Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	«Σχεδίαση Δυναμικών Επιχειρηματικών Διαδιακασιών»
Ονοματεπώνυμο Φοιτητή	Βλαχάκης Γεώργιος
Αριθμός Μητρώου	ΜΠΣΠ/08029
Κατεύθυνση	Συστήματα Υποστήριξης Αποφάσεων
Επιβλέπων	Δημήτρης Αποστόλου, Λέκτορας

Ημερομηνία Παράδοσης Απρίλιος 2010

Τριμελής Εξεταστική Επιτροπή

Δημήτριος Αποστόλου Λέκτορας	Θεμιστοκλής Παναγιωτόπουλος Καθηγητής	Χαράλαμπος Κωνσταντόπουλος Λέκτορας

ΠΕΡΙΕΧΟΜΕΝΑ

Κατάλογος εικόνων

Abstract

Bpel as part of Service Oriented Architecture is a XML based language used to define enterprise business process within Web Services with a standardized way .Processes written in BPEL can orchestrate interactions between Web Services using partnerLinks which define the interface of each service to be used. Although there are cases where there is need to choose dynamically which process to be used providing all the information of the partnerLink at the runtime. This feature provides to every enterprise using Bpel great modularity and creates an adaptive environment in their business process.
In this document we give a simple example of how we can create dynamic partnerLinks and use them in a traditional Bpel process.

1. Εισαγωγή

Οι επιχειρηματικές διαδικασίες στις μέρες μας έχουν λάβει κυρίαρχη θέση στην παγκόσμια οικονομία. Οι σύγχρονες επιχειρήσεις στα πλαίσια μιας πολύπλοκής , ευμετάβλητης και ανταγωνιστικής οικονομίας καλούνται να ακολουθήσουν τα ταχέως μεταβαλλόμενο περιβάλλον της παγκόσμια αγοράς και να προσαρμοστούν σ'αυτό.

Η ικανότητα των επιχειρήσεων να μπορούν γρήγορα να προσαρμόζονται στις εξελίξεις απαιτεί την ύπαρξη επιχειρηματικών διαδικασιών οι οποίες θα μπορούν με μικρές αλλαγές στην δομή τους να είναι σε θέση να ενθυλακώσουν νέες τροποποιήσεις και να δημιουργήσουν για την επιχείρηση τις απαραίτητες οικονομίες κλίμακας.

Ο ρόλος των ΠΣ στο σύγχρονο περιβάλλον της παγκόσμιας οικονομίας και του διεθνή ανταγωνισμού είναι να ανταποκριθούν στις ανάγκες των επιχειρήσεων για προσαρμοστικότητα , ευελιξία , έγκαιρή αντίδραση στις εξελίξεις , για επαναχρησιμοποίηση των μονάδων λειτουργικότητας και αποδέσμευση της επιχειρηματικής λογικής από την φυσική σχεδίαση και την τεχνική υποδομή. Επιπλέον των παραπάνω ίσως το μεγαλύτερο στοίχημα που καλούνται να κερδίσουν τα Π.Σ είναι η πλήρη διαλειτουργικότητα τους και η ενοποίηση παλαιών ετερογενούς χαρακτήρα αυτοτελών συστημάτων.

Η επιστήμη της πληροφορικής για να καλύψει τις προαναφερόμενες ανάγκες δημιούργησε μια αρχιτεκτονική προσανατολισμένη στις υπηρεσίες (Service Oriented Architecture-SOA) , η οποία παρέχει λύσεις στα σύγχρονα προβλήματα του επιχειρηματικού κόσμου.

Στα πλαίσια της προσανατολισμένης στις υπηρεσίες αρχιτεκτονικής κινείται και η παρούσα διπλωματική εργασία και ειδικότερα στην παρουσίαση του τρόπου σχεδίασης ευέλικτων και δυναμικών επιχειρηματικών διαδικασιών , δημιουργώντας πρότυπες διαδικτυακές υπηρεσίες (Web Services) που επιτρέπουν την επαναχρησιμοποίηση τους.

Η έννοια των δυναμικών ροών εργασίας μπορεί να υλοποιηθεί με δύο τρόπους :

- Με την δημιουργία μια ολοκληρωμένης επιχειρηματικής διαδικασίας , η οποία μπορεί να υλοποιηθεί εξολοκλήρου «κτίζοντας» την κατά την διάρκεια κατά την οποία καλείται από επιμέρους διαδικασίες οι οποίες βρίσκονται αποθηκευμένες σε μια βάση δεδομένων.
- Με την δημιουργία μιας αφαιρετικής, βασικής διαδικασίας η οποία για την υλοποίηση της θα απαιτεί την κλήση επιμέρους διαδικασιών δυναμικά. Η δυναμική κλήση των επιμέρους διαδικασιών θα μπορεί να γίνεται μέσω μιας βασικής υποδιαδικασίας η οποία θα αποτελεί ουσιαστικά το σημείο κλήσης των διαδικασιών που θα απαιτούνται κάθε φορά αναλόγως της ροής της διαδικασίας κατά την διάρκεια της κλήσης και της εφαρμογής της.

Συμπερασματικά λοιπόν μπορούν να οριστούν δύο διαφορετικοί τρόποι για την δημιουργία δυναμικών ροών εργασίας, ο πρώτος υλοποιείται ουσιαστικά πριν την διαδικασία κλήσης της χωρίς να είναι γνωστό το τελικό σχήμα της διαδικασίας ενώ ο δεύτερος υλοποιείται ουσιαστικά αφαιρετικά αλλά γνωρίζοντας το γενικότερο σχήμα της διαδικασίας καλώντας επιμέρους

διαδικασίας οι οποίες έχουν προστεθεί στην βασική διαδικασία μέσω της βασικής υποδιαδικασίας αλλά αγνοώντας ποια τελικά θα κληθεί κατά την διάρκεια που θα κληθεί η διαδικασία.

1.1 Σκοπός

Η συγκεκριμένη διπλωματική εργασίας έχει σαν σκοπό την ανάπτυξη μιας εφαρμογής βασισμένης στο δεύτερο τρόπο δυναμικής σχεδίασης, η οποία θα περιγράφει μια επιχειρηματική διαδικασία παραγγελιοληψίας στην οποία οι απαραίτητες υπηρεσίες θα καλούνται τόσο με τον παραδοσιακό τρόπο όσο και με δυναμικό , μέσω μια πρότυπης υπηρεσίας η οποία θα παίξει τον ρόλο του συνδετικού κρίκου ανάμεσα τους και της βασικής επιχειρηματικής ροής δεδομένων.

1.2 Περιγραφή της εργασίας

Αρχικά θα γίνει μια σύντομη αναφορά στην SOA ενώ τα πρότυπα WSDL και BPEL θα αναπτυχθούν περαιτέρω στα Παραρτήματα Α και Β αντίστοιχα. Στην συνέχεια θα ακολουθήσει η απαραίτητη τεχνική περιγραφή με την παρουσίαση του συνόλου του λογισμικού που χρειάστηκε για την ολοκλήρωση του έργου ,ενώ ακολούθως θα υπάρχει μια περιγραφή του λογικού σχεδιασμού όπου θα παρουσιαστούν αναλυτικά τα επιμέρους στοιχεία που συνθέτουν τη συγκεκριμένη επιχειρηματική διαδικασία. Στο τρίτο μέρος της διπλωματικής εργασίας θα γίνει μια αξιολόγηση του σχετικού τρόπου ανάπτυξης επιχειρηματικών διαδικασιών, ενώ παράλληλα θα γίνει μια αναφορά στις μελλοντικές εξελίξεις και θα ακολουθήσει ο επίλογος.

2. Αρχιτεκτονική Προσανατολισμένη στις Υπηρεσίες(SOA)

Ο λόγος δημιουργίας της SOA βρίσκεται στην ανάγκη να έρθουν σε επαφή και να επικοινωνήσουν ετερογενείς εφαρμογές οι οποίες βρίσκονται διάσπαρτες στο Web με τη χρήση μια κοινής πλατφόρμας περιγραφής σε όρους πρωτοκόλλων και λειτουργικότητας.
Η λογική του προσανατολισμού προς τις υπηρεσίες απαιτεί χαλαρή σύνδεση των υπηρεσιών με τα λειτουργικά συστήματα και τις άλλες τεχνολογίες που στηρίζουν τις εφαρμογές. Βάση αυτής τις κεντρικής ιδέας οι συναρτήσεις μια γλώσσας προγραμματισμού που περιγράφουν μια διαδικασία χωρίζονται σε ξεχωριστές μονάδες ή υπηρεσίες οι οποίες γίνονται διαθέσιμες στο διαδίκτυο επιτρέποντας σε κάθε χρήστη την επαναχρησιμοποίηση τους στην παραγωγή νέων εφαρμογών.

2.1 Ανάλυση

Σε κάθε SOA υπάρχουν τρεις ρόλοι –σχέσεις που καθορίζουν τον τρόπο ενέργειας του κάθε εμπλεκόμενου μέρους.
Οι ρόλοι αυτοί είναι :

- Ο χρήστης της υπηρεσίας .
- Ο πάροχος της υπηρεσίας .
- Ο κατάλογος με τις παρεχόμενες υπηρεσίες.

Αναλυτικότερα κάθε τρόπος λειτουργίας ενεργεί ως εξής.

Ο χρήστης της υπηρεσίας είναι αυτός ο οποίος αναζητά και εντοπίζει την περιγραφή μια υπηρεσίας η οποία βρίσκεται δημοσιευμένη σε έναν ή περισσότερους καταλόγους και αποσκοπεί στην χρησιμοποίηση της ώστε να είναι σε θέση να καλέσει την σχετική υπηρεσία .
Ο πάροχος είναι υπεύθυνος για την δημιουργία και την δημοσιοποίηση της περιγραφής μια υπηρεσίας σε ένα ή περισσότερους καταλόγους υπηρεσιών.
Ο κατάλογος υπηρεσιών υπάρχει για την δημοσιοποίηση των περιγραφών υπηρεσιών και επιτρέπει την αναζήτηση από τον χρήστη οποιαδήποτε δημοσιευμένης σ'αυτόν περιγραφής. Ουσιαστικά ο ρόλος του καταλόγου είναι να φέρνει σε επαφή τον χρήστη και τον πάροχο ενεργώντας ως ενδιάμεσος.

2.2 **Βασικά Χαρακτηριστικά** SOA

Επιπλέον των παραπάνω ρόλων στην SOA υπάρχουν και τα παρακάτω βασικά στοιχεία που συνθέτουν την δομή της:

- Services
- Περιγραφή Services
- Δημοσιοποίηση και Αναζήτηση υπηρεσιών
- Προσδιορισμός συσχετιζόμενων μοντέλων δεδομένων

Ουσιαστικά η SOA αποτελείται από υπηρεσίες (Services) οι οποίες αλληλεπιδρούν μεταξύ τους σε ένα αποκεντρωτικό μοντέλο .Σημαντικότατο ρόλο στην SOA είναι η περιγραφή των υπηρεσιών με ένα κοινά αποδεκτό τρόπο με τη χρήση του πρωτοτύπου WSDL για παράδειγμα, ή οποιουδήποτε άλλου προτύπου περιγραφής Services, βάση τις οποίας θα περιγράφεται το τι ακριβώς καλείται η υπηρεσία να κάνει , πως μπορεί κανείς να συνδεθεί μαζί της (με ποιο πρωτόκολλό επικοινωνίας) καθώς και το ποια πρωτόκολλά ασφαλείας χρειάζονται ώστε να χρησιμοποιηθεί.
Η δημοσιοποίηση αφορά την διάθεση της περιγραφής της υπηρεσίας στο κοινό για κάθε ενδεχόμενη χρήση. Υπάρχουν δύο μέθοδοι με τις οποίες μπορεί να γίνει η δημοσιοποίηση :

- Η μέθοδος Pull
- Και η μέθοδος Push.

Στην Pull μέθοδο , ο υποψήφιος χρήστης ζητά την περιγραφή της υπηρεσίας από τον πάροχο ενώ αντίθετα στην Push μέθοδο , ο πάροχος στέλνει την περιγραφή της υπηρεσίας του στον υποψήφιο χρήστη .

Η μέθοδος Push μπορεί να χρησιμοποιήσει τους γνωστούς τρόπους μετάδοσης δεδομένων μέσω δικτύου ήτοι :

- Multicast
- Unicast
- Broadcast
- Anycast

Γενικά οι δύο αυτοί μέθοδοι χρησιμοποιούνται σε από κοινού για την δημιουργία των κατάλληλων συνθηκών δημοσιοποίησης της υπηρεσίας μέσω ενός καταλόγου στο οποίο θα γίνει η αναζήτηση.

Τέλος για τον προσδιορισμό των συσχετιζόμενων μοντέλων τα οποία ανταλλάσουν πληροφορία μέσω παραμέτρων χρησιμοποιείται είτε η WSDL όπως προαναφέρθηκε είτε η ebXML.

2.3 **Βασικά Χαρακτηριστικά** SOA

Τα πλεονεκτήματα της χρησιμοποίησης της τεχνολογίας SOA είναι κυρίως για τις ίδιες τις επιχειρήσεις αφού για αυτές αναπτύχθηκε η συγκεκριμένη τεχνολογία μερικά από τα οποία συνοψίζονται όπως παρακάτω:

- Η SOA μπορεί να μειώσει τα έξοδα της επιχείρησης μειώνοντας τον χρόνο παρουσίασης ενός προϊόντος ή υπηρεσίας στην αγορά και προσελκύοντας νέους πελάτες παρουσιάζοντας στο ευρύ κοινό τις δυνατότητες και το σύνολο του έργου της επιχείρησης.
- Βελτιώνεται η αποδοτικότητα της επιχείρησης , μειώνοντας το χρόνο ανταπόκρισης της επιχείρησης στις αλλαγές , επιτρέποντας την επαναχρησιμοποίηση των ήδη υπαρχόντων υποδομών σε ΠΣ βελτιώνοντας την συνεργασία μεταξύ των τμημάτων της επιχείρησης και την συνεργασία με εξωτερικούς φορείς.
- Μειώνει τα κόστη του IT τμήματος , του R&D τμήματος και της Διοίκησης γενικότερα αφού δεν θα πρέπει να επανασχεδιαστούν νέες διαδικασίες από την αρχή ώστε να καλυφθούν νέες ανάγκες , ενώ αν χρειαστεί να γίνει αυτό η ανάπτυξη γίνεται πολύ ευκολότερα απ' ότι στο παρελθόν.

3. **Δημιουργία της Διαδικασίας**

3.1 **Τεχνική Περιγραφή**

Για την παρουσίαση του τρόπου δημιουργίας δυναμικών ροών εργασίας επιλέχθηκαν και εγκαταστάθηκαν οι παρακάτω εφαρμογές:

- Oracle SOA Suite

- Oracle JDeveloper
- Oracle DataBase 10g

3.1.1 Oracle SOA Suite

Η έκδοση που επιλέχθηκε για εγκατάσταση είναι η 10.1.3.1 μαζί με τον αντίστοιχο Application Server.

Αναλυτικά στο παραπάνω περιβάλλον υπάρχουν τα εξής προϊόντα:

- Oracle Process Manager
- Oracle Web Service Manager
- Oracle BPEL Process Manager
- Oracle Enterprise Service Bus

Η SOA Suite της Oracle είναι ένα ολοκληρωμένο σύνολο εφαρμογών για την δόμηση , διαχείριση και εκτέλεση SOA εφαρμογών , επιτρέποντας την δημιουργία , έλεγχο και σύνθεση υπηρεσιών σε εφαρμογές και επιχειρηματικές διαδικασίες.

