecting the QoS-bounds set by the user, and subsequently computes the k-best service assignments to the functionalities requested for the particular scenario execution. In more detail, the algorithm proceeds as follows:

- For each functionality f_i for which adaptation has been requested, the algorithm retrieves from the semantic service repository the concrete services that (a) deliver this functionality and (b) respect the QoS bounds set by the users. These are the candidates for implementing functionality f_i. Formally, this is expressed as
  $C(f_i) = \{s_{i,j}: (s_{i,j}$ exact $f_i \vee s_{i,j}$ plugin $f_i) \wedge QoS_{min}(req, f_i) \leq QoS(s_{i,j}) \leq QoS_{max}(req, f_i)\}$
  Note that in all steps of this algorithm, the QoS values for response time and availability considered for each service are those returned by predictor methods [66] and [68], respectively, as described in [84].
- Subsequently, the algorithm formulates an integer programming problem to compute the k-best solutions regarding the assignment of concrete services $s_{i,j}$ to each functionality f_i. To express the integer programming optimization problem in this work we adopt the concrete service utility function used in [47], which is

$$U(s_{i,j}) = \sum_{p=1}^{3} \frac{Q_{max}(i,p) - q_p(s_{i,j})}{Q_{max'}(lp) - Q_{min'}(p)} * w_p \quad (1)$$

  where $q_p(s_{i,j})$ is the value of the p^{th} QoS attribute of concrete service $s_{i,j}$ (the first QoS attribute being response time, the second cost and the third one availability), w_p being the weight assigned to the p^{th} QoS attribute,

$$Q_{max}(i,p) = \max_{s \in C(f_i)} q_p(s)$$

  [i.e. the maximum value of QoS attribute p among possible concrete service assignments for functionality f_i], and $Q_{max'}(p)$ [resp. $Q_{min'}(p)$] being the overall maximum (resp. minimum) value of QoS attribute p within the service repository. Using the utility function, the computation of the best solution is expressed as the following integer programming problem: maximise the overall utility value given by

$$OUV_{QoS} = \sum_{i=1}^{F} \sum_{j=1}^{|C(f_i)|} U(s_{i,j}) * x_{i,j}$$

where F is the number of functionalities f_i requiring adaptation, and each $x_{j,i}$ is a binary variable taking the value 1 if $i_{j,j}$ is selected for delivering functionality f_i, and 0, otherwise. Since each functionality f_i is delivered in the final execution plan by exactly one concrete service, the maximization of the utility value is subject to the constraint

$$\sum_{j=1}^{|C(f_j)|} x_{i,j} = 1, \forall i: 1 \leq i \leq F$$

This problem is then solved and the k-best solutions are obtained. Note that this formulation employs the sum function to rate the availability of the composite service taking into account the availability values of the constituent services, rather than the product function, as denoted in Table 1. The transformation from the product function to the sum function is achieved by applying the logarithmic function to the computation of availability [54], since $log(\prod_{i=1}^{n} av_i) = \sum_{i=1}^{n} log(av_i)$. Through this transformation, the problem can be expressed as an integer programming problem and solved efficiently.

The solutions are saved, together with their overall utility score, for perusal in the combination step. In order to solve the integer programming problem computing the k-best solutions, the IBM ILOG CPLEX optimizer was used. In our implementations, we have set k=20.

7.4 The modified CF-based algorithm

The CF-based algorithm employed in our proposal is an adaptation of the standard GroupLens algorithm [69], modified to take into account the semantic distance of the services realizing the same functionality, as described in [46]. For instance, rows 2 and 5 of table 2 are considered “semantically close”, since they both list air transport for travel, a first class hotel for accommodation and classical music events; on the other hand rows 2 and 7 of the same table are considered “semantically distant”, since all three services correspond to diverse real world counterparts (air travel vs. bus, 1st class hotel vs. 3rd class, concert vs. sports). Taking this into account, when a request arrives asking for travel via AirFrance and accommodation in GrandResort and requesting a recommendation for event attendance, the ratings in rows 2 and 7 must be taken more strongly into account than those in row 7, since the former two rows are “closer” to the one under adaptation.

To accommodate this adaptation, we extend the formula of cosine similarity between two rows $\vec{X}$, $\vec{Y}$ of the usage patterns repository as follows:

$$r(\vec{X},\vec{Y}) = \frac{\sum_{k=1}^{n}(\vec{X}[k] * \vec{Y}[k] * d(\vec{X}[k],\vec{Y}[k]))}{\|\vec{X}\| * \|\vec{Y}\|} \quad (2)$$

We can observe in equation (2) that the standard cosine similarity metric has been extended to accommodate the semantic distance between the services that realize the same functionality in rows $\vec{X}$ and $\vec{Y}$; this is accomplished by multiplying each term of the sum in the nominator by a metric of the semantic distance between the two services, which is denoted as $d(s_1, s_2)$ and is computed using the formula introduced in [27]:

$$d(s_1,s_2) = C - lw*PathLength - NumberOfDownDirection \quad (3)$$

where C is a constant set to 8 [27], lw is the level weight for each path in subsumption tree (cf. Fig. 1), PathLength is the number of edges counted from functionality s_1 to functionality s_2 and NumberOfDownDirection is the number of edges counted in the directed path between functionality s_1 and s_2 and whose direction is towards a lower level in the subsumption tree. For more details in the computation of the semantic distance, the interested reader is referred to [27]. We further normalize this similarity metric in the range [0, 1] by dividing the result computed in the above formula by 8; this way, the multiplication by the normalized similarity metric in equation (2) reduces the correlation coefficient between the two rows by a factor proportional to the semantic distance of the services employed in these rows to realize the same functionality.

For items not explicitly rated, we follow the rationale of [46] according to which usage of a service is an indication of preference, and we choose a rating equal to the 80% of the maximum rating, as described in [84]. This is inline with the findings of [70], which asserts that dissatisfied users will provide negative feedback with a very high probability (≥89%). Rows that have not been rated at all (and therefore have a default value for all ratings) are the reason behind choosing the cosine similarity against the Pearson similarity, since the latter disregards rows whose ratings have no variance (i.e. are all equal).

Using the modified cosine similarity, the CF-based algorithm operates as follows:

1. It retrieves from the usage patterns repository all rows that contain a service implementing the functionality on which a recommendation is requested. For example, if a recommendation on event attendance is requested, only rows 1, 2, 4, 5 and 7 of table 10 will be retrieved.

2. The rows retrieved from step 1 are filtered to retain only those that fulfil the QoS criteria requested by the user.
3. The similarities between the request and each row are computed using the modified cosine similarity metric. The request is represented here as a vector $\vec{R}$, having a rating equal to 10 for each functionality included in the scenario and a rating equal to 0 for each functionality designated as not to be executed.
4. For each distinct service implementing the requested functionality that is included in the remaining rows, we compute its rating prediction using the standard rating prediction formula

   $$p(\vec{R}[k]) = \underset{m}{mean}(\vec{R}[m]) + \frac{\sum_{\vec{N} \in raters(\vec{R}[k])} (\vec{N}[k]) * r(\vec{R}, \vec{N})}{\sum_{\vec{N} \in raters(\vec{R}[k])} r(\vec{R}, \vec{N})}$$

   [69] (we again do not subtract the mean $\overrightarrow{N}$ from $\overrightarrow{N}[k]$, so as not to render useless the rows having only default values).
5. Finally, we retain the 20-best services for each functionality requiring adaptation, for perusal in the combination step.

After the lists of candidates for each individual service that is subject to adaptation have been computed, the algorithm selects the top-20 execution plans with respect to their CF-score. Given an execution plan containing services ($s_{1,i}$, …, $s_{N,k}$) with the similarity scores of the services computed in step 5 being (CFS($s_{1,i}$), …, CFS($s_{N,,k}$)), then the CF-score of the execution plan is equal to CFS($s_{1,i}$)+…+CFS($s_{N,k}$). Computing the top-20 execution plans is modelled as an integer programming optimization problem, formulated in a similar fashion to the one described in section 6.2. Full details on the formulation of the integer programming optimization problem are given in [67].