Επίσης το συγκεκριμένο προϊόν είναι hot-plugged , δηλαδή εγκαθίσταται στην ήδη υπάρχουσα υποδομή χωρίς καμία άλλη απαίτηση.

Η επιλογή της συγκεκριμένης πλατφόρμας έγινε λόγω της πληρότητας που παρέχει σε SOA ανάπτυξη καθώς και στην εξοικείωση του γράφοντα σε προϊόντα Oracle .

Η εγκατάσταση γίνεται αφού κατεβάσουμε το προϊόν από την διεύθυνση http://www.oracle.com/technology/software/products/ias/htdocs/101310.html

και τρέξουμε το Setup.exe. Θα μας ζητηθεί να δηλώσουμε ένα Password για το λογαριασμό του Administrator (το Username του οποίου είναι το oc4jadmin) καθώς και ένα όνομα για το instance.

Αφού γίνει επιτυχώς η εγκατάσταση πηγαίνουμε στην διαδρομή Programs->Oracle-(Instance Name) -> Start Soa Suite . Κατόπιν στην σελίδα http://localhost:8888/BPELConsole μπορούμε να διαχειριστούμε οποιοδήποτε διαδικασία θα δημιουργήσουμε αργότερα.

Η επιλογή της συγκεκριμένης πλατφόρμας έγινε λόγω της πληρότητας που παρέχει σε SOA περιβάλλοντα ανάπτυξης καθώς και στην εξοικείωση του γράφοντα με το περιβάλλον και τα προϊόντα Oracle. Επίσης το πλεονέκτημα του συγκεκριμένου προϊόντος είναι ότι είναι hot-pluggable ,δηλαδή εγκαθίσταται στην ήδη υπάρχουσα υποδομή χωρίς καμιά άλλη απαίτηση.

3.1.2 Oracle DataBase 10g Enterpraise Edition

Το συγκεκριμένο προϊόν της εταιρείας Oracle παρέχει ένα επαρκές ,αξιόπιστο και ασφαλές περιβάλλον διαχείρισης δεδομένων για επιχειρησιακές εφαρμογές. Με την συγκεκριμένη έκδοση παρέχονται εργαλεία και λειτουργίες διαχείρισης μέσω της κονσόλας Enterprise Manager ενώ τα

πλεονεκτήματα της συγκεκριμένης ΒΔ είναι ότι είναι ανεξάρτητη του μεγέθους ή του αριθμού των επεξεργαστών , ενώ υπάρχει διαθέσιμη για κάθε γνωστό λειτουργικό σύστημα.

Η εγκατάσταση γίνεται αφού κατεβάσουμε το προϊόν από την σελίδα http://www.oracle.com/technology/software/products/database/index.html και εκτελέσουμε το Setup.exe. Σε κάποιο στάδιο της εγκατάστασης θα μας ζητηθεί η διαδρομή εγκατάστασης της ΒΔ καθώς και το όνομα του Instance και ένα password για το λογαριασμό του SYS και του SYSTEM.

3.1.3 Oracle JDeveloper

Ο Oracle JDeveloper είναι ένα IDE (Intergraded Development Environment) για πλατφόρμες Java υποστηρίζοντας Java EE και Java SE. Το προϊόν περιλαμβάνει το Oracle Application Development Framework (ADF) , ένα Model View Controller περιβάλλον για Java EE ανάπτυξη ,έχοντας χαρακτηριστικά που διευκολύνουν την συγγραφή κώδικα. Επίσης πλέον των άλλων χαρακτηριστικών του προϊόντος είναι κατάλληλό για την ανάπτυξη εφαρμογών Web Services , Service Oriented Architecture Business Process Management και Mobile Services.

Η εγκατάσταση γίνεται αφού κατεβάσουμε το προϊόν από την σελίδα http://www.oracle.com/technology/software/products/jdev/archives.html και εκτελέσουμε το Setup.exe.Θα πρέπει να δοθεί μεγάλη σημασία στο ποια έκδοση θα εγκατασταθεί διότι υπάρχει περίπτωση να μην είναι συνεργάσιμη με την παραπάνω έκδοση της Soa Suite. Στην συγκεκριμένη περίπτωση χρησιμοποιήθηκε η 10.1.3.4.

3.2 **Λογική Σχεδίαση**

Στο σενάριο που θα παρουσιαστεί ο δυναμικός τρόπος σχεδίασης επιχειρηματικών διαδικασιών εμφανίζεται μια τυπική διαδικασία παραγγελιοληψίας ενός υλικού από ένα πελάτη. Θα πρέπει να σημειωθεί ότι ο σκοπός της παρούσας διπλωματικής εργασίας δεν είναι η παρουσίαση της σχεδίασης μια υποδειγματικής διαδικασίας παραγγελιοληψίας αλλά να δώσει στον αναγνώστη την δυνατότητα να κατανοήσει και να πραγματοποιήσει αργότερα δυναμικές σχεδίασης διαδικασιών. Για λόγους συντομίας ο συνολικός σχεδιασμός της συγκεκριμένης διαδικασίας που θα παρουσιαστή έχει λάβει δευτερεύουσα σημασία .

3.2.1 **Δυναμικές Διαδικασίες**

Οι δυναμικές διαδικασίες όπως προαναφέρθηκε και στην εισαγωγή της διπλωματικής εργασίας μπορούν να υλοποιηθούν με δύο τρόπους οι οποίοι καθορίζουν και χρησιμοποιούνται ανάλογα το μέγεθος της ευελιξίας και της δυναμικότητας που απαιτούμε από τις διαδικασίες μας αλλά και από το γενικότερο επιχειρησιακό περιβάλλον για το οποίο θα κληθούν να υλοποιηθούν.

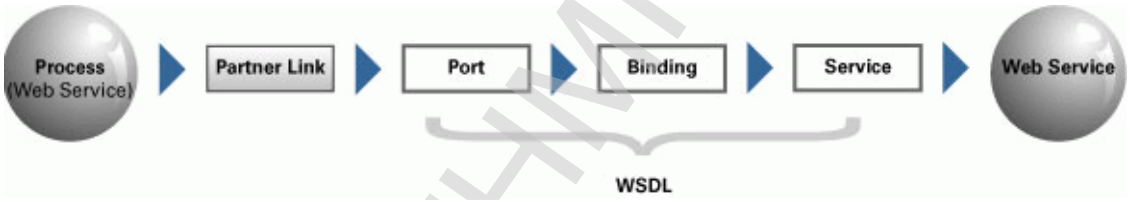
Οι μέθοδοι υλοποιήσεις δυναμικών διαδικασιών είναι :

- Η δημιουργία σύνθετων διαδικασιών από απλούστερες διαδικασίες, οι οποίες βρίσκονται αποθηκευμένες σε μια βάση δεδομένων κατά το στάδιο που το process καλείται να υλοποιηθεί (runtime).
- Η δυναμική κλήση διαδικασιών μέσω μιας «τυπικής» διαδικασίας κατά το στάδιο που καλείται η διαδικασία να υλοποιηθεί, αναλόγως των δεδομένων εισαγωγής ή εξέλιξης της διαδικασίας, χωρίς οι διαδικασίες που καλούνται να φαίνονται στην σχεδίαση του συνολικού process.

Στην παρούσα εργασία παρουσιάζεται ο δεύτερος τόπος δυναμικού σχεδιασμού bpel process μέσα από το παράδειγμα που προαναφέρθηκε.

Η υλοποίηση μια δυναμικής σχεδίασης ενός bpel process στηρίζεται ουσιαστικά στο dynamic binding των Web Service που καλούνται. Για την κλήση άλλων υπηρεσιών η Bpel χρησιμοποιεί τα Partner links.

Μέσα σε ένα Partner link βρίσκονται όλες οι πληροφορίες που χρειάζονται για να υλοποιηθεί μια κλήση ενός WSDL ορισμένου Service μέσω του στοιχείου portTypes.



Εικόνα 1: Κλήση ενός Web Service από Bpel Process.

Σε μια τυπική Bpel Process οι πληροφορίες που περιέχει το Partner Link και χρησιμοποιούνται για την κλήση του Web Service ορίζονται κατά την σχεδίαση του process, ωστόσο όλες αυτές οι πληροφορίες είτε μπορεί να μην είναι γνωστές κατά το στάδιο αυτό είτε ο σχεδιαστής για να θέλει να τις αποκρύψει για σχεδιαστικούς λόγους.

Η δυναμική σχεδίαση ενός Bpel Process επιτρέπει να οριστούν οι πληροφορίες που απαιτούνται για την υλοποίηση του binding με το Web Service ακριβώς την στιγμή που το Bpel Process βρίσκεται σε εξέλιξη.

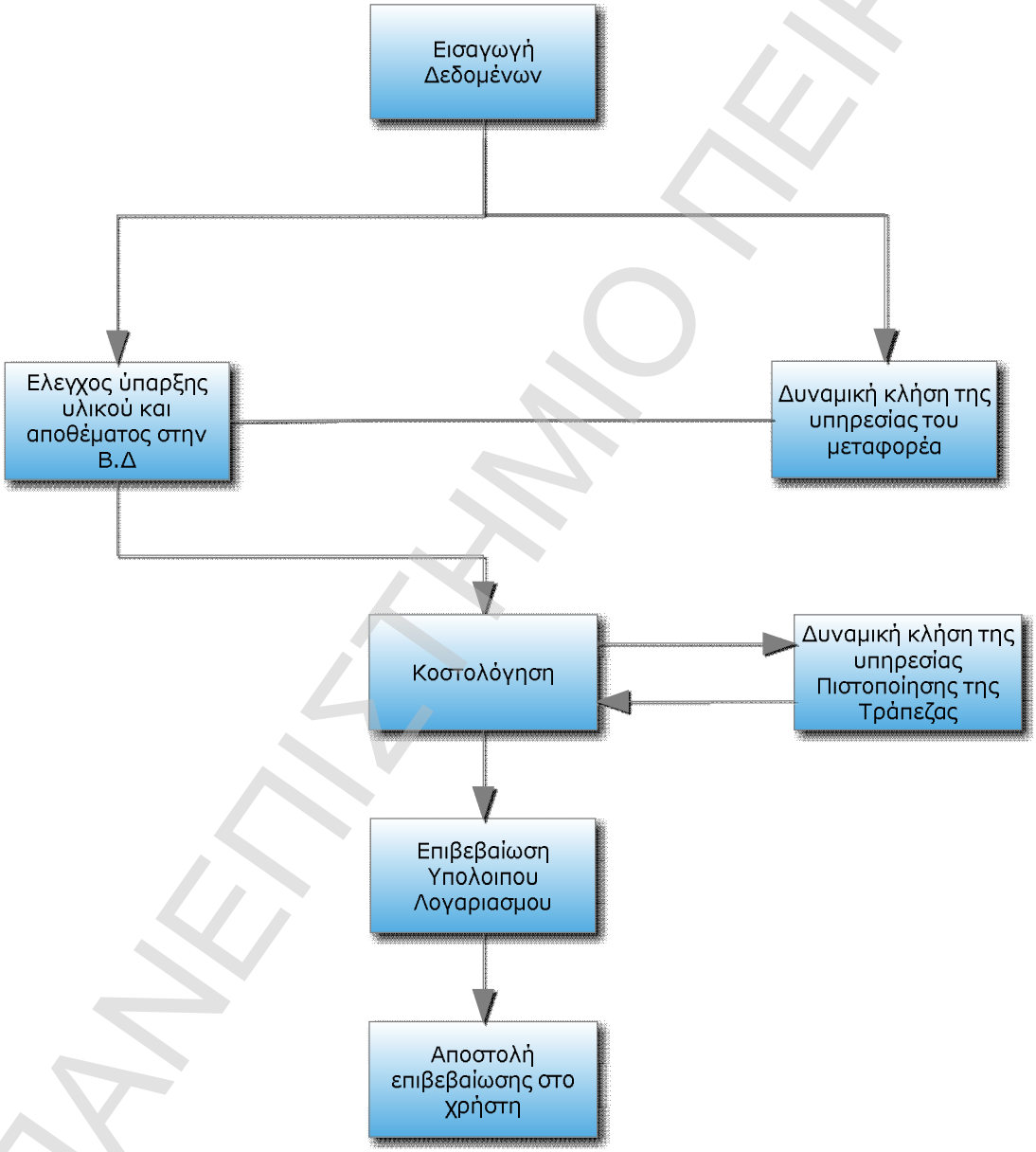
3.2.2 Παρουσίαση της Διαδικασίας

Στην διαδικασία της παραγγελιοληψίας υποθέτουμε ένα πελάτη ο οποίος επιθυμεί να προμηθευτεί ένα υλικό μέσω Internet από την σελίδα κάποιου online καταστήματος. Τα απαραίτητα στοιχεία τα οποία ζητούνται από τον πελάτη είναι τα εξής:

- Αριθμός κατασκευαστή υλικού (Part Number)

- Αιτούμενη ποσότητα.
- Email πελάτη.
- Τόπος παράδοσης
- Ημέρες Επιθυμητής παράδοσης.
- Αριθμός Πιστωτικής κάρτας πελάτη.
- Εκδίδουσα τράπεζα της πιστωτικής κάρτας.

Στο σχεδιάγραμμα που ακολουθεί φαίνονται σχηματικά τα στάδια που πρόκειται να ακολουθήσει η διαδικασία.



Εικόνα 2: Η Διαδικασία Παραγγελιοληψίας

Ο πελάτης καταχωρεί τα ζητούμενα στοιχεία στην φόρμα εισαγωγής του προμηθευτή .Στην συνέχεια καλούνται παράλληλα η υπηρεσία (Web Services) του ίδιου του προμηθευτή και η υπηρεσία του μεταφορέα. Το Web Service του προμηθευτή αναζήτα το αιτούμενο προϊόν σε μια ΒΔ ελέγχει την ύπαρξη ή μη του προϊόντος και στην συνέχεια την επάρκεια της υπάρχουσας ποσότητας σε σχέση με την αιτηθείσα. Σε περίπτωση θετική αναζήτησης (ύπαρξη προϊόντος και επάρκεια ποσότητας) επιστρέφεται στην κύρια διαδικασία μια τιμή η οποία εκπροσωπεί την αξία του προϊόντος χωρίς τα μεταφορικά έξοδα.

Η παράλληλη κλήση της υπηρεσίας του μεταφορέα έχει σαν σκοπό να επιλέξει ανάμεσα σε δύο υπάρχουσες μεταφορικές εταιρείες την πλέον συμφέρουσα από άποψη τιμής δίνοντας σαν δεδομένα εισόδου το τόπο παράδοσης και τις επιθυμητές μέρες παράδοσης. Η κάθε μεταφορική εταιρεία που εξετάζεται έχει διαφορετική τιμολογιακή πολιτική ανάλογα το τόπο παράδοσης (Ήπειρο) και το χρόνο παράδοσης.

Το Web Service επιστρέφει όπως προαναφέρθηκε τον οικονομικότερο μεταφορέα δίνοντας το κόστος αυτού στην κύρια διαδικασία.

Τελικά κατόπιν ολοκλήρωσης των παραπάνω διαδικασιών καλείται η υπηρεσία που ασχολείται με την πιστοποίηση του αριθμού της πιστωτικής κάρτας του πελάτη και την επάρκεια του προς χρέωση τελικού ποσού της παραγγελίας. Το Web Service αυτό με την λήψη του στοιχείου της ονομασίας της Τράπεζας βρίσκει την ανάλογη υπηρεσία, την καλεί και ελέγχει την ορθότητα και την πληρότητα των προς εξέταση στοιχείων επιστρέφοντας μια Boolean μεταβλητή στην κύρια διαδικασία ή οποία με την λήψη της μεταβλητής αυτή ολοκληρώνεται και επιστρέφει το ανάλογο μήνυμα στον πελάτη.

Στο μήνυμα αυτό φαίνεται το αρχικό κόστος της παραγγελίας χωρίς τα μεταφορικά έξοδα καθώς και το κόστος μαζί με τα μεταφορικά έξοδα.

Στην συνέχεια θα ακολουθήσει μια αναλυτική παρουσίαση της κάθε υπηρεσίας και θα δοθεί ιδιαίτερη βαρύτητα στον τρόπο δημιουργίας της δυναμικής υπηρεσίας πιστοποίησης της πιστωτικής κάρτας και στα πλεονεκτήματα που παρουσιάζει σε σχέση με τον συμβατικό τρόπο ανάπτυξης επιχειρηματικών διαδικασιών.

4. Υλοποίηση της Διαδικασίας

4.1 MaterialService

Η υπηρεσία αυτή είναι ένα Web Service το περιγράφει την διαδικασία αναζήτησης ενός υλικού στην ΒΔ του προμηθευτή και στην επιστροφή στην κύρια διαδικασία των απαραίτητων πληροφοριών για την συνέχιση της παραγγελιοληψίας. Η ανάπτυξη του συγκεκριμένου Web Service έγινε με την χρήση PL/SQL κώδικα. Το package body συντάθηκε και στην συνέχεια μετατράπηκε σε υπηρεσία είναι το παρακάτω:

```
PACKAGE           "SERVICE_YLIKA"    AS
FUNCTION get_apothema(v_nsn IN ylika.yliko_nsn%type,
              v_posothta in ylika.apothema%type)
 RETURN number;
END service_ylika;

PACKAGE BODY          "SERVICE_YLIKA"  as
  function get_apothema
    ( v_nsn in   ylika.yliko_nsn%type,
      v_posothta ylika.apothema%type )
      return number
        is
v_timh ylika.timh%type;
v_timh_total ylika.timh%type;
v_apothema ylika.apothema%type;
        begin

select apothema into v_apothema
from ylika
where yliko_nsn=v_nsn;

select timh into v_timh
from ylika
where yliko_nsn=v_nsn;

if v_apothema>v_posothta then
v_timh_total:=v_timh*v_posothta;

else v_timh_total:=2;
end if;
return v_timh_total;
        end;
  end;
```

Εικόνα 3: Το Package Service_Ylika

Το Package αυτό έχει μια function η οποία ονομάζεται get_apothema με μεταβλητές εισόδου :

- V_nsn , η μεταβλητή αυτή είναι ίδιου τύπου με το αντίστοιχο πεδίο στον πίνακα της ΒΔ που αναφέρεται.
- V_posothta , η μεταβλητή αυτή είναι ίδιου τύπου με το αντίστοιχο πεδίο στον πίνακα της ΒΔ που αναφέρεται.

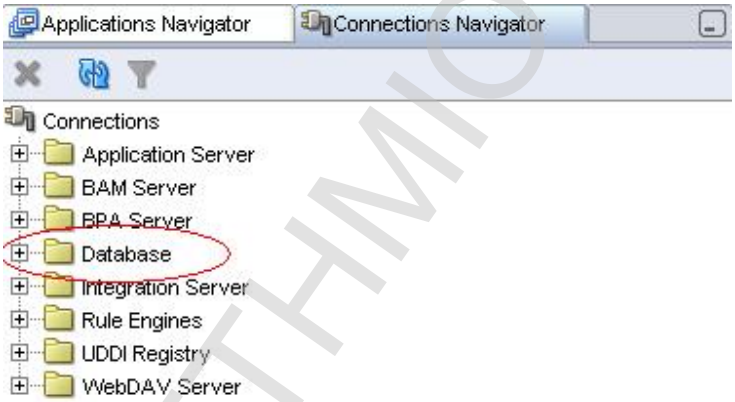
Ενώ επιστρέφεται μια μεταβλητή τύπου number.

Κατά την εκτέλεση αυτής της Function καλούνται δύο select στο πίνακα υλικών του προμηθευτή όπου αναζητείται στην πρώτη περίπτωση η ύπαρξη του υλικού στην ΒΔ ενώ στην δεύτερη περίπτωση ελέγχεται η επάρκεια του αποθέματος .Σε περίπτωση που υπάρχει το υλικό και επαρκεί η ποσότητα επιστρέφεται το γινόμενο της αιτούμενης ποσότητας με την τιμή μονάδας του υλικού, σε αντίθετη περίπτωση επιστρέφεται μηδέν.

Η μετατροπή του package σε Web Service είναι μια σχετικά εύκολή διαδικασία με την χρήση του JDeveloper.

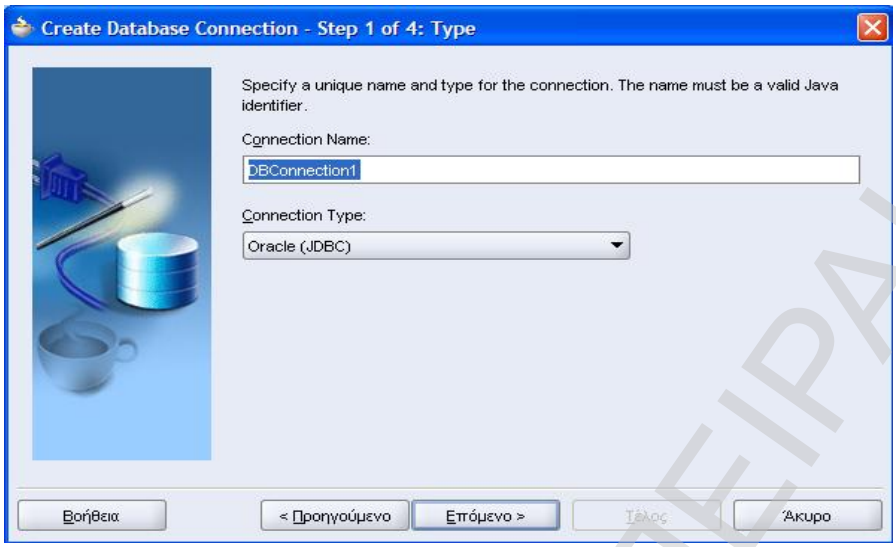
Αρχικά θα πρέπει να δημιουργηθεί μια σύνδεση του IDE με την ΒΔ.

Από το tag του Connection Navigator στον JDeveloper και από την υπάρχουσα λίστα επιλογών , επιλέγουμε το Database.



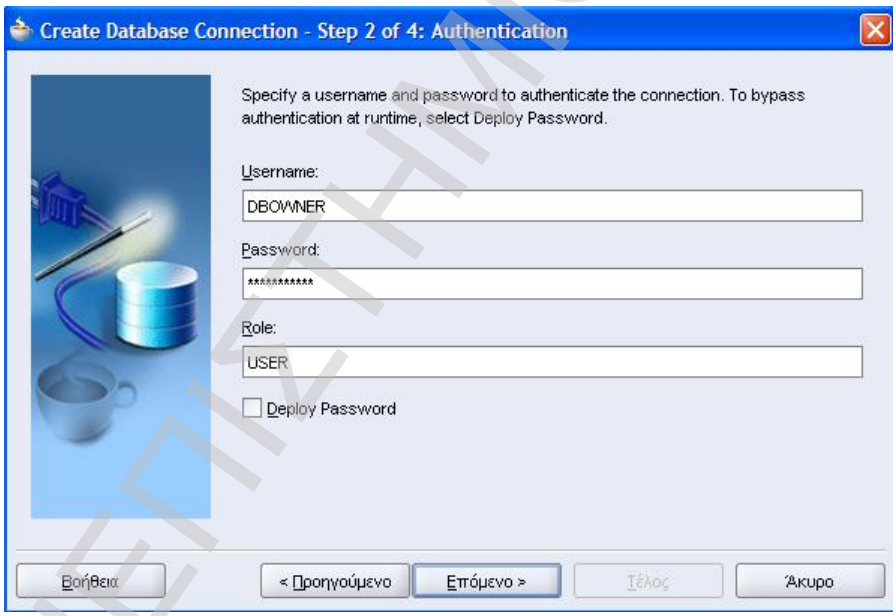
Εικόνα 4: Service Navigator

Στην συνέχεια με δεξί κλικ επιλέγουμε New Database Connection όπου ξεκινάει τον Connection Wizard .Δίνουμε ένα όνομα στην σύνδεση μας και ορίζουμε τον τύπο της (στην περίπτωση Oracle JDBC) και στην συνέχεια επιλέγουμε επόμενο.



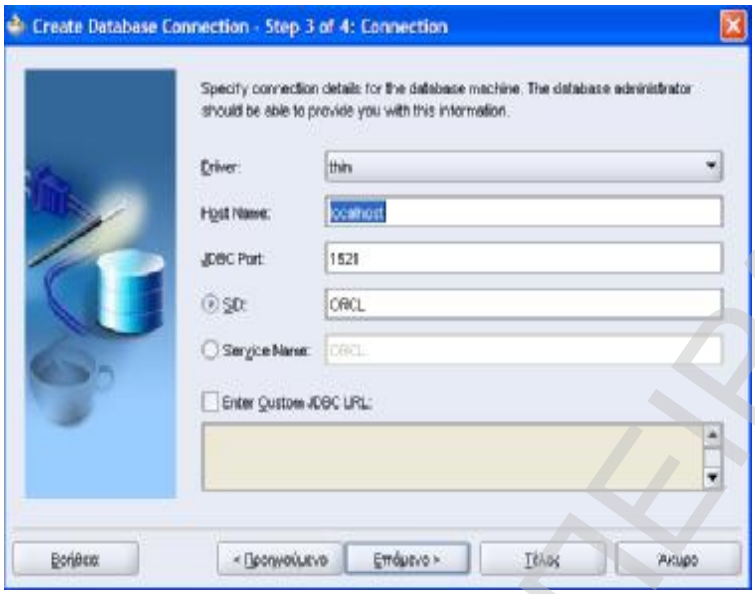
Εικόνα 5: Database Connection 1

Στην επόμενη οθόνη δίνουμε ένα username και ένα password καθώς επίσης ορίζουμε και τον ρόλο του χρήστη που θα συνδεθεί στη ΒΔ, ενώ επιλέγουμε να κάνουμε Deploy το password και πατάμε επόμενο.



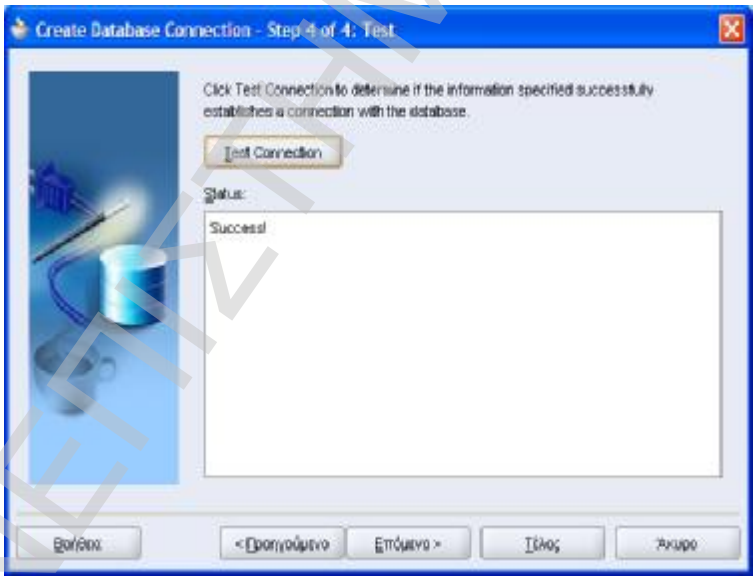
Εικόνα 6: Database Connection 2

Στο επόμενο παράθυρο αφήνουμε την επιλογή του driver ως έχει , δίνουμε το Hostname (στην περίπτωση μας localhost), την θύρα επικοινωνίας , καθώς και το SID της ΒΔ και πατάμε επόμενο.



Εικόνα 7: Database Connection 3

Στην τελευταία οθόνη κάνουμε ένα Test στην σύνδεση που μόλις δημιουργήσαμε και παίρνουμε το ανάλογο μήνυμα.



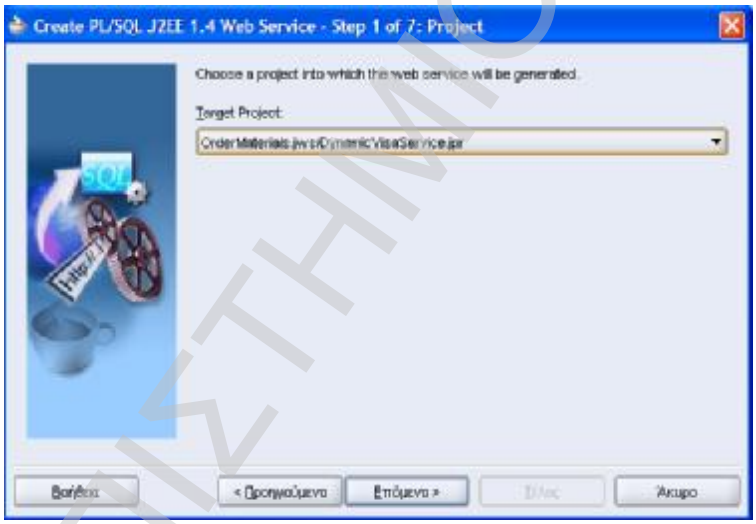
Εικόνα 8: Database Connection 4

Με την ολοκλήρωση της σύνδεσης στην ΒΔ από το tag του Connection Navigator έχουμε την δυνατότητα να εντοπίσουμε το package που δημιουργήσαμε νωρίτερα.
Για την μετατροπή του σε Web Service με δεξί κλικ επιλέγουμε Publish as a Web Service οπότε και ξεκινάει ο Wizard που θα μας βοηθήσει στην μετατροπή του.

Αρχικά θα επιλέξουμε την έκδοση του Web Service που θα δημιουργήσουμε.