The CF module has been implemented using Apache Mahout (https://mahout.apache.org/), by subclassing the UncenteredCosineSimilarity class and reimplementing in the subclass the UserSimilarity method, to accommodate the semantic similarity metric described above.

7.5 The combination step

The combination step synthesizes the results given by individual algorithms to produce to a single result. Recall from the previous two subsections that each algorithm produces a set of candidate execution plans, with each execution plan being tagged with the relevant normalized score (QoS-score or CF-score). In order to combine the scores, we use the CombMNZ metasearch algorithm, since it has been found to have the best performance [9] [the CombMNZ rating of a solution is computed by multiplying the sum of the individual scores by the number of non-zero scores, i.e.

$$\mathrm{CombMNZ_i} = \mathrm{m_i} * \sum_{\mathrm{j=1}}^{\mathrm{m_i}} \mathrm{r_j(i)}$$

where m_i is the number of algorithms giving non-zero rating to item i and $r_j(i)$ is the rating given by algorithm j to item i]. After computing the CombMNZ metasearch for all candidate execution plans, the combination step selects the execution plan with the highest score, which will be used to drive the adaptation process.

7.6 An example of the algorithm operation

In this section, an illustrative example on the operation of the adaptation algorithm is given. In this example, we consider the following:

a) The scenario to be adapted is the trip reservation application used in the examples in section 6.2. The scenario includes mandatory invocations to a travel reservation and a hotel reservation service, and an optional invocation to an even attendance booking service.
b) The subsumption relationships that will be used in this scenario are as depicted in figure 27.

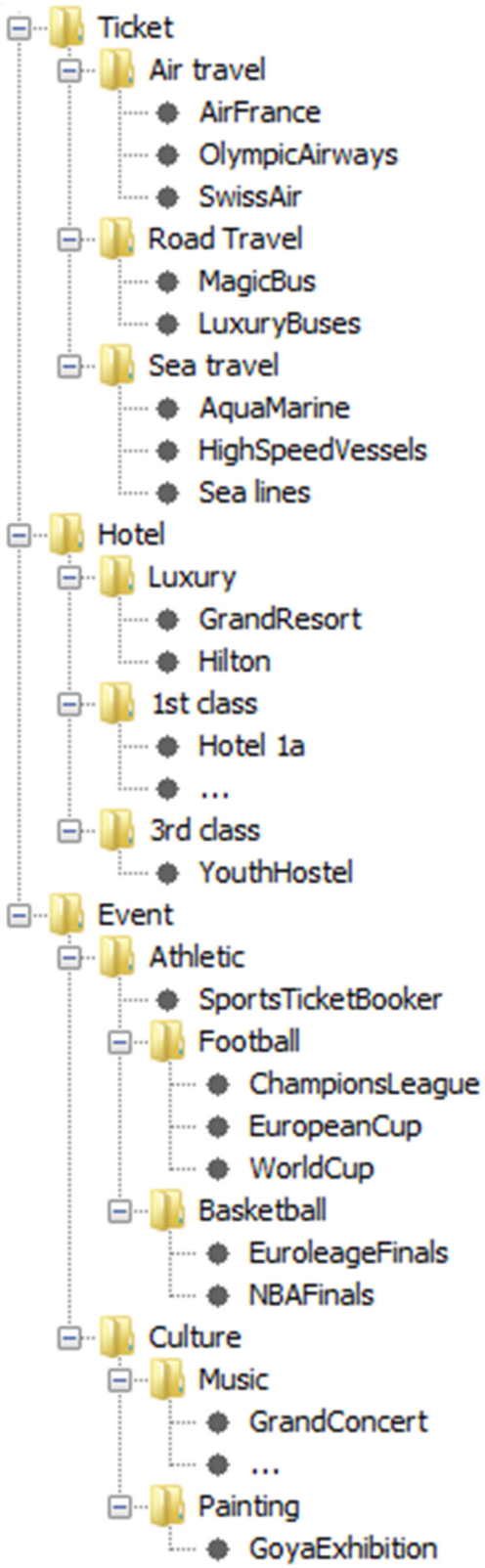


Figure 27: Subsumption relationships tree used in the example

c) The QoS values for the services implementing the “Air travel” functionality are as shown in table 11. The table lists only the QoS values for the service implementing the “Air travel” functionality, since these are the only ones pertinent in this example.

Table 11: QoS values for the services implementing the "Air travel"

Equivalent WS	Cost	Response Time	Availability
AirFrance	8	10	7
Lufthansa	9	8	7
OlympicAirways	2	5	9
Swissair	7	7	8

d) The contents of the usage patterns repository are as shown in table 6-4.

Table 12: Usage patterns repository used in the example

#exec	Travel	R(travel)	Hotel	R(Hotel)	Event	R(Event)
1	OlympicAirways	8	Hilton	3	GrandConcert	5
2	Lufthansa	9	YouthHostel	9	EuropaLeague	7
3	HighSpeedVessels		YouthHostel			
4	HighSpeedVessels	4			ChampionsLeague	9
5	AirFrance	7	GrandResort	8	OperaPerformance	6
6	SwissAir	6	Hilton			
7	LuxuryBuses	9	YouthHostel	6	EuroleagueFinals	9
8	Lufthansa	9	YouthHostel	8	EuroleagueFinals	

e) The user request to be adapted is: AirTravel(R), YouthHostel, ChampionsLeague

which effectively reads: "I want to stay in YouthHostel and attend the ChampionsLeague event, and I want a recommendation regarding an AirTravel service".

The user has set a QoS weight vector equal to W= (0.4, 0.3, 0.3), while the MIN and MAX vectors, setting the lower and upper limits respectively for service QoS attributes, are set as follows:

MINAirTravel=(4, null, 5) , MAXAirTravel=(null, null, null) i.e. a minimum of 4 and 5 is set for the travel cost and availability respectively, while no lower limit is set for its response time. Similarly, no upper bounds are imposed for any service.

f) We assume that the minimum and maximum values for the QoS attributes within the repository are as follows (these are needed in the utility function U): MIN=(2, 1, 2) MAX=(10, 10, 9)

7.6.1 Applying the QoS-based algorithm

1. First, we retrieve from the service repository (table 12) all rows implementing the "air travel" functionality. All rows of the repository qualify (since the excerpt of the repository depicted in table 12 consists exactly of these rows)
2. Subsequently, the rows not meeting the QoS bounds are filtered out. As a consequence, row #1, corresponding to the *OlympicAirways* service, is rejected.

   Subsequently, we compute the utility function *U* for each of the remaining services. The values for the utility function are as follows (recall that the utility function is defined as

   $U(s_{i,j}) = \sum_{p=1}^{3} \left(1 - \frac{Q_{max}(i,\mathrm{p}) - q_p(s_{i,j})}{Q_{max'}(lp) - Q_{min'}(p)}\right) * w_p$):

   U(AirFrance) = $\left(1 - \frac{9-8}{10-2}\right) * 0.4 + \left(1 - \frac{10-10}{10-1}\right) * 0.3 + \left(1 - \frac{8-7}{9-2}\right) * 0.3 = 0.913$

   U(Lufthansa) = $\left(1 - \frac{9-9}{10-2}\right) * 0.4 + \left(1 - \frac{10-8}{10-1}\right) * 0.3 + \left(1 - \frac{8-7}{9-2}\right) * 0.3 = 0.888$

   U(SwissAir) = $\left(1 - \frac{9-9}{10-2}\right) * 0.4 + \left(1 - \frac{10-8}{10-1}\right) * 0.3 + \left(1 - \frac{8-7}{9-2}\right) * 0.3 = 0.788$

Subsequently, we formulate the integer programming problem to maximize the overall utility function

$$OUV_{QoS} = \sum_{i=1}^{F} \sum_{j=1}^{|C(f_i)|} U(s_{i,j}) * x_{i,j}$$

subject to the constraint

$$\sum_{j=1}^{L(i)} x_{i,j} = 1, 1 \leq i \leq F$$

Since now F=1 (F is the number of functionalities for which adaptation is requested), we have that the overall utility function is reduced to

$$OUV_{QoS} = \sum_{j=1}^{|C(AirTravel)|} U\left(s_{AirTravel,j}\right) * x_{AirTravel,j} =$$

$$\mathrm{U(AirFrance)} * \mathrm{x_{AirFrance}} + \mathrm{U(Lufthansa)} * \mathrm{x_{Lufthansa}} + \mathrm{U(SwissAir)} * \mathrm{x_{SwissAir}}$$

subject to the constraint

$$\mathrm{x_{AirFrance}} + \mathrm{x_{Lufthansa}} + \mathrm{x_{SwissAir}} = 1$$

The three possible solutions to this integer programming problem are as shown in table 13:

Table 13: Solutions proposed by the QoS-based algorithm

Solution	*QoS-score*	*Normalized QoS-score*
$x_{AirFrance}$=1, $x_{Lufthansa}$=0, $x_{SwissAir}$=0	0.913	1.000
$x_{AirFrance}$=0, $x_{Lufthansa}$=1, $x_{SwissAir}$=0	0.888	0.800
$x_{AirFrance}$=0, $x_{Lufthansa}$=0, $x_{SwissAir}$=1	0.788	0.000

We save these solutions for perusal in the combination step.