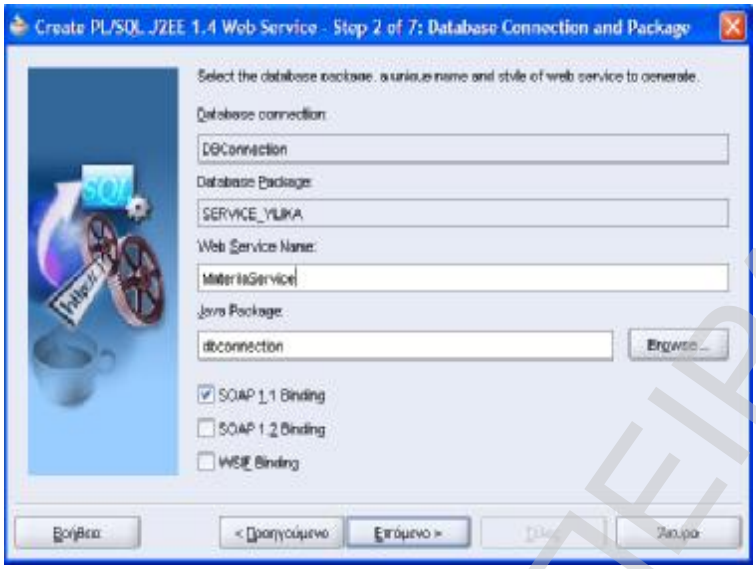
Εικόνα 9: Μετατροπή PL/SQL Block σε Service 1

Στην συνέχεια προσδιορίζουμε το Project στο οποίο θα δημιουργηθεί το Web Service.



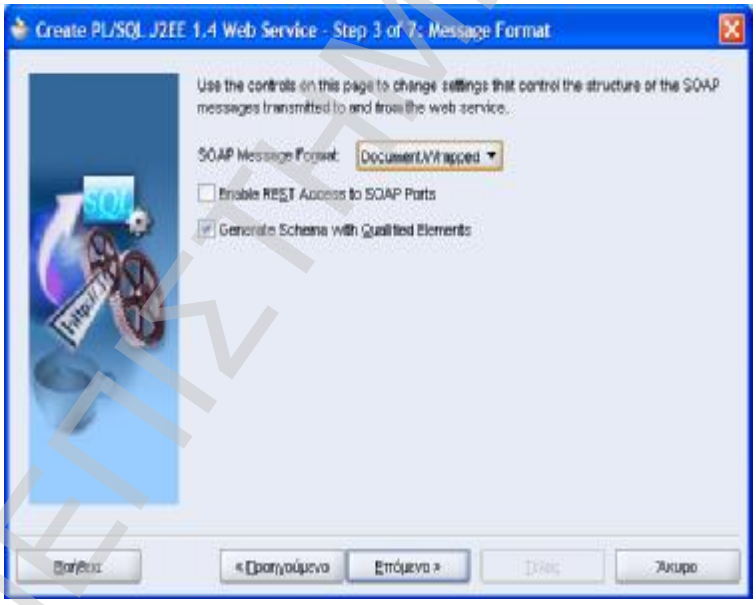
Εικόνα 10: Μετατροπή PL/SQL Block σε Service 2

Κατόπιν επιλέγουμε ένα όνομα για το Web Service καθώς επίσης και το SOAP Binding που θα χρησιμοποιήσουμε.



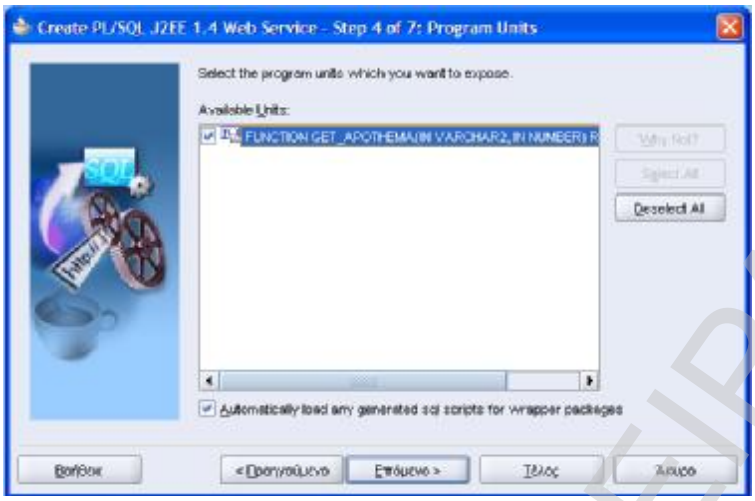
Εικόνα 11: Μετατροπή PL/SQL Block σε Service 3

Ενώ στην επόμενη οθόνη αφήνουμε τις προεπιλογές και πατάμε επόμενο.



Εικόνα 12: Μετατροπή PL/SQL Block σε Service 4

Στο επόμενο βήμα επιλέγουμε την function που θα περιγραφεί στο Web Service, στην περίπτωση μας είναι η GET_APOTHEMA και πατάμε Τέλος .



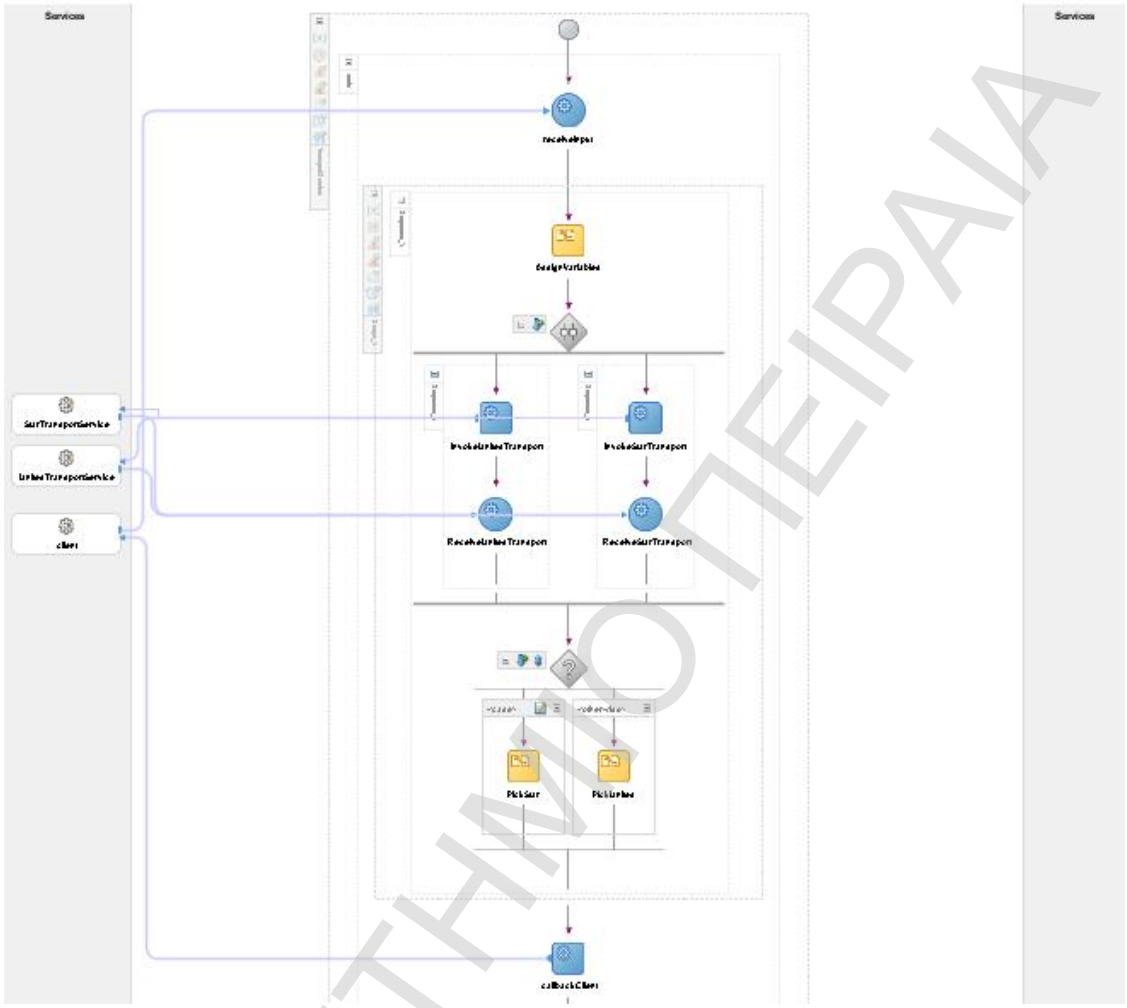
Εικόνα 13: Μετατροπή PL/SQL Block σε Service 5

4.2 Transport Service

Η υπηρεσία αυτή είναι μια BPEL Process η οποία σαν σκοπό έχει να περιγράψει την διαδικασία επιλογής του καλύτερου μεταφορέα με κριτήριο την τιμή και να ενημερώσει τον πελάτη για το ύψος των μεταφορικών εξόδων. Σαν μεταβλητές εισόδου έχει την χώρα παράδοσης καθώς και τις επιθυμητές μέρες εντός της οποίας θα γίνει η παράδοση. Για την ολοκλήρωση του process καλούνται δύο άλλες υπηρεσίες , η StarTransportServive και η UnitedTransportService , οι οποίες εκπροσωπούν αντίστοιχα δύο μεταφορικές εταιρείες που έχουν δημιουργήσει αντίστοιχα Web Services. Η σχεδίαση της συγκεκριμένης υπηρεσίας έγινε με συμβατικό τρόπο υποθέτοντας ότι ο σχεδιαστής ήδη γνώριζε εκ των προτέρων ποιες υπηρεσίες χρειάζεται το process για να ολοκληρωθεί.

Στην περίπτωση λοιπόν που υπήρχε η απαίτηση για να προστεθεί ακόμα μια υπηρεσία ενός άλλου μεταφορέα θα έπρεπε να γίνουν τα παρακάτω βήματα:

- Θα έπρεπε να δημιουργηθεί ένα νέο Sequence.
- Εντός του παραπάνωSequence θα έπρεπε να κληθεί η νέα υπηρεσία με ένα Invoke Activity, ενώ παράλληλα θα πρέπει να γίνει και ο ανάλογος χειρισμός των μεταβλητών που απαιτούνται για την κλήση και ανταλλαγή μηνυμάτων μεταξύ των process.
- Θα πρέπει να δημιουργηθεί μια Receive Activity η οποία θα λαμβάνει την απάντηση της νέας υπηρεσίας με χρήση κάποιας από τις μεταβλητές που προαναφέραμε.
- Τέλος η τιμή μεταφοράς που επιστρέφει η νέα υπηρεσία στην υπηρεσία TransportService αυξάνει την πολυπλοκότητα της ,αφού θα πρέπει να υπολογιστή η νέα χαμηλότερη τιμή μεταφοράς.



Εικόνα 14: Material Service

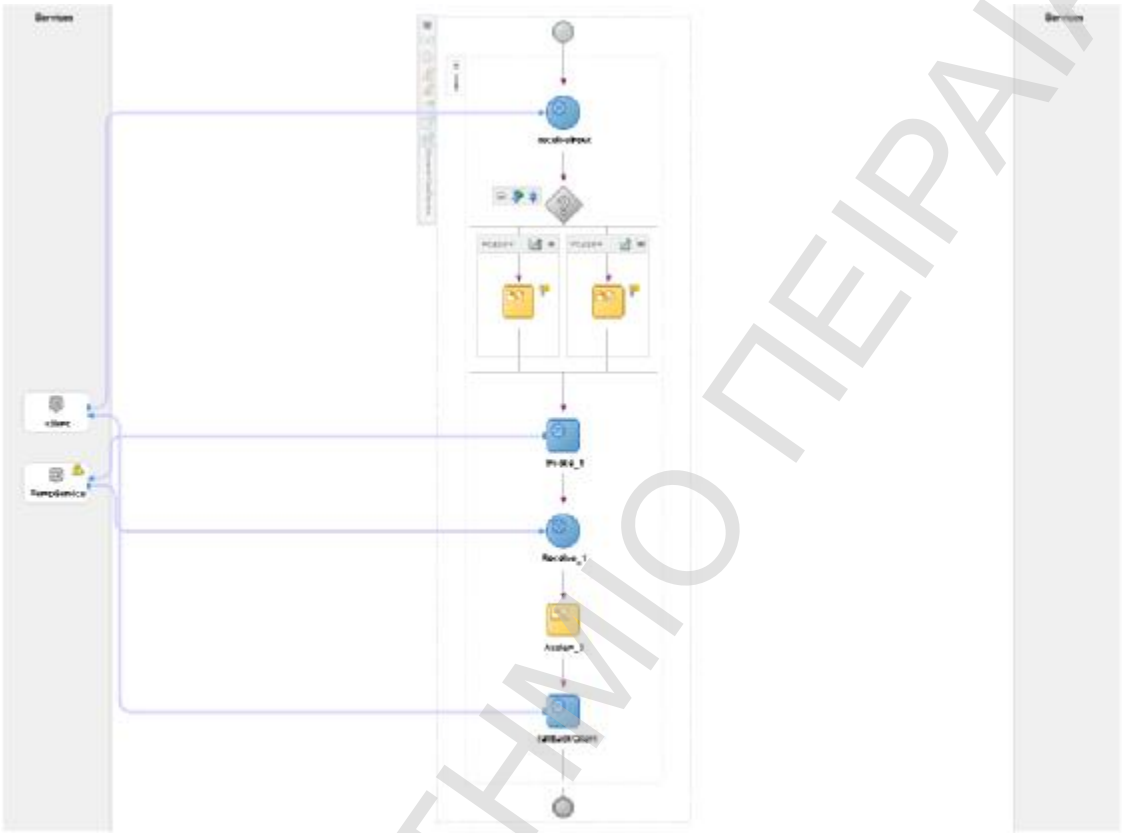
5. Δημιουργία Δυναμικών Διαδικασιών

5.1 DynamicVisaService

Η υπηρεσία αυτή είναι μια BPEL Process της οποίας η λειτουργία αφορά την επιβεβαίωση ύπαρξης του αριθμού της πιστωτικής κάρτας και τον έλεγχο επάρκειας του λογαριασμού του πελάτη για το ποσό της αγοράς που προτίθεται να κάνει. Η διαδικασία αυτή δέχεται σαν μεταβλητές εισόδου δύο παραμέτρους:

- Τον αριθμό της πιστωτικής κάρτας
- Το όνομα της εκδίδουσας τράπεζας της πιστωτικής κάρτας.

Ενώ επιστρέφεται στην κύρια διαδικασία μια Boolean μεταβλητή που εκφράζει την επιβεβαίωση ύπαρξης της κάρτας καθώς την επιβεβαίωση της επάρκειας του προς χρέωση ποσού.
Η δομή της διαδικασίας φαίνεται στην εικόνα.



Εικόνα 15: Dynamic VisaService

Από την παραπάνω εικόνα φαίνεται ότι η υπηρεσία DynamicVisaService καλεί μια άλλη υπηρεσία την Temp και όχι την υπηρεσία κάποιας συγκεκριμένης τράπεζας .Αυτό γίνεται λόγω του δυναμικού σχεδιασμού που έχει το συγκεκριμένο Process.
Μέσω της υπηρεσίας Temp η DynamicVisaService καλεί άλλες υπηρεσίες τραπεζών που ο υπεύθυνος σχεδιασμού και ανάπτυξης πιθανόν να μην γνώριζε κατά τον χρόνο σχεδιασμού του συγκεκριμένου Process , αφήνοντας στην ευχέρεια του κάθε χρήστη να ορίσει ποια υπηρεσία Τράπεζας θα κληθεί κατά την διάρκεια της εκτέλεσης της διαδικασίας.
Στο παράδειγμα αυτό η κλήση κάθε ξεχωριστού Service γίνεται με την εισαγωγή του ονόματος της Τράπεζας.