7.6.2 Applying the CF-based algorithm

According to the first step of the CF-based algorithm, we will retrieve from the usage patterns repository (table 14) only those rows that involve the functionality requested for adaptation. Since the functionality for which adaptation is requested is *AirTravel*, rows 3, 4 and 7 will be eliminated, since they involve other means of transportation (sea travel for rows 3 and 4 and bus travel for row 7). Therefore, the rows depicted in table 6 will be retrieved.

Table 14: Rows of the usage patterns repository delivering the functionality under adaptation

1	OlympicAirways	8	Hilton	3	GrandConcert	5
2	Lufthansa	6	YouthHostel	9	EuropaLeague	6
5	AirFrance	7	GrandResort	8	OperaPerformance	6
6	SwissAir	6	Hilton	*null*		
8	Lufthansa	9	YouthHostel	8	EuroleagueFinals	*null*

The second step of the CF-based algorithm eliminates the rows for which the service delivering the functionality under adaptation does not meet the QoS bounds set by the client. Row 1 fails to satisfy them so it is eliminated, and the rows retained for further processing are as shown in table 15. At this point, we fill the *null* value of row #6 and row #8 with the default value (8).

Table 15: Rows of table 6-6 satisfying the QoS bounds

2	Lufthansa	6	YouthHostel	9	EuropaLeague	6
5	AirFrance	7	GrandResort	8	OperaPerformance	6
6	SwissAir	6	Hilton	*8*		
8	Lufthansa	9	YouthHostel	8	EuroleagueFinals	*8*

We now compute the similarity of each row to a request vector $\vec{R} = (10, 10, 10)$, taking into account the semantic distances between the services. The semantic distances between the services pertinent to this adaptation are computed through the formula

$d(s_1,s_2) = (8 – lw*PathLength – NumberOfDownDirection) / 8$

and their values are as follows:

d(AirTravel, Lufthansa) = (8 - 2/3 *1 – 1) / 8 = (19/3) / 8 = 19/24

d(AirTravel, AirFrance) = (8 - 2/3 *1 – 1) / 8 = (19/3) / 8 = 19/24

d(AirTravel, SwissAir) = (8 - 2/3 *1 – 1) / 8 = (19/3) / 8 = 19/24

d(YouthHostel, YouthHostel) = (8 - 1/3 *0 – 0)/ 8 = 8 / 8 = 1

d(YouthHostel, GrandResort) = (8 - 1/3 *4 – 2) / 8 = (14/3) / 8 = 14/24

d(YouthHostel, Hilton) = (8 - 1/3 *4 – 2) / 8 = (14/3) / 8 = 14/24

d(ChampionsLeague, EuropaLeague) = (8 - 1/4 *2 – 1) / 8 = (26/4) / 8 = 26/32

d(ChampionsLeague, EuroleagueFinals) = (8 - 1/4 *4 – 2) / 8 = (5) / 8 = 5/8

d(ChampionsLeague, OperaPerformance) = (8 - 1/4 *6 – 3) / 8 = (14/4) / 8 = 14/32

The third step of the CF-based algorithm is to compute the similarity between the user request vector $\vec{R} = (10, 10, 10)$ and the vectors corresponding to the raters of the functionality for which adaptation is requested. Recall from section 6.3 that the similarity is computed using the cosine similarity metric, using the formula

$$r(\vec{X}, \vec{Y}) = \frac{\sum_{k=1}^{n} (\vec{X}[k] * \vec{Y}[k] * d(\vec{X}[k], \vec{Y}[k]))}{\|\vec{X}\| * \|\vec{Y}\|}$$

Therefore, the similarity metric *r* between the rows of table 15 and the user request vector $\vec{R} = (10, 10, 10)$ are as follows:

$$r(\vec{R}, \overrightarrow{row_2}) = \frac{\sum_{k=1}^{3} (R[k] * \overrightarrow{row_2}[k] * d(R[k], \overrightarrow{row_2}[k]))}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(6 * 10 * \frac{9}{24}\right) + (9 * 10 * 1) + \left(6 * 10 * \frac{26}{32}\right)}{\sqrt{6^2 + 9^2 + 6^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.869$$

$$r(\vec{R}, \overrightarrow{row_5}) = \frac{\sum_{k=1}^{3} (R[k] * \overrightarrow{row_5}[k] * d(R[k], \overrightarrow{row_5}[k]))}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(7 * 10 * \frac{9}{24}\right) + \left(8 * 10 * \frac{14}{24}\right) + (6 * 10 * \frac{14}{32})}{\sqrt{7^2 + 8^2 + 6^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.607$$

$$r(\vec{R}, \overrightarrow{row_6}) = \frac{\sum_{k=1}^{3} (R[k] * \overrightarrow{row_6}[k] * d(R[k], \overrightarrow{row_6}[k]))}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(6 * 10 * \frac{19}{24}\right) + \left(8 * 10 * \frac{14}{24}\right)}{\sqrt{6^2 + 8^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.544$$

$$r(\vec{R}, \overrightarrow{row_8}) = \frac{\sum_{k=1}^{3} (R[k] * \overrightarrow{row_8}[k] * d(R[k], \overrightarrow{row_8}[k]))}{\|\vec{R}\| * \|\overrightarrow{row_2}\|} =$$

$$\frac{\left(9 * 10 * \frac{19}{24}\right) + (9 * 10 * 1) + (8 * 10 * \frac{5}{8})}{\sqrt{9^2 + 8^2 + 8^2} * \sqrt{10^2 + 10^2 + 10^2}} = 0.844$$

Subsequently, we compute each service's rating prediction using, as discussed in section 6.3, the rating prediction formula

$$p(\vec{R}[k]) = \frac{\sum_{\vec{N} \in raters(\vec{R}[k])} (\vec{N}[k]) * r(\vec{R}, \vec{N})}{\sum_{\vec{N} \in raters(\vec{R}[k])} r(\vec{R}, \vec{N})}$$

And therefore we obtain:

p(Lufthansa) = (6*0.869 + 0.844*9)/ (0.869+0.844) = 7.48

p(AirFrance) = 7* 0.607 / 0.607 = 7

p(SwissAir) = 6* 0.544 / 0.544 = 6

These values are then normalized to the range [0,1] by dividing by the maximum possible value of a rating, in our case 10:

p_n(Lufthansa) = (6*0.869 + 0.844*9)/ (0.869+0.844) = 0.748

p_n(AirFrance) = 7* 0.607 / 0.607 = 0.7

p_n(SwissAir) = 6* 0.544 / 0.544 = 0.6

Since the number of possible solutions is less than 20, all solutions are retained. Subsequently, similarly to the case of the QoS-based algorithm, we formulate the integer programming problem, i.e. to maximize the overall utility function

$$OUV_{CF} = (\sum_{i=1}^{F} \sum_{j=1}^{L(i)} prediction(s_{i,j}) * x_{i,j})/F$$

subject to the constraint

$$\sum_{j=1}^{L(i)} x_{i,j} = 1, 1 \leq i \leq F$$

Since the number of functionalities requiring adaptation *F* is equal to 1, the integer optimization problem is reduced to

$$OUV_{CF} = \sum_{j=1}^{3} prediction(s_{AirTravel,j}) * x_{AirTravel,j}) =$$

$$\text{prediction(AirFrance)} * \text{x}_{\text{AirFrance}} +$$

$$\text{prediction(Lufthansa)} * \text{x}_{\text{Lufthansa}} +$$

$$\text{prediction(SwissAir)} * \text{x}_{\text{SwissAir}}$$

subject to the constraint

$$\text{x}_{\text{AirFrance}} + \text{x}_{\text{Lufthansa}} + \text{x}_{\text{SwissAir}} = 1$$

The three possible solutions to this problem are as shown in table 16.

Table 16: Solutions proposed by the CF-based algorithm

Solution	CF-score	Normalized CF-score
$x_{AirFrance}$=1, $x_{Lufthansa}$=0, $x_{SwissAir}$=0	0.700	0.675
$x_{AirFrance}$=0, $x_{Lufthansa}$=1, $x_{SwissAir}$=0	0.748	1.000
$x_{AirFrance}$=0, $x_{Lufthansa}$=0, $x_{SwissAir}$=1	0.600	0.000

7.6.3 Combining the results

Using the rules from section 6.2 for computing CFweight and QoSweight (we get CFweight=0.40 and QoSweight = 0.6), we finally apply the CombMNZ metasearch algorithm to the normalized scores to compute the final score of each solution. The CombMNZ algorithm adds the individual weighted scores and multiplies the result by the number of algorithms proposing each solution. In our case, all solutions are proposed by both the QoS-based algorithm and the CF-based algorithm, hence the results are as follows:

$CombMNZ_{AirFrance}$ = (1.000*0.6 + 0.675*0.4) * 2 = 1.740

$CombMNZ_{Lufthansa}$ = (0.800*0.6 + 1.000*0.4) * 2 = 1,760

$CombMNZ_{SwissAir}$ = (0.000*0.6 + 0.000*0.4) * 2 = 0.916

We can observe that the $CombMNZ_{Lufthansa}$ score is the maximum among all scores, hence the Lufthansa service will be chosen for the particular adaptation, so the user scenario execution plan becomes «Lufthansa, YouthHostel, ChampionsLeague».

7.7 The execution adaptation architecture

The execution adaptation architecture, illustrated in figure 28, follows the middleware-based approach, with an adaptation layer intercepting web service invocations and appropriately directing them to the services decided by the adaptation algorithm.

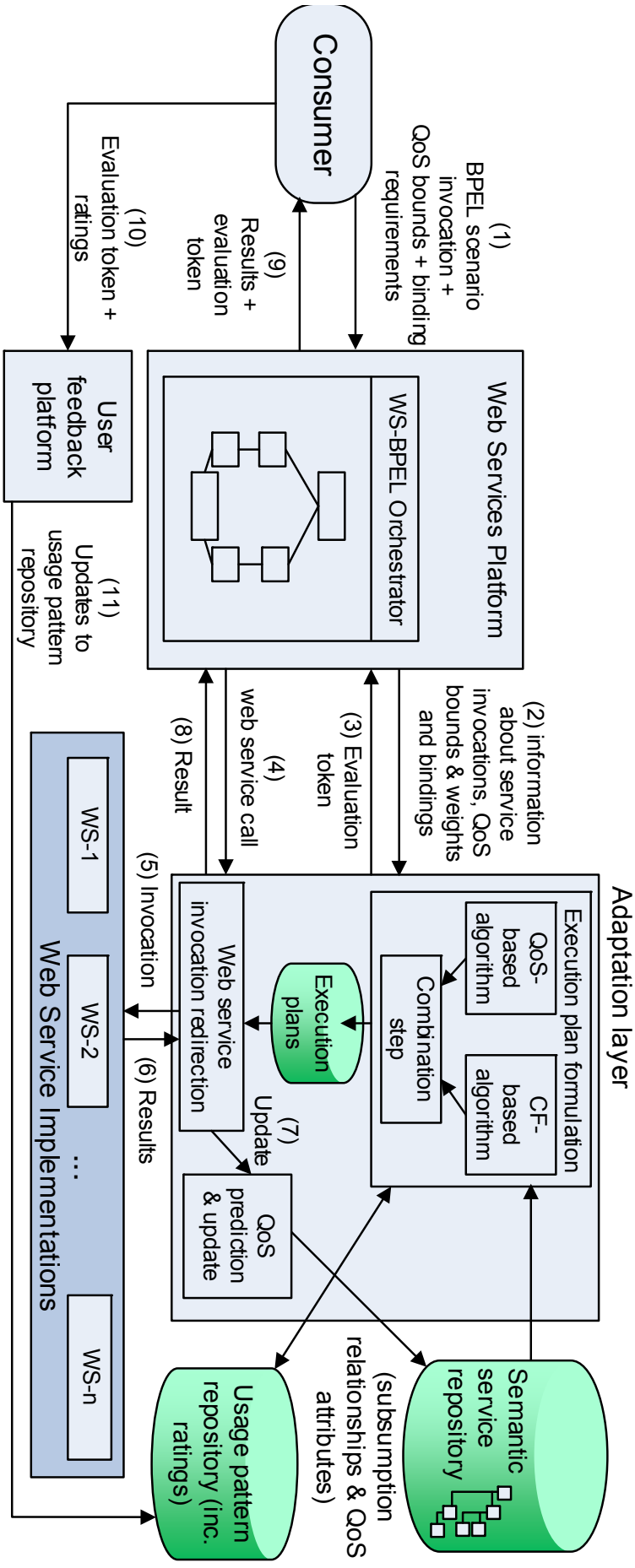


Figure 28: The execution adaptation architecture

As shown in figure 28, the BPEL scenario execution initially passes to the adaptation layer the information regarding service invocations that will be performed, QoS bounds and weights as well as specific service bindings. When the adaptation layer receives this information, it applies the adaptation algorithm to formulate the execution plan for the particular scenario execution (i.e. decide the actual services that will be invoked to deliver each functionality) and stores the execution plan for later perusal. Subsequently, when a web service invocation is intercepted by the adaptation layer, the respective execution plan is retrieved from the execution plan storage, the web service decided to deliver the specific functionality is extracted and the invocation is routed accordingly to that service. Note that steps (4)-(8) depicted in figure 28 are repeated multiple times within each BPEL scenario execution, once per web service invocation performed. When the invocation to a service implementation has concluded, the data regarding the service's response time and availability are passed to the QoS prediction and update module, which computes the predicted values for the respective QoS parameters and updates the corresponding elements in the semantic service repository.

Additionally, the BPEL scenario returns at the end of its execution, along with the result, an evaluation token, which the consumer may use to enter the ratings for the services s/he has used in the context of the BPEL scenario execution. The evaluation token is returned in the response headers, to retain the response payload schema intact. To accommodate this additional functionality (passing the necessary information to the adaptation layer and returning the evaluation token), the BPEL scenario is preprocessed as described in [46] before being deployed to the web services platform, with the preprocessing step injecting the necessary invocations to the adaptation layer into the scenario, and the result of the preprocessing step is then deployed and made available for invocations.

7.8 Experimental evaluation

In this section, the experiments are reported aiming to substantiate the feasibility of the proposed approach, both in terms of execution time (quantifying the introduced overhead) and solution quality. For these experiments two machines were used: (a) a workstation, equipped with one 6-core Intel Xeon E5-2620@2.0GHz CPU and 16 GB of RAM, which hosted the preprocessor and the clients and (b) a workstation with identical

configuration to the first, except for the memory which was 64GBytes, that hosted the BPEL orchestration engine, the adaptation layer, the target web services, the service repository and the usage patterns repository. The machines were connected through a 1Gbps local area network. Both repositories (the semantic service repository and the usage patterns repository) were implemented as in-memory hash-based structures, which proved more efficient than using a separate (memory or disk-based) databases.