5.2 Σχεδίαση Δυναμικού Process

5.2.1 Προαπαιτήσεις

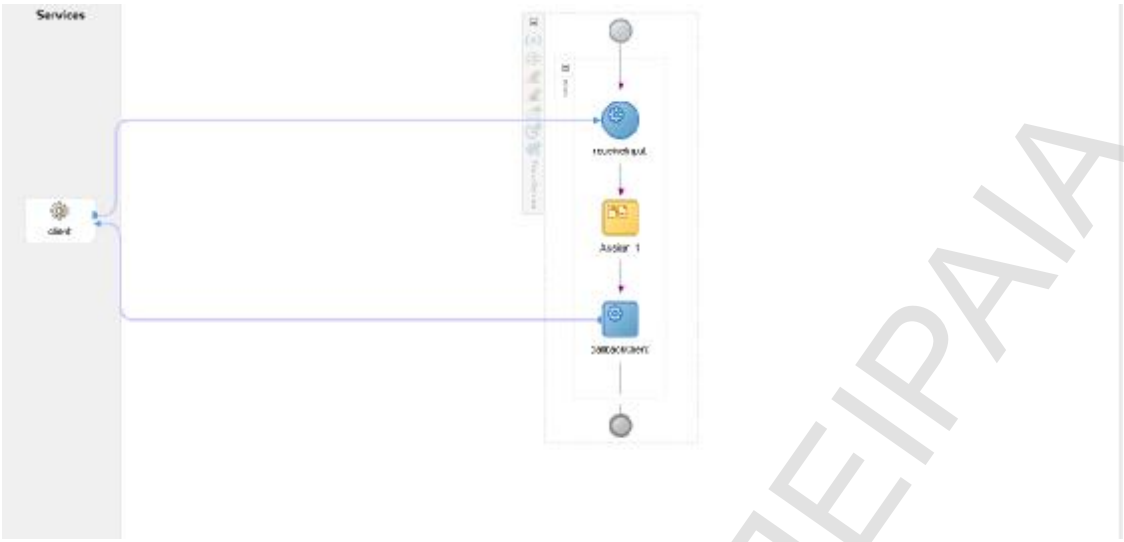
Στην περίπτωση σχεδίασης ενός δυναμικού Process υπάρχει ένα βασικό προαπαιτούμενο. Αυτό έγκειται στο γεγονός ότι κάθε process το οποίο πρόκειται να κληθεί, θα πρέπει να βασίζεται στο ίδιο αρχείο WSDL και στο ίδιο XSD που καθορίζει τα στοιχεία κλήσης και απάντησης , ενώ το μόνο ουσιαστικά που μπορεί να διαφέρει είναι το τελικό URL από το οποίο μπορεί να κληθεί η υπηρεσία.

5.2.2 Σχεδίαση της Temp Process.

Η διαδικασία της σχεδίασης TempProcess δεν διαφέρει από την σχεδίαση οποιοιδήποτε άλλης BPEL Process , παρόλα αυτά θα πρέπει να ληφθούν υπόψη τα παρακάτω ώστε να υπάρχει μια δομημένη σχεδίαση στην συνολική δυναμική διαδικασία:

- Θα πρέπει να σχεδιαστεί ένα XSD αρχείο με στοιχεία κλήσης και απάντησης τέτοια ώστε να ταιριάζουν σε κάθε πιθανή ανάγκη της κύριας διαδικασίας. Ειδικά στα στοιχεία κλήσης (request document) θα πρέπει να παρέχεται η μέγιστη δυνατή πληροφορία σε κάθε πιθανό Service ώστε αυτό με την σειρά του να γνωρίζει το τι θα πρέπει να κάνει. Με τα στοιχεία απάντησης (response document) κάθε process θα πρέπει να μπορεί να παρέχει κάθε απαιτητή πληροφορία που θα χρειαστεί η βασική διαδικασία κατά την διάρκεια της εκτέλεσης της.
- Θα πρέπει να δοθεί ένα όνομα στο process που θα είναι γενικό και αυτό γιατί το WSDL που θα δημιουργηθεί από την συγκεκριμένη διαδικασία θα χρησιμοποιηθεί επίσης από κάθε άλλο process που θα κληθεί μέσω αυτού , οπότε τα στοιχεία messagetypes portypes κτλ θα έχουν το όνομα αυτό.
- Θα πρέπει να καθοριστεί από την αρχή αν το process θα είναι asynchronous ή synchronous εφόσον κάθε άλλο process που θα καλείται μέσω αυτού θα πρέπει να είναι ίδιου τύπου.

Το TempProcess που χρησιμοποιήθηκε στο παράδειγμα είναι το παρακάτω



Εικόνα 16: Temp Process

Το WSDL αρχείο

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="TempService"
             targetNamespace="http://xmlns.oracle.com/TempService"
             xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:client="http://xmlns.oracle.com/TempService"
             xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-
link/">

      <!--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      TYPE DEFINITION - List of services participating in this BPEL
process
      The default output of the BPEL designer uses strings as input and
      output to the BPEL Process. But you can define or import any XML
      Schema type and use them as part of the message types.
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~ -->
      <types>
            <schema xmlns="http://www.w3.org/2001/XMLSchema">
                  <import
namespace="http://xmlns.oracle.com/TempService"
schemaLocation="TempService.xsd" />
```

```
            </schema>
      </types>

      <!--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      MESSAGE TYPE DEFINITION - Definition of the message types used as
      part of the port type defintions
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~ -->
      <message name="TempServiceRequestMessage">
            <part name="payload"
element="client:TempServiceProcessRequest"/>
      </message>

      <message name="TempServiceResponseMessage">
            <part name="payload"
element="client:TempServiceProcessResponse"/>
      </message>



      <!--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      PORT TYPE DEFINITION - A port type groups a set of operations
into
      a logical service unit.
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~ -->
      <!-- portType implemented by the TempService BPEL process -->
      <portType name="TempService">
            <operation name="initiate">
                  <input message="client:TempServiceRequestMessage"/>
            </operation>
      </portType>

      <!-- portType implemented by the requester of TempService BPEL
process
      for asynchronous callback purposes
      -->
      <portType name="TempServiceCallback">
            <operation name="onResult">
```

```
                  <input message="client:TempServiceResponseMessage"/>
            </operation>
      </portType>

      <!--
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
      PARTNER LINK TYPE DEFINITION
      the TempService partnerLinkType binds the provider and
      requester portType into an asynchronous conversation.
      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~ -->
      <plnk:partnerLinkType name="TempService">
            <plnk:role name="TempServiceProvider">
                  <plnk:portType name="client:TempService"/>
            </plnk:role>
            <plnk:role name="TempServiceRequester">
                  <plnk:portType name="client:TempServiceCallback"/>
            </plnk:role>
      </plnk:partnerLinkType>
</definitions>
```

Εικόνα 17: TempProcess.wsdl

Και το αρχείο xsd

```
<schema attributeFormDefault="unqualified"
      elementFormDefault="qualified"
      targetNamespace="http://xmlns.oracle.com/TempService"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="TempServiceProcessRequest">
            <complexType>
                  <sequence>
                        <element name="VisaNumber" type="string"/>
                              <element name="LoanAmount"
type="double"/>
                  </sequence>
            </complexType>
      </element>
```

```
      <element name="TempServiceProcessResponse">
            <complexType>
                  <sequence>
                        <element name="Approval" type="boolean"/>
                  </sequence>
            </complexType>
      </element>
</schema>
```

Εικόνα 18: TempProcess.xsd

5.2.3 **Δημιουργία Επιπλέον** Process

Υποθέτοντας ότι έχουμε δημιουργήσει ένα BPEL Process που περιγράφει την διαδικασία πιστοποίησης της υποτιθέμενης τράπεζας CyprusBankService. Ο τρόπος σχεδιασμού και ανάπτυξης δεν διαφέρει σε αυτό το σημείο από κανένα άλλο process , το μόνο που θα πρέπει στην παρούσα φάση να είναι κοινό με το TempProcess είναι το είδος της διαδικασίας (synchronous , asynchronous).
Στην συνέχεια η διαδικασία μετατροπής του σε δυναμικά καλούμενη διαδικασία μέσω της TempProcess περιλαμβάνει τα εξής βήματα:

- Αντιγράφεται και επικολλείται ολόκληρο το WSDL αρχείο της TempProcess στην θέση του CyprusBankService.wsdl.
- Στην συνέχεια γίνονται οι αλλάγες στο CyprusBankService.bpel αρχείο ώστε να ταυτίζεται με το νέο wsdl αρχείο του process και συγκεκριμένα:
  i. Αλλάζονται τα namespaces .
  ii. Αλλάζονται τα portypes.
  iii. Αλλάζονται τα PartnerLink Types.
  iv. Αλλάζονται και τα ονόματα των μεταβλητών εισόδου και εξόδου.

Το WSDL και το BPEL αρχείο αντίστοιχα φαίνονται παρακάτω.

```
<definitions
     name="TempService"
     targetNamespace="http://xmlns.oracle.com/TempService"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
     xmlns:client="http://xmlns.oracle.com/TempService"
    >
```

```
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
             xmlns:client="http://xmlns.oracle.com/TempService">
            <import namespace="http://xmlns.oracle.com/TempService"
schemaLocation="TempService.xsd"/>
        </schema>
    </types>
    <message name="TempServiceResponseMessage">
        <part name="payload"
element="client:TempServiceProcessResponse"/>
    </message>
    <message name="TempServiceRequestMessage">
        <part name="payload"
element="client:TempServiceProcessRequest"/>
    </message>
    <portType name="TempServiceCallback">
        <operation name="onResult">
            <input message="client:TempServiceResponseMessage"/>
        </operation>
    </portType>
    <portType name="TempService">
        <operation name="initiate">
            <input message="client:TempServiceRequestMessage"/>
        </operation>
    </portType>
    <plnk:partnerLinkType name="TempService">
        <plnk:role name="TempServiceProvider">
            <plnk:portType name="client:TempService"/>
        </plnk:role>
        <plnk:role name="TempServiceRequester">
            <plnk:portType name="client:TempServiceCallback"/>
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

Εικόνα 19: CyprusBankService.wsdl

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<!--
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////
  Oracle JDeveloper BPEL Designer

  Created: Sat Jan 16 18:33:46 EET 2010
  Author:  G.Vlahakis
  Purpose: Asynchronous BPEL Process
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////
-->
<process name="CyprusBankService"
         targetNamespace="http://xmlns.oracle.com/TempService"
         xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-
process/"
         xmlns:bpws="http://schemas.xmlsoap.org/ws/2003/03/business-
process/"

xmlns:xp20="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.serv
ices.functions.Xpath20"
         xmlns:hwf="http://xmlns.oracle.com/bpel/workflow/xpath"

xmlns:ids="http://xmlns.oracle.com/bpel/services/IdentityService/xpath"
         xmlns:ldap="http://schemas.oracle.com/xpath/extension/ldap"

xmlns:ehdr="http://www.oracle.com/XSL/Transform/java/oracle.tip.esb.ser
ver.headers.ESBHeaderFunctions"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:bpelx="http://schemas.oracle.com/bpel/extension"
         xmlns:client="http://xmlns.oracle.com/TempService"
         xmlns:ora="http://schemas.oracle.com/xpath/extension"

xmlns:orcl="http://www.oracle.com/XSL/Transform/java/oracle.tip.pc.serv
ices.functions.ExtFunc"

xmlns:xref="http://www.oracle.com/XSL/Transform/java/oracle.tip.xref.xp
ath.XRefXPathFunctions">
```

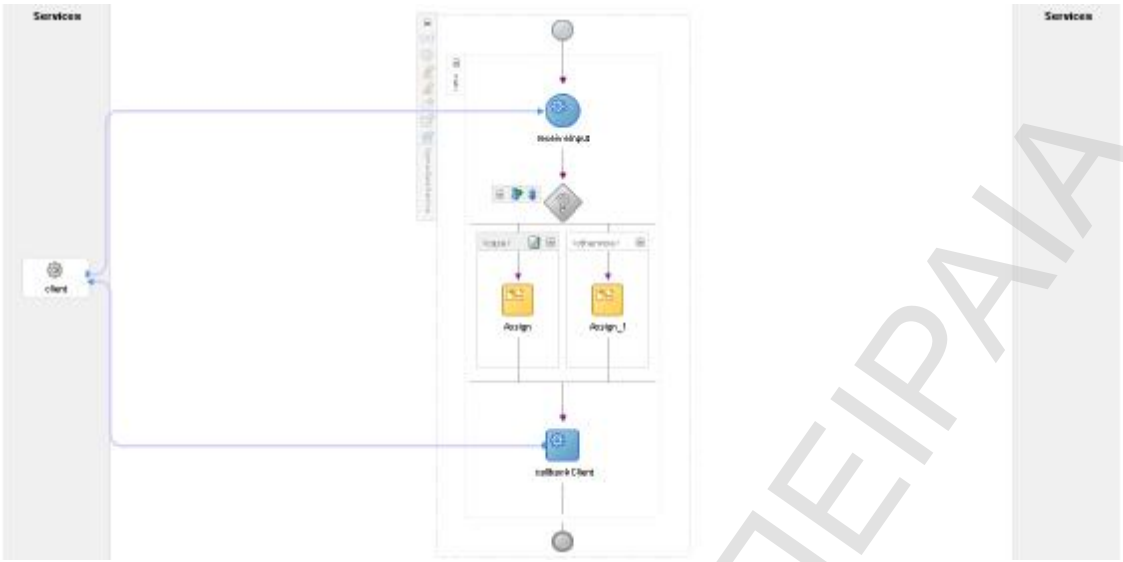
```
    <!--

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////
        PARTNERLINKS
        List of services participating in this BPEL process

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////
    -->
    <partnerLinks>
        <!--
      The 'client' role represents the requester of this service. It is
      used for callback. The location and correlation information
associated
      with the client role are automatically set using WS-Addressing.
    -->
        <partnerLink name="client" partnerLinkType="client:TempService"
                     myRole="TempServiceProvider"
                     partnerRole="TempServiceRequester"/>
    </partnerLinks>
    <!--

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
//////////
        VARIABLES
        List of messages and XML documents used within this BPEL
process

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////
/////////
    -->
    <variables>
        <!-- Reference to the message passed as input during initiation
-->
        <variable name="inputVariable"
```

```
                    messageType="client:TempServiceRequestMessage"/>
        <!-- Reference to the message that will be sent back to the
requester during callback -->
        <variable name="outputVariable"
                    messageType="client:TempServiceResponseMessage"/>
    </variables>
    <!--

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////
       ORCHESTRATION LOGIC
       Set of activities coordinating the flow of messages across the
       services integrated within this business process

////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////
////////
    -->
    <sequence name="main">
        <receive name="receiveInput" partnerLink="client"
                    portType="client:TempService" operation="initiate"
                    variable="inputVariable" createInstance="yes"/>
        <!--
          Asynchronous callback to the requester. (Note: the callback
location and correlation id is transparently handled using WS-
addressing.)
        -->
        <switch name="Switch_1">
            <case
condition="bpws:getVariableData('inputVariable','payload','/client:Temp
ServiceProcessRequest/client:VisaNumber')='c123456789' and
bpws:getVariableData('inputVariable','payload','/client:TempServicePro
cessRequest/client:LoanAmount')  &lt;  1000">
                <assign name="Assign">
                    <copy>
                        <from expression="'1'"/>
                        <to variable="outputVariable" part="payload"

query="/client:TempServiceProcessResponse/client:Approval"/>
```

```
                    </copy>
                </assign>
            </case>
            <otherwise>
                <assign name="Assign_1">
                    <copy>
                        <from expression="'0'"/>
                        <to variable="outputVariable" part="payload"

query="/client:TempServiceProcessResponse/client:Approval"/>
                    </copy>
                </assign>
            </otherwise>
        </switch>
        <invoke name="callbackClient" partnerLink="client"
                portType="client:TempServiceCallback"
operation="onResult"
                inputVariable="outputVariable"/>
    </sequence>
</process>
```

Εικόνα 20: CyprusBankService.bpel

- Τέλος γίνεται import το σχήμα xsd που αφορά το TempProcess. Εδώ θα πρέπει να σημειωθεί ότι για το xsd σχήμα στην περίπτωση ενός παραγωγικού περιβάλλοντος καλό θα είναι να τοποθετείται σε ένα HTTP Server και να γίνεται η κλήση του από εκεί , ώστε οποιαδήποτε αλλαγή γίνεται σε αυτό να ενημερώνει αυτόματα όλα τα process που αναφέρονται σε αυτό.



Εικόνα 21: CyprusBankService.bpelΝαδω τι είναι

5.2.4 Δημιουργία Καλούσας Διαδικασίας

Δημιουργούμε μια βασική δομή μιας asynchronous BPEL process .

Δημιουργούμε ένα PartnerLink το οποίο βασίζεται στην TempService που δημιουργήσαμε προηγουμένως και έχει γίνει deploy στο τοπικό BPEL Process Manager.

Στην συνέχεια προσθέτουμε το ακόλουθο namespace στο BPEL αρχείο του process.