Regarding the execution time, the introduction of the adaptation layer imposes overheads in two points: (a) when the BPEL scenario passes to the adaptation layer information regarding the web services to be invoked, their QoS-bounds and weights as well as the service bindings in order to formulate the execution plan and (b) when a service is invoked, since two extra messages to/from the adaptation layer and some processing therein are introduced. These overheads are quantified in [45] (a) and (b) respectively, for varying degrees of concurrency. In this experiment, the usage patterns repository was set to include 1,000 qualifying entries (i.e. the usage patterns repository contained 1,000 entries matching the functionality for which a recommendation was requested), the number of functionalities in the BPEL scenario was set to 10 and one recommendation was requested (i.e. 9 service bindings were fixed by the consumer).

Figure 29 indicates that the overhead to formulate the execution plan is small per BPEL scenario execution (approximately 4 msec) and scales linearly with the number of concurrent executions. Clearly, having available more execution units in the machine performing the adaptation, or offloading some tasks to other machines (e.g. execution of web services, which in a real-world case would be hosted in the service providers' computers) would result to smaller overheads in high degrees of concurrency.

Figure 30 indicates that the additional per-service execution overhead is small, ranging from 3 to 8 msec, depending on the level of concurrency.

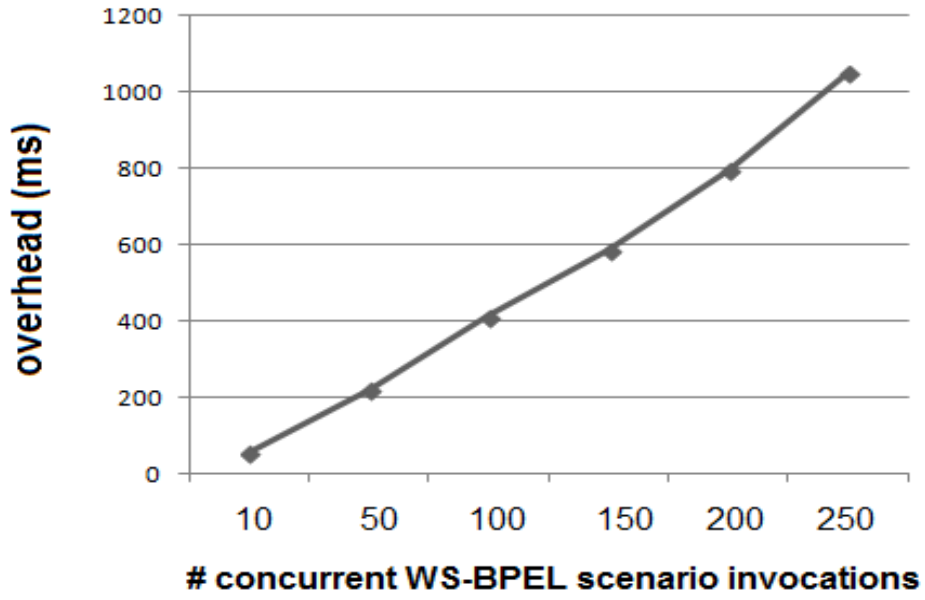


Figure 29: Execution plan formulation overhead for varying degrees of concurrency

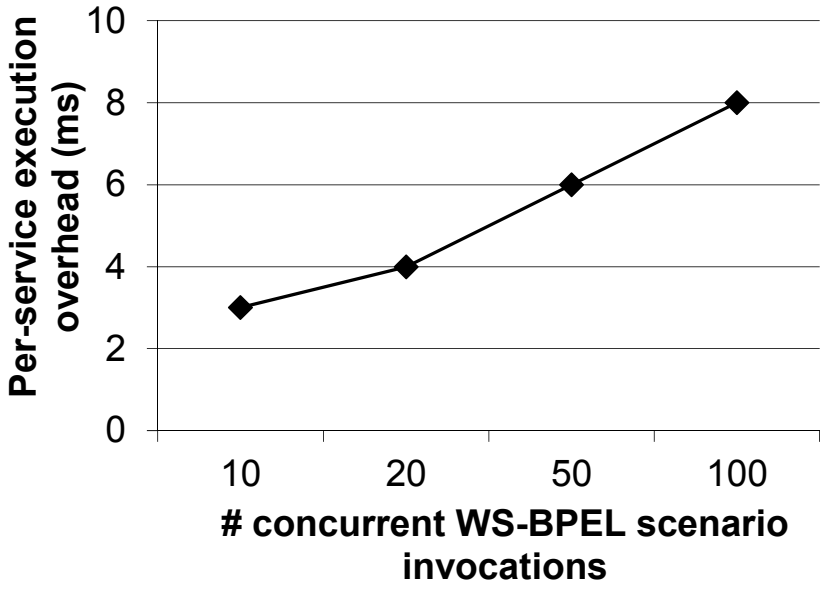


Figure 30: Per-service invocation overhead, for varying degrees of concurrency

Figure 31 compares the QoS of the execution plan formulated for a number of trial cases (10 cases) by (i) the QoS-based algorithm (ii) the CF-based algorithm and (iii) the combined algorithm proposed in this chapter. The diagram shows that the combined algorithm succeeds in maintaining a high solution QoS (93% of the QoS of the optimal QoS that can be attained on average, and higher than the 80% of the optimal QoS in all cases), while at the same time allows to consider the user ratings on the services. Each

trial case in this diagram corresponds to the invocation of a WS-BPEL scenario containing 10 functionalities in total and requesting 1 recommendation (the remaining 9 functionalities were bound to specific services). The lower QoS bounds for the functionalities were randomly drawn from the domain [0,0.4], while the upper QoS bounds for the functionalities were randomly drawn from the domain [0.8,1]. The weights of the QoS attributes were again randomly selected from the domain [0,1]. In all cases, a uniform distribution was used. Finally figure 32 illustrates that the introduction of QoS feedback and estimation on the response time leads to the formulation of more efficient execution plans, when the services' actual QoS deviates from the values initially stored in the repository. In this experiment, a varying number of concurrent client to request adaptation for a particular functionality has been used, setting a high weight on the response time, which leads to the selection of services declared to have high QoS values (e.g. low response times). In situations with high degrees of concurrency these services are overloaded, hence the offered QoS drops. When QoS feedback and estimation is used, the QoS degradation trend of these services is noticed and QoS estimations are adjusted accordingly, therefore other services will then become the ones with optimal values. Consequently, invocations are spread among different service providers, resulting in smaller response times on average (shown in figure 32), and hence recommendations of higher quality.

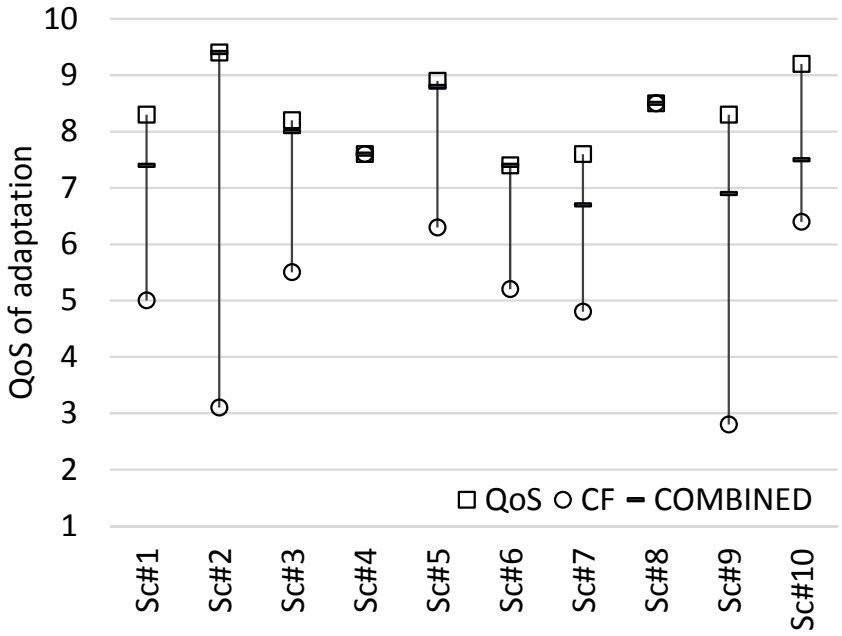


Figure 31: QoS of solutions proposed by individual algorithms and the combined one

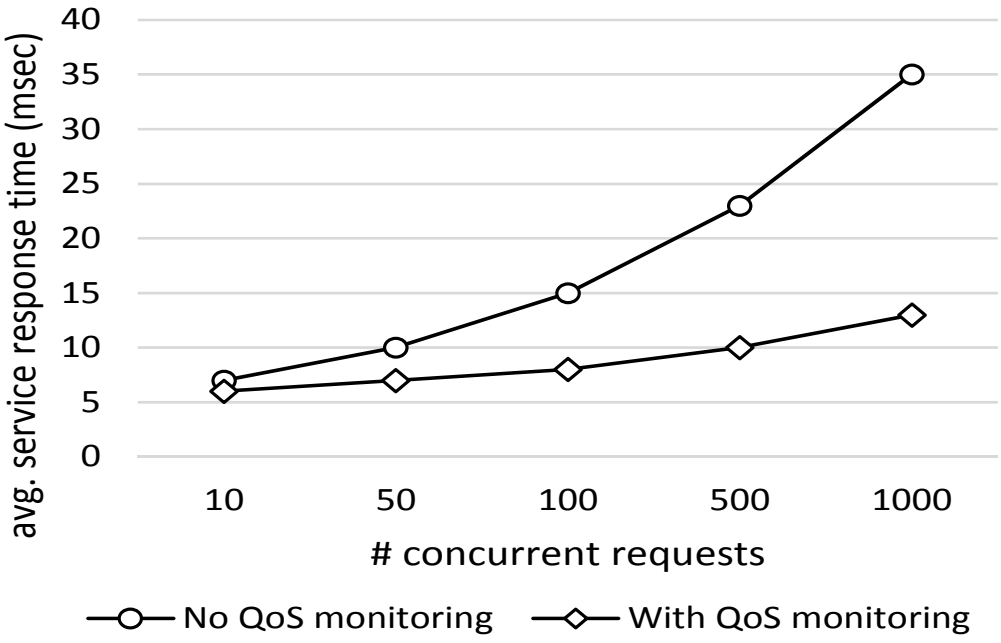


Figure 32: Improvement of response time due to the introduction of QoS monitoring and estimation

8. CONCLUSION AND FUTURE WORK

In this dissertation, frameworks for adapting the execution of BPEL scenarios are presented. To perform the adaptation, the metasearch paradigm is followed, by combining two candidate execution plan ranking algorithms. The first one examines the execution plan QoS aspects only, while the second is based on CF techniques. The framework provides means for monitoring the QoS parameters of the services and adjusting accordingly the values of the services' QoS attributes, as well as accepting user ratings for the services they have used, which are taken into account by the CF-based algorithm. The execution architecture includes a scenario preprocessing step, which transforms existing WS-BPEL scenarios into an "adaptation ready" form, and an adaptation layer (a middleware component intervening between the WS-BPEL execution engine and the actual service implementation), which undertakes the tasks of executing the adaptation algorithm and redirecting invocations to the selected services.

The proposed frameworks have been experimentally validated regarding (i) their performance, (ii) the quality of execution plans generated and (iii) the effectiveness of the QoS monitoring and estimation mechanisms. The proposed approaches have been also found to be scalable, exhibiting a linear increase in the imposed overhead. The overhead imposed for performing the adaptation has been found to be small (approximately 4-5msec per concurrent user), while the execution architecture's scalability has also proven to be satisfactory, exhibiting a linear increase for all tested factors (concurrent users, size of usage patterns repository, number of recommendations requested per invocation and number of functionalities within the WS-BPEL scenario). The final proposed algorithm has also been found to choose execution plans whose QoS are very close to the optimal ones, while at the same time crafting the execution plans taking into account the proposals of the CF-based algorithm.

Our future work will focus on the integration of QoS-adherence monitoring mechanisms, such as those described in [24], as well as gathering and utilizing statistical information from prior scenario executions as information sources to the adaptation process. This statistical information will be used to quantify aspects regarding the execution of control constructs in the scenario, e.g. the probability that a conditional branch is executed or the distribution of the number of executions of a loop. These quantities will be then taken into account by the adaptation algorithm. QoS-adherence monitoring mechanisms could also provide feedback to the adaptation algorithm, by either changing the

services' QoS parameters in the repository (e.g. increasing the value of a services' response time QoS attribute if it consistently exhibits worse response time than the one recorded in the repository) or by instructing the algorithm to refrain temporarily from choosing a service (e.g. if the service has been recently invoked many times and its degraded performance could thus be attributed to a temporary overload condition). We also plan to examine how the algorithm can be extended to consider different adaptation strategies [71], and to evaluate the performance of these strategies. Finally, a user survey is planned to assess the degree to which users are satisfied by the plans generated by the various adaptation algorithms; work in this direction will consider recent results in the area of exploiting social networks for software evaluations, as described in [72].

ABBREVIATIONS

ABBREVIATION	DESCRIPTION
UDDI	Universal Description, Discovery and Integration
WSDL	Web Service Description Language
SOAP	Simple Object Access Protocol
W3C	World-Wide Web Consortium
P2P	Peer to Peer
HTTP	HyperText Transfer Protocol
BPEL	Business Process Execution Language
WS-BPEL	Web Services - Business Process Execution Language
SSL	Secure Socket Layer
QoS	Quality of Service
SOA	Service Oriented Architecture
ASOB	Alternate Service Operation Binding
XML	Extensible Markup Language
API	Application Programming Interface
B2B	Business-to-Business
OWL	Web Ontology Language
RDF	Resource Description Framework

ABBREVIATION	DESCRIPTION
BPMN	Business Process Management Notation
WSMO	Web Service Modeling Ontology
CF	Collaborative Filtering
SLA	Service Level Agreement
RT	Response Time

REFERENCES

[1] OASIS WSBPEL TC. WS-BPEL 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

[2] MP. Papazoglou, P. Traverso, F. Leymann, “Service-Oriented Computing: State of the Art and Research Challenges”, IEEE Computer vol. 40, no 11, 2007, pp. 38-45.

[3] V. Cardellini, V. Di Valerio, V. Grassi, S. Iannucci, F. Lo Presti, “A Performance Comparison of QoS-Driven Service Selection Approaches”, Proceedings of ServiceWave 2011, Abramowicz W et al. (Eds.): LNCS 6994, 2011, pp. 167–178.

[4] C. Kareliotis, C. Vassilakis, S. Rouvas, P. Georgiadis. “QoS-Driven Adaptation of BPEL Scenario Execution”, Proceedings of ICWS 2009, E. Damiani, R. Chang, J. Zhang (eds), 2009, pp. 271-278.

[5] M. Claypool, A. Gokhale, Y. Miranda, P. Murnikov, D. Netes, M. Sartin, “Combining Content-Based and Collaborative Filters in an Online Newspaper”. SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation, I. Soboroff, C. Nicholas, M. Pazzani (eds), Berkeley, California, 1999,

[6] CL. Hwang, K. Yoon, “Multiple Criteria Decision Making”, Lecture Notes in Economics and Mathematical Systems, Springer-Verlag, 1981.

[7] J. Basilico, T. Hofmann, “Unifying collaborative and content-based filtering”. Proceedings of the twenty-first international conference on Machine learning (ICML 04), C.E. Brodley (ed), 2004, pp. 9-16.

[8] B.M. Kim, Q. Li, C.S. Park, S.G. Kim, J.Y. Kim. “A new approach for combining content-based and collaborative filters”. Journal of Intelligent Information Systems, July 2006, vol. 27, Issue 1, pp 79-91.

[9] J. Fürnkranz, Eyke Hüllermeier (eds), “Preference Learning”. Springer; 2011 edition (October 13, 2010), ISBN: 3642141242

[10] J. Aslam, M. Montague, “Models for metasearch”, Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR 2001, W.B. Croft, D.J. Harper, D.H. Kraft, J. Zobel (eds), 2001, pp. 276-284.