```
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
```

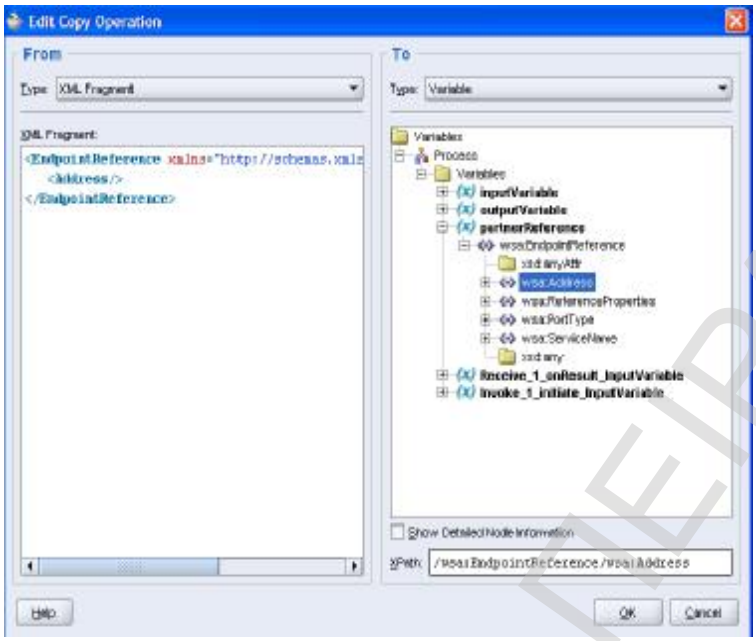
Προσθέτουμε μια μεταβλητή τύπου Endpoint Reference

```
xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
```

Αυτή η μεταβλητή θα πρέπει να αρχικοποιηθεί ώστε να μην έχουμε ένα μήνυμα null pointer exception κατά την διάρκεια του deploy.

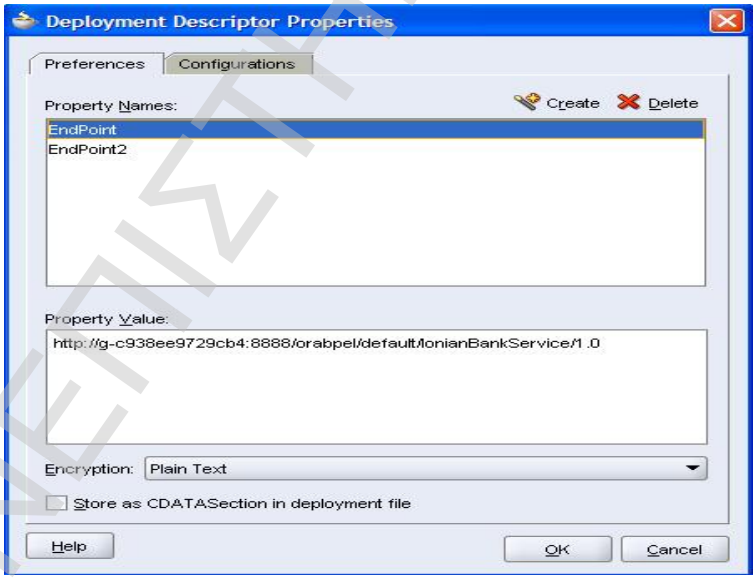
Για να αρχικοποιηθεί η μεταβλητή δημιουργούμε μια Assign Activity και σε ένα Copy Operation γράφουμε το παρακάτω κομμάτι κώδικα.

```
<EndpointReference
xmlns="http://schemas.xmlsoap.org/ws/2003/03/addressing">
   <Address/>
</EndpointReference>
```



Εικόνα 22: Δημιουργία Copy Operation

Δημιουργούμε στο Deployment Descriptor δύο Property και τους δίνουμε τις τιμές των URL των διαδικασιών που πρόκειται να κληθούν.

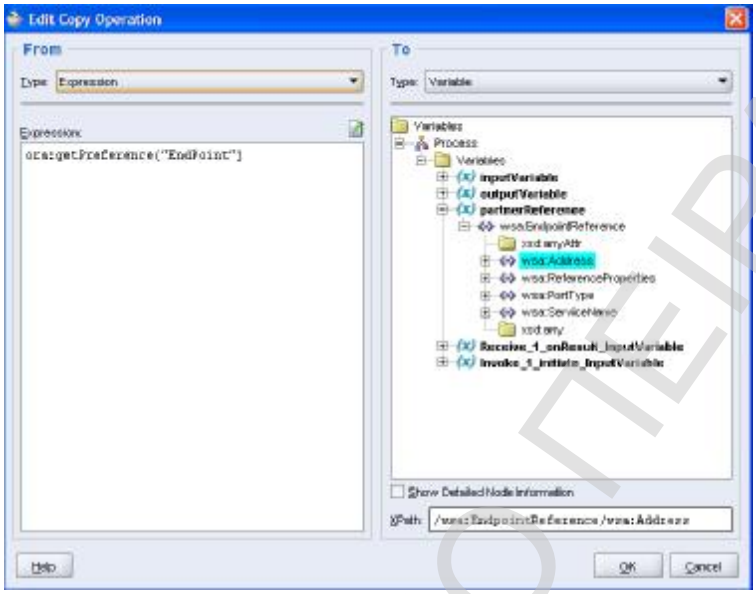


Εικόνα 23: Deployment Descriptor

Στην συνέχεια στην ίδια Assign Activity δημιουργούμε μια νέα Copy Operation και αυτή την φορά γράφουμε το παρακάτω expression.

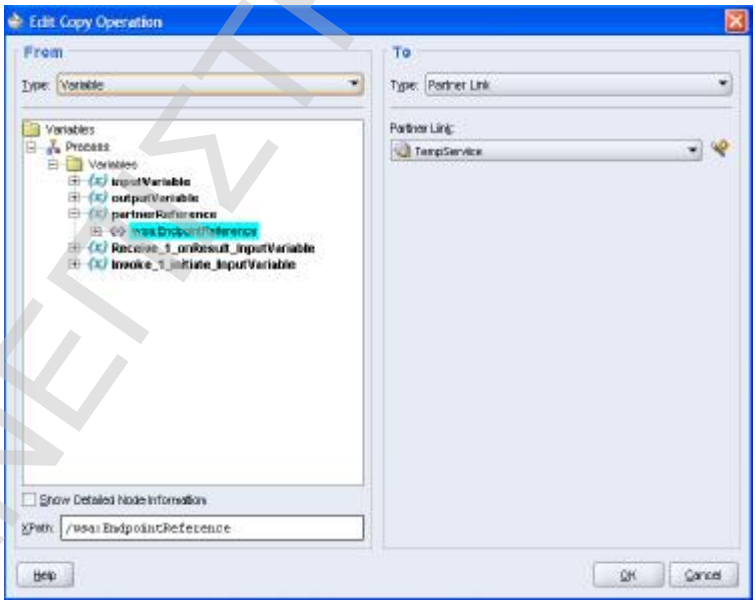
ora:getPreference(“Endpoint”)

Και δίνουμε τιμή στο στοιχείο Address της μεταβλητής PartnerReference.

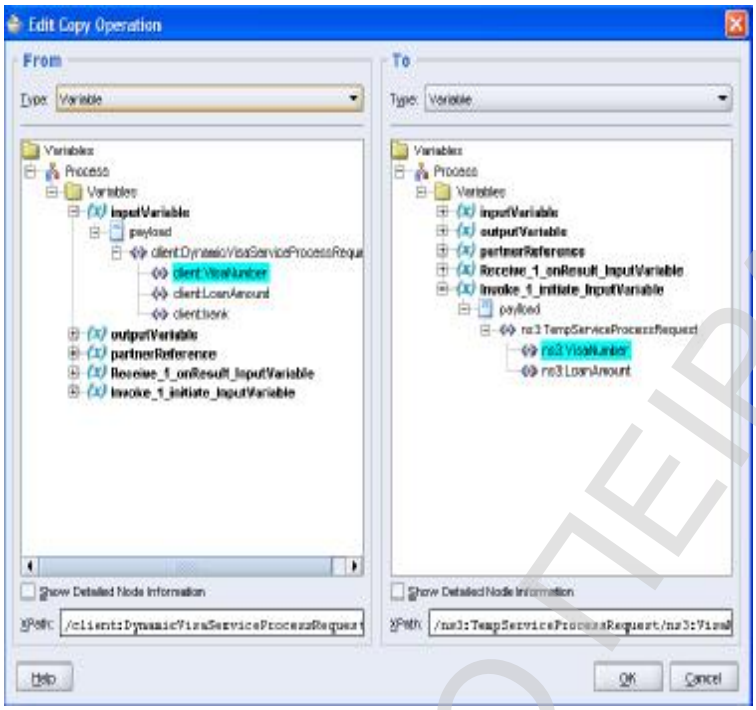


Εικόνα 24: Δημ/ργία Copy Operation με χρήση Expression

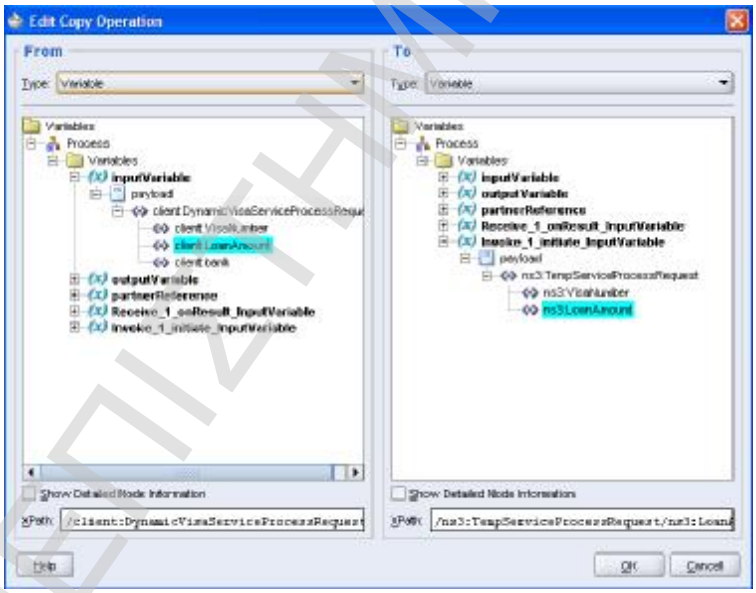
Τέλος στην μεταφέρουμε την μεταβλητή EndpointReference με μια Copy Operation στην TempService ενώ αρχικοποιούμε της μεταβλητές της με αντίστοιχες Copy Operation .



Εικόνα 25: Δημ/ργία Copy Operation σε PartnerLink

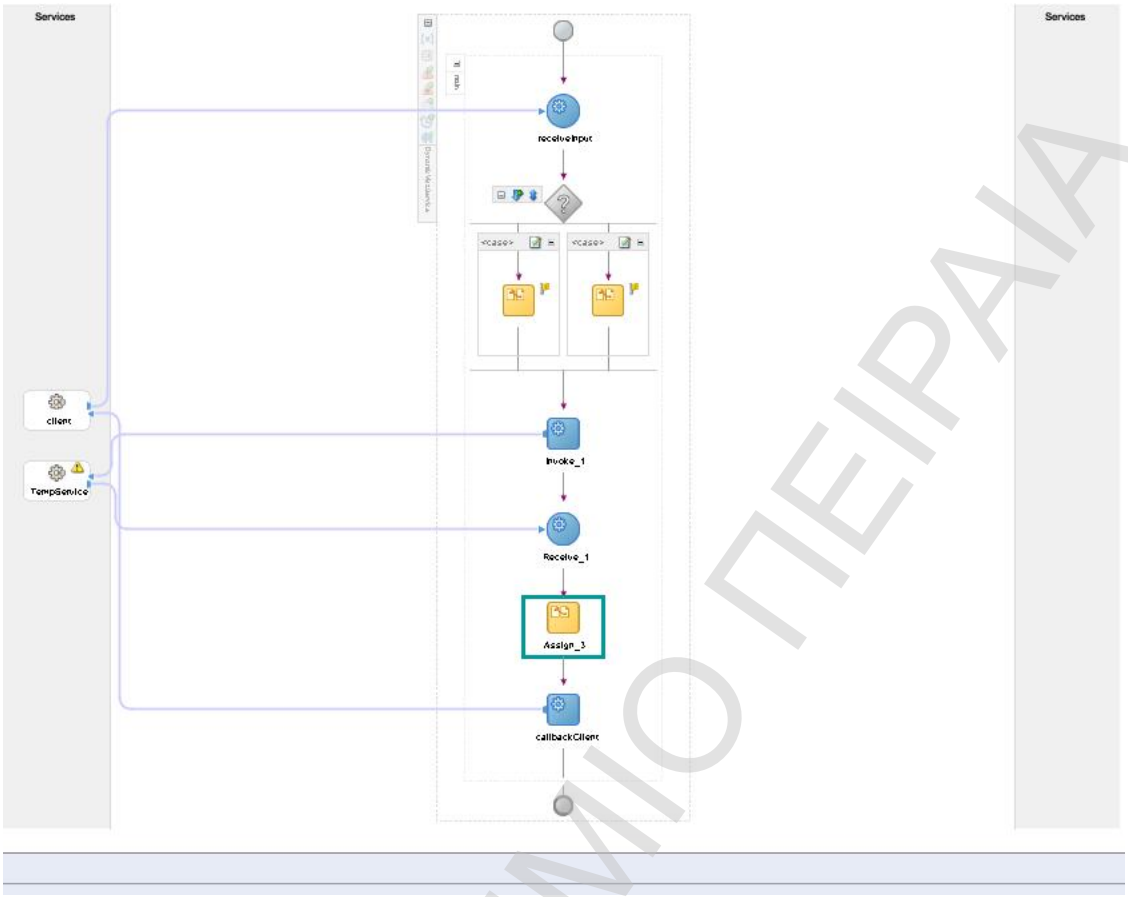


Εικόνα 26: Δημ/ργία Copy Operation στις μεταβλητές του TempService 1



Εικόνα 27: Δημ/ργία Copy Operation στις μεταβλητές του TempService 2

Η συνολική εικόνα του process φαίνεται παρακάτω



Εικόνα 28: Συνολική Εικόνα του DynamicVisa Service

Τελικά τα αρχεία WSDL και BPEL θα είναι όπως παρακάτω

```
<definitions
     name="DynamicVisaService"
     targetNamespace="http://xmlns.oracle.com/DynamicVisaService"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:ns1="http://schemas.xmlsoap.org/ws/2003/03/addressing"
     xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"

xmlns:pns1="http://xmlns.oracle.com/DynamicVisaService/correlationset"
     xmlns:client="http://xmlns.oracle.com/DynamicVisaService"
    >
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
```

```
            <import
namespace="http://xmlns.oracle.com/DynamicVisaService"
schemaLocation="DynamicVisaService.xsd"/>
        </schema>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import
namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
schemaLocation="http://g-c938ee9729cb4:8888/orabpel/xmllib/ws-
addressing.xsd"/>
        </xsd:schema>
    </types>
    <message name="DynamicVisaServiceResponseMessage">
        <part name="payload"
element="client:DynamicVisaServiceProcessResponse"/>
    </message>
    <message name="DynamicVisaServiceRequestMessage">
        <part name="payload"
element="client:DynamicVisaServiceProcessRequest"/>
    </message>
    <portType name="DynamicVisaServiceCallback">
        <operation name="onResult">
            <input message="client:DynamicVisaServiceResponseMessage"/>
        </operation>
    </portType>
    <portType name="DynamicVisaService">
        <operation name="initiate">
            <input message="client:DynamicVisaServiceRequestMessage"/>
        </operation>
    </portType>
    <plnk:partnerLinkType name="DynamicVisaService">
        <plnk:role name="DynamicVisaServiceProvider">
            <plnk:portType name="client:DynamicVisaService"/>
        </plnk:role>
        <plnk:role name="DynamicVisaServiceRequester">
            <plnk:portType name="client:DynamicVisaServiceCallback"/>
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

Εικόνα 29: DynamicVisa Service.wsdl

```
<definitions
     name="DynamicVisaService"
     targetNamespace="http://xmlns.oracle.com/DynamicVisaService"
     xmlns="http://schemas.xmlsoap.org/wsdl/"
     xmlns:ns1="http://schemas.xmlsoap.org/ws/2003/03/addressing"
     xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"

xmlns:pns1="http://xmlns.oracle.com/DynamicVisaService/correlationset"
     xmlns:client="http://xmlns.oracle.com/DynamicVisaService"
    >
    <types>
        <schema xmlns="http://www.w3.org/2001/XMLSchema">
            <import
namespace="http://xmlns.oracle.com/DynamicVisaService"
schemaLocation="DynamicVisaService.xsd"/>
        </schema>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <xsd:import
namespace="http://schemas.xmlsoap.org/ws/2003/03/addressing"
schemaLocation="http://g-c938ee9729cb4:8888/orabpel/xmllib/ws-
addressing.xsd"/>
        </xsd:schema>
    </types>
    <message name="DynamicVisaServiceResponseMessage">
        <part name="payload"
element="client:DynamicVisaServiceProcessResponse"/>
    </message>
    <message name="DynamicVisaServiceRequestMessage">
        <part name="payload"
element="client:DynamicVisaServiceProcessRequest"/>
    </message>
    <portType name="DynamicVisaServiceCallback">
        <operation name="onResult">
            <input message="client:DynamicVisaServiceResponseMessage"/>
        </operation>
    </portType>
    <portType name="DynamicVisaService">
        <operation name="initiate">
            <input message="client:DynamicVisaServiceRequestMessage"/>
```

```
        </operation>
    </portType>
    <plnk:partnerLinkType name="DynamicVisaService">
        <plnk:role name="DynamicVisaServiceProvider">
            <plnk:portType name="client:DynamicVisaService"/>
        </plnk:role>
        <plnk:role name="DynamicVisaServiceRequester">
            <plnk:portType name="client:DynamicVisaServiceCallback"/>
        </plnk:role>
    </plnk:partnerLinkType>
</definitions>
```

Εικόνα 30: DynamicVisa Service.bpel

Τελειώνοντας την σχεδίαση του process DynamicVisaService το κάνουμε deploy στο τοπικό BPEL Process Manager.

6. Συμπεράσματα

6.1 Αξιολόγηση Δυναμικού Τρόπου Σχεδιασμού Bpel Process

Ο δυναμικός τρόπος σχεδιασμού BPEL Process λειτουργεί με παρόμοιο τρόπο όπως στις παραδοσιακές αντικειμενοστραφής γλώσσες προγραμματισμού λειτουργεί η αρχή του δομημένου το Modularization. Τα πλεονεκτήματα αυτής της τεχνικής είναι:

- Επιτρέπει την ομαδική εργασία σε περιβάλλοντας ανάπτυξης χωρίζοντας τον όγκο της εργασίας σε μικρότερα τμήματα.
- Η δημιουργία και η διαχείριση επιπρόσθετων subprocess χωρίς την ανάγκη για τροποποίηση και redeploy της βασικής κύριας διαδικασίας.
- Μικρότερη ανάγκη για χρήση , συντήρηση και αύξηση της αποτελεσματικότητας μεμονωμένων process.
- Οι όποιες αλλαγές και τροποποιήσεις γίνονται στις subpocess δεν επηρεάζουν την κύρια διαδικασία, ενώ είναι άμεσα προσβάσιμες σε αυτήν.

Επιπρόσθετα θα πρέπει να σημειωθεί ότι οι δυναμικές διαδικασίες επιτρέπουν στο σύστημα να προσαρμόζεται σε συνθήκες που αλλιώς δεν θα ήταν γνωστές στην διάρκεια της σχεδίασης του process. Έτσι και στην περίπτωση μας η ροή του process καθορίζεται από το περιεχόμενο των δεδομένων που θα εισάγει ο χρήστης.

6.2 Μελλοντικές Προεκτάσεις

Ο δυναμικός τρόπος ανάπτυξης στην σχεδίαση επιχειρηματικών διαδικασιών δεν είναι δυνατόν να περιορίζεται μόνο στη συγκεκριμένη μέθοδο ανάπτυξης.

Μια μελλοντική προέκταση της παραπάνω εργασίας είναι πιθανών η δυναμική αλλαγή περισσότερων παραμέτρων της διαδικασίας, πέρα από τα partner links, κατά την διάρκεια εκτέλεσης του process. Με αυτόν τον τρόπο σχεδιασμού επιχειρηματικοί κανόνες θα μπορούν να συνδυάζονται μεταξύ τους κατά περίπτωση και ανάλογα ίσως των δεδομένων εισόδου και του χρόνου εκτέλεσης που θα απαιτεί το process.

6.3 **Επίλογος**

Στην παρούσας εργασία παρουσιάστηκε ο τρόπος δημιουργίας δυναμικών ροών εργασίας που βασίζεται στην κατά περίπτωση επιλογή του κατάλληλου partner link, σε ένα απλουστευμένο σενάριο ανάπτυξης που ωστόσο δεν επηρέασε ούτε κατ'ελάχιστο την μεθοδολογία ανάπτυξης.
Οι δυνατότητες που παρέχονται από την δημιουργία δυναμικών ροών εργασίας με τον συγκεκριμένο τρόπο αυξάνουν σε μεγάλο βαθμό την αποτελεσματικότητα SOA εφαρμογών προσθέτοντας επιπλέον πλεονεκτήματα όπως της ευελιξίας , της φορητότητας της μείωσης της πολυπλοκότητας και της προσαρμοστικότητας στις ενδεχόμενες αλλαγές.
Το μέλλον των πληροφορικών συστημάτων βρίσκεται στη SOA και στις τεχνολογίες που την αποτελούν. Τα Web Services καθώς και η Bpel που είναι υπεύθυνη για την ενορχήστρωση των διαδικασιών θα αποτελέσουν κυρίαρχα στοιχεία στις μελλοντικές εφαρμογές διαδικτύου.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Βιβλιογραφικές Αναφορές

[1] "SOA BEST Practices: The BPEL Cookbook: Best Practices for SOA-based integration and composite applications development "ISBN : 1904811337, Συγγραφείς: Arun Poduval, Doug Todd, Harish Gaur, Jeremy Bolie, Jerry Thomas, Kevin Geminiuc, Lawrence Pravin, Markus Zirn, Matjaz Juric, Michael Cardella, Praveen Ramachandran, Sean Carey, Stany Blanvalet, The Hoa Nguyen, Yves Coene

[2]"SOA in Practice: The Art of Distributed System Design",Συγγραφέας : Nicolai M. Josuttis,Εκδόσεις :O'Reilly.

[3] "OASIS Web Services Business Process Execution Language Version 2.0 Primer "

[4] "OASIS Web Services Business Process Execution Language Version 2.0"

[5] "BPEL 100 Success Secrets - Business Process Execution Language for Web Services- THE XML-based language for the formal specification of business processes" Συγγραφέας:Tony Willis

[6] "OASIS Web Services Business Process Execution Language Version 2.0"

[7] "Enhancing BPEL scenarios with Dynamic Relevance-Based Exception Handling" Συγγραφείς: Kareliotis Christos, Dr. Vassilakis Costas, Dr. Georgiadis Panayiotis.

[8] "On Accommodating Inter Service Dependencies in Web Process Flow Composition" Συγγραφείς: Kunal Verma, Rama Akkiraju, Richard Goodwin, Prashant Doshi, Juhnyoung Lee.

[9] "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer" Συγγραφείς: David Booth, Canyang Kevin Liu.

[10] "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" Συγγραφείς: Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, Sanjiva Weerawarana.

- Διαδικτυακές Αναφορές

[3W1] http://www.oracle.com/technology/products/ias/bpel/pdf/orabpel-Tutorial3-DataManipulationTutorial.pdf

[3W2] http://darwin-it.blogspot.com/2008/11/dynamic-partnerlinks-made-simple.html

[3W3] http://www.sti-innsbruck.at/fileadmin/documents/sws_ss09/tutorial/3-WS-BPEL.pdf

[3W4] http://www.cs.colorado.edu/~kena/classes/7818/f06/lectures/WSDL.pdf

[3W5] http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html

[3W6] http://www.oracle.com/technology/pub/articles/matjaz_bpel2.html

[3W7] http://wiki.open-esb.java.net/Wiki.jsp?page=DynamicPartnerLinks

[3W8] http://java.sun.com/developer/Books/j2ee/jwsa/JWSA_CH02.pdf

[3W9] http://www.ciber.com/bidw/soa/

[3W10] http://www.1105govinfo.com/pdfs/custom/SP_SOA_10202008_PDF.pdf

[3W11] http://www.softcare.com/whitepapers/wp_whatis_bpel.php

WS –BPEL

Εισαγωγή

Η WS-BPEL(Business Process Execution Language for Web Services) είναι γλώσσα που χρησιμοποιείται για τον ορισμό, περιγραφή και την εκτέλεση επιχειρηματικών διαδικασιών χρησιμοποιώντας Web services. Η BPEL ολοκληρώνει την Service Oriented Architecture (SOA) , συνθέτοντας , ενορχηστρώνοντας και συντονίζοντας Web Services, παρέχοντας ένα σχετικά εύκολο και απλό τρόπο για την σύνθεση υπηρεσιών σε μια ολοκληρωμένη επιχειρηματική διαδικασία.

Η BPEL4WS αρχικά παρουσιάστηκε τον Ιούλιο του 2002 με την έκδοση BPEL4WS 1.0 κατόπιν συνδυασμένης προσπάθειας των εταιρειών IBM , Microsoft και την BEA. Τον Μάρτιο του επόμενου έτους παρουσιάστηκε η έκδοση 1.1 κατόπιν συμμετοχής και άλλων εταιρειών όπως η SAP και η Siebel Systems και η οποία έλαβε αξιόλογη προσοχή από εταιρείες ανάπτυξης λογισμικού που οδήγησε σε ένα αριθμό μηχανών ενορχήστρωσης BPEL διαδικασιών.

Η έκδοση η οποία υφίσταται κατά την συγγραφή της παρούσης εργασίας είναι η 2.0 η οποία αποτελεί και επίσημο , ανοικτό πρότυπο.

BPEL **Δομή**

Μια διαδικασία BPEL είναι μια δομή στην οποία μπορούν να ορισθούν σχέσεις με εξωτερικές επαφές (πχ Web Services) , ορισμοί επεξεργασίας δεδομένων , ρουτίνες χειρισμού διαφορών γεγονότων και κυρίως οι δραστηριότητες οι οποίες συνθέτουν την διαδικασία.

Υπάρχουν δύο τύποι BPEL διαδικασιών :

- Οι Executable (εκτελέσιμες) διαδικασίες
- Οι Abstract (αφαιρετικές) διαδικασίες

Οι μεν πρώτες ορίζουν μια πλήρη επιχειρηματική λογική παρουσιάζοντας το σύνολο των εξωτερικών τους επαφών καθώς και το εσωτερικό εκτελέσιμο κομμάτι της διαδικασίας ενώ οι δεύτερες παρουσιάζουν μερικώς την διαδικασία χωρίς λεπτομέρειες εκτέλεσης και χωρίς τις ενδεχόμενες εξωτερικές επαφές , αποτελώντας ένα πρότυπο για συγκεκριμένες επιχειρηματικές διαδικασίες.

Το στοιχείο που βρίσκεται στην κορυφή μιας BPEL δομής είναι το στοιχείο process για το οποίο η σύνταξη φαίνεται παρακάτω.

```
<process name="PrimerProcess"
targetNamespace="http://oasis-open.org/WSBPEL/Primer/"
xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/executable" />
```

Κάθε μεταβλητή που ορίζεται σε αυτό το σημείο είτε κάθε εξωτερική επαφή που δηλώνεται εδώ θεωρείται σαν global και μπορεί να αναφερθεί σε οποιοδήποτε σημείο το BPEL κειμένου αργότερα.

Ένα άλλο βασικό δομικό στοιχείο της BPEL είναι το scope , το οποίο έχει κατά κάποια έννοια τον ρόλο του block σε αντιστοιχία με μια παραδοσιακή γλώσσα προγραμματισμού.

```
<scope name="Scope" />
```

Με την χρήση του στοιχείου αυτού μπορούμε να αποδομίσουμε την επιχειρηματική λογική και σε κάθε στοιχείο scope να περιλάβουμε ένα μόνο μέρος αυτής .Σε αντίθεση με την ισχύς που έχουν οι δηλώσεις που γίνονται στο στοιχείο process εδώ οι δηλώσεις είναι τοπικού χαρακτήρα , όπως επίσης και το σύνολο των εργασιών.

Εξωτερικές Επαφές

Ένα βασικό χαρακτηριστικό της BPEL είναι η δυνατότητα επικοινωνίας με Web Services και ο συντονισμός τους ώστε να συνθέσει μια ορισμένη επιχειρηματική λογική. Κάθε επαφή που γίνεται για την κλήση μια υπηρεσίας ονομάζεται Partner link. Ένα Partner link είναι ένα στιγμιότυπο στο οποίο ορίζονται τα αναγκαία WSDL port types που προσφέρονται από την BPEL και απαιτούνται από την υπηρεσία για να γίνει η επαφή με αυτήν.