[11] M. Montague, J.A. Aslam, “Relevance Score Normalization for Metasearch”, Proceedings of the tenth international conference on Information and knowledge management, CIKM 2001, H. Paques, L. Liu, D. Grossman, C. Pu (eds), 2001, pp. 427-433.

[12] ISO. UNI EN ISO 8402 (Part of the ISO 9000 2002): Quality Vocabulary, 2002.

[13] ITU. Recommendation E.800 Quality of service and dependability vocabulary.

[14] Cardoso, J. Quality of Service and Semantic Composition of Workflows. PhD thesis, Univ. of Georgia, 2002.

[15] O'Sullivan, J., Edmond, D., Ter Hofstede, A. What is a Service?: Towards Accurate Description of Non-Functional Properties, Distributed and Parallel Databases, vol. 12, 2002.

[16] Canfora, G., Di Penta, M., Esposito, R., Villani, M.L. An Approach for QoS-aware Service Composition based on Genetic Algorithms. In Proceedings of the 2005 Conference on Genetic and evolutionary computation, pp. 1069-1075, 2005.

[17] Chothia, T., Kleijn, J. Q-Automata: Modelling the Resource Usage of Concurrent Components. Electronic Notes in Theoretical Computer Science 175 (2007) 153–167.

[18] Barbosa, L., Meng, S. QoS-aware Component Composition. In Proceedings of CISIS 2010, pp. 1008 – 1013.

[19] Apache ODE, Headers Handling. http://ode.apache.org/headers-handling.html

[20] GlassFish Community. http://glassfish.java.net/

[21] HSQLDB. http://hsqldb.org/

[22] US. Manikrao, TV. Prabhakar, "Dynamic Selection of Web Services with Recommendation System" Proceedings of the International Conference on Next Generation Web Services Practices, 2005, pp. 117-121.

[23] LB. Zeng, AHN. Benatallah, M. Dumas, J. Kalagnanam, H. Chang, "QoS-aware middleware for web services composition". IEEE Transactions on Software Engineering, vol. 30, no 5, 2004.

[24] O. Moser, F. Rosenberg, S. Dustdar, "Non-Intrusive Monitoring and Service Adaptation for WS-BPEL", Proceedings of WWW 2008, Beijing, China, , 2008, pp. 815-824.

[25] S. Ran, "A Model for Web Services Discovery With QoS", ACM SIGecom Exchanges, vol. 4(1), Spring, 2003, pp. 1-10.

[26] M. Paolucci, T. Kawamura, T. Payne, T. Sycara, "Semantic Matching of Web Services Capabilities", Proceedings of the International Semantic Web Conference, Sardinia, 2002, pp. 333-347.

[27] A. Bramantoro, S. Krishnaswamy, M, Indrawan, "A semantic distance measure for matching web services", Proceeding of the 2005 international conference on Web Information Systems Engineering, 2005, pp 217-226.

[28] J. Yu, Q. Sheng, J. Han, Y. Wu, C. Liu, “A semantically enhanced service repository for user-centric service discovery and management”, Data & Knowledge Engineering, vol. 72, Feb 2012, pp. 202-218.

[29] JB. Schafer, D. Frankowski, J. Herlocker, S. Sen, “Collaborative Filtering Recommender Systems”, in “The Adaptive Web”, Lecture Notes in Computer Science Volume 4321, 2007, pp 291-324.

[30] LR. Dice, “Measures of the Amount of Ecologic Association Between Species”, Ecology vol. 26, no 3, 1945, pp. 297–302, doi:10.2307/1932409

[31] V. Cross, X. Hu, “Using Semantic Similarity in Ontology Alignment”, Proceedings of the The Sixth International Workshop on Ontology Matching (collocated with the 10th International Semantic Web Conference ISWC-2011), 2011, pp. 61-72.

[32] G. Hirst, D. St-Onge, “Lexical Chains as Representations of Context for the Detection and Correction of Malapropisms”, chapter in “WordNet: An Electronic Lexical Database”, Christiane Fellbaum (ed), chapter 13, 1998, pp. 305-332, The MIT Press, Cambridge, MA.

[33] T. Seidl, HP. Kriegel, “Optimal multi-step k-nearest neighbor search”, Proceedings of the 1998 ACM SIGMOD international conference on Management of data, 1998, pp. 154-165.

[34] NB. Mabrouk, S. Beauche, E. Kuznetsova, N. Georgantas, V. Issarny, “QoS-aware Service Composition in Dynamic Service Oriented Environments”, Proceedings of Middleware 2009, LNCS vol. 5896, 2009, pp 123-142.

[35] Oracle, Manipulating SOAP Headers in BPEL. http://docs.oracle.com/cd/E14571_01/integration.1111/e10224/bp_manipdoc.htm#CIHFCBAD

[36] G. Castagna, N. Gesbert, L. Padovani, “A theory of contracts for Web services”, ACM Transactions on Programming Languages and Systems, vol. 31, issue 5, June 2009, article no 19.

[37] D. He, D. Wu, “Toward a Robust Data Fusion for Document Retrieval”, IEEE 4th International Conference on Natural Language Processing and Knowledge Engineering - NLP-KE, 2008.

[38] E. Al-Masri, Q.H. Mahmoud, “Discovering the best web service”, (poster) 16th International Conference on World Wide Web (WWW), P. Bouquet, H. Stoermer, G. Tummarello, H. Halpin (Eds.) 2007, pp. 1257-1258.

[39] E. Al-Masri, Q.H. Mahmoud, "QoS-based Discovery and Ranking of Web Services", IEEE 16th International Conference on Computer Communications and Networks (ICCCN), 2007, pp. 529-534.
[40] E. Al-Masri, Q.H. Mahmoud, “The QWS data set”, http://www.uoguelph.ca/~qmahmoud/qws/
[41] Z. Zheng, M.R. Lyu, “Collaborative Reliability Prediction for Service-Oriented Systems”, in Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering (ICSE2010), J. Kramer, J. Bishop, P. T. Devanbu, S. Uchitel (Eds.), Cape Town, South Africa, May 2-8, 2010, pp. 35 – 44.
[42] Z. Zheng, M.R. Lyu, “WS-DREAM: Web Service QoS Datasets”, http://www.wsdream.net/dataset.html
[43] J. Cardoso, A. Sheth, “Semantic e-Workflow Composition”, Journal of Intelligent Information Systems, vol. 21 no 3, pp. 191-225, 2003.
[44] P. Melville, RJ. Mooney, R. Nagarajan, “Content boosted collaborative filtering for improved recommendations”, Proceedings of the *Eighteenth National Conference on Artificial Intelligence, R. Dechter, R.S. Sutton (eds), Canada*, 2002, pp. 187-192.
[45] D. Margaris, C. Vassilakis, P. Georgiadis, “An integrated framework for QoS-based adaptation and exception resolution in WS-BPEL scenarios”, Proceedings of the 28th ACM Symposium on Applied Computing, S.Y. Shin, J.C. Maldonado (eds), Coimbra, Portugal, 2013, pp. 1900-1906.
[46] D. Margaris, C. Vassilakis, P. Georgiadis, “Adapting WS-BPEL scenario execution using collaborative filtering techniques”, Proceedings of the IEEE 7th International Conference on Research Challenges in Information Science, RCIS 2013, R. Wieringa, S. Nurcan, C. Rolland, J-L. Cavarero (eds), Paris, France, 2013.
[47] M. Alrifai, T. Risse, “Combining Global Optimization with Local Selection for Efficient QoS-aware Service Composition”, Proceedings of the 18th international conference on World wide web (WWW '09), Th. Karagiannis and M. Vojnovic (Eds.), 2009, pp. 881-890.
[48] F. Lécué, N. Mehandjiev, “Towards Scalability of Quality Driven SemanticWeb Service Composition”, Proceedings of the IEEE International Conference on Web Services (ICWS 2009), E. Damiani, R. Chang and J. Zhang (eds), 2009, pp. 469–476.
[49] M. Alrifai, T. Risse, “Efficient QoS-aware Service Composition”, In Emerging Web Services Technology Volume III, W. Binder, S. Dustdar (eds), Birkhäuser Basel, 2009.

[50] W. Mayer, R. Thiagarajan, M. Stumptner, “Service Composition as Generative Constraint Satisfaction”, IEEE International Conference on Web Services, 2009 (ICWS 2009), E. Damiani, R. Chang and J. Zhang (eds), 2009, pp. 888-895.
[51] L. Qi, X. Xia, J. Ni, Ch. Ma, Y. Luo, “A Decomposition-based Method for QoS-aware Web Service Composition with Large-scale Composition Structure”, Proceedings of the Fifth International Conferences on Advanced Service Computing, A. Koschel, J.L. Mauri (eds), 27 May - 1 June, 2013, Valencia, Spain, pp. 81-86.
[52] A.B. Hassine, S. Matsubara, T. Ishida, “A Constraint-Based Approach to Horizontal Web Service Composition”, Proceedings of the 5th International Semantic Web Conference, ISWC 2006, I.F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, L. Aroyo (Eds.) Athens, GA, USA, November 5-9, 2006, pp. 130-143.
[53] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. Lo Presti, R. Mirandola, “MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems”, IEEE Transactions on Software Engineering, vol. 38, no. 5, September/October 2012, pp. 1138-1159.
[54] T. Yu, K-J. Lin, “Service selection algorithms for Web services with end-to-end QoS constraints”, Proceedings of IEEE International Conference on e-Commerce Technology, 2004, pp. 129 - 136.
[55] J. Bisschop, “Linear Programming Tricks”, chapter in “AIMMS Optimization Modeling”, ISBN: 9781847539120. The chapter is available at http://www.aimms.com/aimms/download/manuals/aimms3om_linearprogrammingtricks.pdf
[56] StackOverflow, “Using min/max *within* an Integer Linear Program”, http://stackoverflow.com/questions/10792139/using-min-max-within-an-integer-linear-program
[57] IBM, “IBM ILOG CPLEX Optimizer”, 2013, http://www.ibm.com/software/commerce/optimization/cplex-optimizer/
[58] IBM, “Using CPLEX to examine alternate optimal solutions”, 2013, http://www.ibm.com/support/docview.wss?uid=swg21399929
[59] Y. Yanbe, A. Jatowt, S. Nakamura, and K. Tanaka, “Towards Improving Web Search by Utilizing Social Bookmarks”, In Web Engineering, LNCS vol. 4607, L. Baresi, P. Fraternali, G.J. Houben (eds), 2007, pp 343-357.
[60] TA. Sorensen, “A method of establishing groups of equal amplitude in plant sociology based on similarity of species content, and its application to analyses of the vegetation on Danish commons”, K dan Vidensk Selsk Biol Skr 5, 1948, pp. 1-34.

[61] LR. Dice, “Measures of the Amount of Ecologic Association Between Species”, Ecology vol. 26, no 3, 1945, pp. 297–302, doi:10.2307/1932409
[62] E. Hullermeier, M. Rifqi, S. Henzgen, R. Senge, “Comparing Fuzzy Partitions: A Generalization of the Rand Index and Related Measures”, IEEE Transactions on Fuzzy Systems, vol. 20, no 3, June 2012, pp. 546-556.
[63] Glassfish Community, “Metro”, 2013, https://metro.java.net/
[64] R. Salakhutdinov, A. Mnih, G. Hinton, “Restricted Boltzmann machines for collaborative filtering”, Proceedings of the 24th international conference on Machine learning (ICML 07), Z. Ghahramani (ed), 2007, pp. 791-798.
[65] R. Beier, B. Vocking, “Typical Properties of Winners and Losers in Discrete Optimization”, Proceedings of the 36th ACM Symposium on Theory of Computing (STOC 2004), June 13–15, 2004, Chicago, Illinois, USA.
[66] Shao, L., Guo, Y., Chen, X., He, Y.: Pattern-Discovery-Based Response Time Prediction. In: Advances in Automation and Robotics, vol. 2 LNEE, vol. 123, 355-362 (2012)
[67] Margaris, D., Vassilakis, C., Georgiadis, P.: Combining Quality of Service-based and Collaborative filtering-based techniques for BPEL scenario execution adaptation. University of Peloponnese SDBS Technical report TR-14002, available at http://sdbs.dit.uop.gr/?q=TR-14002
[68] Duan, Y., Huang, Y.: Research on availability prediction model of web service. In: 2011 International Conference on Computer Science and Service System, 1590–1594 (2011)
[69] Saric, A., Hadzikadic, M., Wilson, D: Alternative Formulas for Rating Prediction Using Collaborative Filtering. In: Proceedings of the 18th International Symposium on Foundations of Intelligent Systems, 301-310 (2009)
[70] Chelminski, P., Coulter, R.: An examination of consumer advocacy and complaining behavior in the context of service failure. In: Journal of services marketing, vol. 25(5), 361–370 (2011)
[71] Zeginis, C., Plexousakis, D.: Web Service Adaptation: State of the art and Research Challenges. Institute of Computer Science, FORTH-ICS, Tech. Rep. 410, ICS-FORTH (2010)
[72] R. Ali, C. Solis, I. Omoronyia, M. Salehie, B. Nuseibeh, “Social Adaptation: When Software Gives Users a Voice”, Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012), J. Filipe, L.A. Maciaszek (eds), 2012, pp. 75-84.

[73] Kareliotis Christos, Costas Vassilakis, Efstathios Rouvas, Panayiotis Georgiadis, “QoS-aware Exception Resolution for BPEL Processes: A Middleware-based Framework and Performance Evaluation”, International Journal on Web and Grid Services (IJWGS), 2009 - Vol. 5, No.3 pp. 284 - 320

[74] Kareliotis Christos, “BPEL Scenario Execution: Dynamic adaptation and exception resolution”, Doctoral Dissertation, NKUA, 2010

[75] Kareliotis, C., Vassilakis, C., Rouvas, E., Georgiadis P.: IQoS-aware exception resolution for BPEL processes: a middleware-based framework and performance evaluation. *IJWGS* 5(3), pp. 284-320, 2009

[76] D. Margaris, C. Vassilakis, P. Georgiadis, “An integrated framework for adapting WS-BPEL scenario execution using QoS and collaborative filtering techniques”, Science of Computer Programming, Volume 98, Part 4, 1 February 2015, Pages 707–734, http://www.sciencedirect.com/science/article/pii/S0167642314004778

[77] Arpacı, A.E., Bener, A.B.: Agent Based Dynamic Execution of BPEL documents. In: ISCIS 2005, LNCS 3733, P. Yolum, et al. (eds.), 332 – 341 (2005)

[78] C. Kareliotis, C. Vassilakis, P. Georgiadis, “Enhancing BPEL scenarios with dynamic relevance-based exception handling”, Proceedings of ICWS07, Salt Lake City, Utah, USA, 9–13 July 2013, pp.751–758.

[79] Xia, Y., Chen, P., Bao, L., Wang, M., Yang, J. A QoS-Aware Web Service Selection Algorithm Based on Clustering. *In Proceedings of ICWS11*, 2011.

[80] Comerio, M., De Paoli, F., Grega, S., Maurino A., Batini, C. WSMoD: A Methodology for QoS-Based Web Services Design. *Web services research*, Volume 4, Issue 2, 2007

[81] JL. Herlocker, JA. Konstan, LG. Terveen, JT. Riedl, “Evaluating collaborative filtering recommender systems”, ACM Transactions on Information Systems vol. 22, no 1, January 2004, pp. 5-53.

[82] M. Balabanovic, Y. Shoham. “Fab: content-based, collaborative recommendation”, Communications of the ACM, vol. 40, issue 3, 1997, pp 66-72.

[83] MJ. Pazzani, “A Framework for Collaborative, Content-Based and Demographic Filtering”, Artificial Intelligence Review, vol. 13, issue 5-6, December 1999, pp 393-408.

[84] D. Margaris, C. Vassilakis, P. Georgiadis, “A hybrid framework for WS-BPEL scenario execution adaptation, using monitoring and feedback data”, to appear in the 30th ACM Symposium on Applied Computing, Salamanca, Spain, 2015.