Η σύνταξη για την δημιουργία ενός Partner Link είναι όπως παρακάτω:

```
<partnerLinks>
<partnerLink name="ClientStartUpLink"
partnerLinkType="wsdl:ClientStartUpPLT" myRole="Client" />
</partnerLinks>
```

Για να γίνει περισσότερο κατανοητή η έννοια του Partner Link ο αναγνώστης θα πρέπει να το θεωρήσει σαν ένα κανάλι επικοινωνίας στο οποίο η φορά ανταλλαγής της πληροφορίας είναι διπλή ενώ η δυνατότητα κλήσης κάθε συμμετέχοντος στοιχείου είναι επίσης αμφίδρομη . Δηλαδή το process μπορεί να καλέσει την υπηρεσία όπως και η υπηρεσία μπορεί να καλέσει το process. Όπως φαίνεται και στις παραπάνω γραμμές όπου παρουσιάζεται η σύνταξη του Partner Link , κάθε τέτοιο στοιχείο χαρακτηρίζεται από μια ιδιότητα που προσδιορίζει το όνομα του (name) , το τύπο του Partner Link (partnerLinkType) καθώς και το όνομα του ρόλου (myRole) που θα έχει στο process.
Υπενθυμίζεται ότι η δήλωση ενός Partner Link εντός του στοιχείου process το καθιστά global ενώ η αντίστοιχη δήλωση του εντός ενός στοιχείου scope του δίνει τοπικό χαρακτήρα.

Δημιουργία ενός BPEL Process

Σε ένα BPEL process η δήλωση μεταβλητών μπορεί είτε να γίνει εντός του process στοιχείου είτε εντός ενός scope .Όπως και σε μια παραδοσιακή γλώσσα προγραμματισμού έτσι και στην BPEL οι μεταβλητές κρατούν πληροφορίες για τα δεδομένα , τα οποία επεξεργάζονται κατά την διάρκεια εκτέλεσης του προγράμματος . Οι τιμές που περιέχονται στις μεταβλητές μπορεί να είναι δύο ειδών:

- να προέρχονται από ανταλλαγή μηνυμάτων με μια υπηρεσία μιας εξωτερικής επαφής
- να περιέχουν εσωτερικά δημιουργημένη εντός της διαδικασίας πληροφορία.

Για να μπορεί να επιτευχθεί επικοινωνία με εξωτερικές επαφές όλες οι μεταβλητές της BPEL θα πρέπει να είναι είτε WSDL message types, είτε XML schema types είτε XML schema elements ενώ η γλώσσα χειρισμού των μεταβλητών έχει συμβατικά ορισθεί να είναι η XPath 1.0.

Η δήλωση των μεταβλητών μπορεί να γίνει με ένα από τους τρείς παρακάτω τρόπους

```
 <variables>
<variable name="myVar1" messageType="myNS:myWSDLMessageDataType" />
<variable name="myVar1" element="myNS:myXMLElement" />
<variable name="myVar2" type="xsd:string" />
<variable name="myVar3" type="myNS:myComplexType" />
</variables>
```

Δημιουργώντας και Χρησιμοποιώντας Web Services

Τα δομικά στοιχεία της BPEL είναι δραστηριότητες. Υπάρχουν δύο είδη δραστηριοτήτων :

- Οι δομημένες δραστηριότητες που ορίζουν την επιχειρηματική λογική και οι οποίες αποτελούνται από ένα σύνολο άλλων περισσότερο βασικών δραστηριοτήτων.
- Οι βασικές δραστηριότητες που έχουν σαν σκοπό περισσότερο βασικών λειτουργιών (όπως των χειρισμό δεδομένων ή την λήψη ενός μηνύματος από κάποιο partner link) και δεν μπορούν να συμπεριλάβουν άλλες δραστηριότητες.

Όπως έχει γίνει ήδη κατανοητό η BPEL έχει σαν κύριο σκοπό την επικοινωνία και τον συντονισμό μεταξύ Web Services , έτσι στο ρεπερτόριο των εντολών της έχει και μια σειρά από εντολές (activities) οι οποίες πραγματοποιούν την ανταλλαγή των μηνυμάτων μεταξύ της επιχειρηματικής μας διαδικασίας που περιγράφεται με BPEL και των εξωτερικών Web Service.
Αναλυτικότερα αυτές οι activities είναι οι :

Receive. Ο σκοπός μιας receive activity είναι να λαμβάνει μηνύματα από μια εξωτερική επαφή. Για αυτό το λόγο πάντα σε μια receive activity ορίζεται το αναγκαίο Partner link και το operation , ενώ θα πρέπει να ορισθεί επίσης και μια μεταβλητή (ή ένα σύνολο από μεταβλητές) στη οποία θα αποθηκεύονται τα ζητούμενα από την εξωτερική επαφή δεδομένα.

```
<receive name="ReceiveRequestFromPartner"
createInstance="yes"
partnerLink="ClientStartUpPLT"
operation="StartProcess" ... />
```

Η ιδιότητα createInstance="yes" υποδηλώνει ότι χρησιμοποιούμε αυτή την δραστηριότητα δημιουργώντας ένα νέο στιγμιότυπο για την συγκεκριμένη διαδικασία ενώ αν ή τιμή της ιδιότητας ήταν createInstance="no" ,τότε το εισερχόμενο μήνυμα δεν θα δημιουργούσε νέο στιγμιότυπο και θα αναλωνόταν από την ήδη υπάρχον ενεργό στιγμιότυπο.

Reply .Κάθε receive activity μπορεί να συνδυαστεί με μια reply activity στην περίπτωση που θέλουνε να έχουμε μια επικοινωνία με την εξωτερική επαφή της μορφής request-response. Ένα τέτοιο παράδειγμα είναι η παραγγελία ενός βιβλίου από ένα online βιβλιοπωλείο ή η κράτηση αεροπορικών εισιτηρίων από ένα ταξιδιωτικό γραφείο . Η απάντηση που μπορούμε να έχουμε σε αυτές τις περιπτώσεις είναι δύο ειδών , η πρώτη θα αφορά την επιβεβαίωση της παραγγελίας ή της κράτησης ενώ σε περίπτωση μη διαθεσιμότητας (είτε του βιβλίου είτε των εισιτηρίων) ένα μήνυμα fault . Το δεύτερο ενδεχόμενο μας υποχρεώνει να ορίσουμε μια επιπλέον ιδιότητα η οποία θα διαχειρίζεται αυτή την κατάσταση.

```
<reply name="ReplyResponseToPartner"
partnerLink="ClientStartUpPLT"
operation="StartProcess"
variable="ClientAnswer"
faultName="ClientAnswer"…/>
```

Το βασικό χαρακτηριστικό της reply activity είναι η μεταφορά των δεδομένων από την εξωτερική επαφή , αυτό το ρόλο των αναλαμβάνει το στοιχείο variable ενώ σε περίπτωση fault η ιδιότητα faultName στην οποία συσχετίζεται το αντίστοιχο WSDL μήνυμα λάθους.

Invoke. Η δραστηριότητα αυτή χρησιμοποιείται για την κλήση ενός Web Service που παρέχεται από μια εξωτερική επαφή. Απαραίτητα στοιχεία που πρέπει να ορισθούν είναι το partnerLink και ένα operation του web service που θα κληθεί.

```
<invoke name="InvokePartnerWebService"
partnerLink="BusinessPartnerServiceLink"
operation="partnerOperation" ... />
```

Μια invoke δραστηριότητα μπορεί να καλέσει μια μονόδρομή λειτουργία και να μην περιμένει μια απάντηση είτε μια αμφίδρομη ή οποία θα σταματήσει την εξέλιξη του process και θα αναμένει απάντηση από την εξωτερική επαφή. Στην τελευταία περίπτωση θα πρέπει να συμπεριληφθεί στην invoke δύο μεταβλητές , μια εισαγωγής και μια εξαγωγής.

```
<invoke name="RequestResponseInvoke"
partnerLink="BusinessPartnerServiceLink"
operation="RequestResponseOperation"
inputVariable="Input"
outputVariable="Output" />
```

Ενώ στην περίπτωση της μονόδρομης λειτουργίας απαιτείται μόνο μια εισαγωγής.

```
<invoke name="OneWayInvoke "
      partnerLink="BusinessPartnerServiceLink"
operation="OneWayOperation"
inputVariable=”Input”/>
```

Δομώντας την Επιχειρηματική Λογική

Η BPEL για την δόμηση επιχειρηματικών διαδικασιών έχει στο ρεπερτόριο της δομικά στοιχεία δραστηριοτήτων που όπως προαναφέρθηκε μπορούν να συνθέσουν μεγαλύτερες και πολυπλοκότερες δραστηριότητες καθώς επίσης και κλασσικές εντολές προγραμματισμού για τον χειρισμό της ροής του κώδικα.
Αναλυτικότερα για την δόμηση των επιχειρηματικών διαδικασιών μπορούν να χρησιμοποιηθούν οι παρακάτω εντολές-στοιχεία.

sequence

Η ιδιότητα αυτή μας επιτρέπει να συμπεριλάβουμε μια σειρά από άλλες δραστηριότητες οι οποίες θα πρέπει να εκτελεστούν με μια ορισμένη σειρά .

```
<sequence name="InvertMessageOrder">
<receive name="receiveOrder" ... />
<invoke name="checkPayment" ... />
<invoke name="shippingService" ... />
<reply name="sendConfirmation" ... />
</sequence>
```

Στο παραπάνω παράδειγμα οι δραστηριότητες

- receiveOrder ,
- checkPayment,
- shippingService ,
- sendConfirmation

εκτελούνται με την σειρά που αναγράφονται εντός της sequence.

if-else

Μια άλλη δραστηριότητα γνωστή από άλλες παραδοσιακές γλώσσες προγραμματισμού είναι η if-else activity. Όπως ίσως φαντάζεται κανείς η if-else activity μας επιτρέπει να δημιουργήσουμε κλαδιά στην δομή του process ανάλογα αν οι συνθήκες αλήθειας που έχουμε είναι αληθείς η ψευδείς. Για την δόμηση των συνθηκών αληθείας χρησιμοποιείται η XPath.

```
<if name="isOrderBiggerThan5000Dollars">
<condition>
$order &gt; 5000
</condition>
<invoke name="calculateTenPercentDiscount" ... />
<elseif>
```

```
<condition>
$order &gt; 2500
</condition>
<invoke name="calculateFivePercentDiscount" ... />
</elseif>
```

```
<else>
<reply name="sendNoDiscountInformation" ... />
</else>
</if>
```

Στο παραπάνω παράδειγμα η πρώτη συνθήκη αληθείας επαληθεύεται αν το ποσό της παραγγελίας είναι μεγαλύτερο από 5000, όπου θα γίνει invoke της υπηρεσίας calculateTenPercentDiscount, ενώ σε άλλη περίπτωση αν το ποσό της παραγγελίας είναι μεγαλύτερο από 2500 θα κληθεί η υπηρεσία calculateFivePercentDiscount, κοκ.

Επαναλήψεις

Στην BPEL προσφέρονται τρείς δραστηριότητες-εντολές που επιτρέπουν την επαναλαμβανόμενη εκτέλεση της επιχειρηματικής λογικής.

While. Η δραστηριότητα αυτή είναι μια δομημένη δραστηριότητα αποτελούμενη από μια συνθήκη αληθείας και μια υποκείμενη δραστηριότητα . Η επανάληψη της υποκείμενης δραστηριότητας γίνεται όσο η συνθήκη που έχουμε αληθεύει. Η While δραστηριότητα δεν εξασφαλίζει την εκτέλεση της υποκείμενης δραστηριότητας αφού αν η συνθήκη δεν αληθεύει δεν θα εκτελεστεί καθόλου.

```
<while>
<condition>
$iterations &gt; 3
</condition>
<invoke name="increaseIterationCounter" ... />
</while>
```

repeatUntil. Η δραστηριότητα αυτή έχει την διαφορά ότι η υποκείμενη δραστηριότητα θα εκτελεστεί τουλάχιστον μία φορά εφόσον η συνθήκη αληθείας βρίσκεται στο τέλος της δομής της.

```
<repeatUntil>
<invoke name="increaseIterationCounter" ... />
<condition>
$iterations &gt; 3
</condition>
</repeatUntil>
```

forEach. Σε αυτή την δραστηριότητα επιλέγουμε να εκτελέσουμε την υποκείμενη δραστηριότητα (μία η περισσότερες) Ν φορές. Υπάρχουν δύο παραλλαγές της forEach δραστηριότητας , η

παράλληλη εκτέλεση και η σειριακή οι οποίες θα αναπτυχθούν παρακάτω . Ένας περιορισμός που υπάρχει στην δραστηριότητα αυτή είναι ότι ενώ στις άλλες δραστηριότητες μπορούμε να έχουμε σαν υποκείμενες προς επανάληψη άλλες δραστηριότητες , εδώ μπορούμε να έχουμε μόνο μια scope δραστηριότητα.
Μια τυπική forEach φαίνεται παρακάτω:

```
<forEach parallel="no" counterName="N" ...>
<startCounterValue>1</startCounterValue>
<finalCounterValue>5</finalCounterValue>
<scope>
<documentation>check availability of each item ordered</documentation>
<invoke name="checkAvailability" ... />
</scope>
</forEach>
```

Στο παραπάνω παράδειγμα η επανάληψη θα εκτελεστεί 5 φορές , ενώ η εκτέλεση είναι σειριακή , δηλαδή για να ξεκινήσει κάθε νέα επανάληψη θα πρέπει να έχει ολοκληρωθεί η προηγούμενη.

Παράλληλη επεξεργασία

Για την παράλληλη εκτέλεση διαδικασιών υπάρχει η flow activity. Θα πρέπει να σημειωθεί εδώ ότι εκτός από την flow activity, η παραλλαγή της forEach activity και οι χειρισμοί γεγονότων που δημιουργούν πολλαπλά στιγμιότυπα μας επιτρέπουν την δημιουργία παράλληλης εκτέλεσης.
Στο παράδειγμα που ακολουθεί , μια σειρά από τρείς δραστηριότητες (*checkFlight*, *checkHotel* and *checkRentalCar*) εκτελούνται παράλληλα ενώ και οι τρεις αυτές δραστηριότητες ξεκινούν ταυτόχρονα όταν η flow ξεκινά.

```
<flow ...>
<links> ... </links>
<documentation>
check availability of a flight, hotel and rental car concurrently
</documentation>
<invoke name="checkFlight" ... />
<invoke name="checkHotel" ... />
<invoke name="checkRentalCar" ... />
</flow>
```

Μερικές φορές είναι απαραίτητο να συγχρονίζεται η παράλληλη εκτέλεση δραστηριοτήτων .
Αν υποθέσουμε ότι προσθέτουμε άλλη μια δραστηριότητα την *bookFlight* στην παραπάνω flow δραστηριότητα αυτή θα ξεκινήσει παράλληλα με τις άλλες τρεις δραστηριότητες. Ωστόσο όμως

κάτι τέτοιο δεν είναι επιθυμητό εφόσον το να κλείσει κάποιος αεροπορικά εισιτήρια έχει νόημα μόνο αν υπάρχει διαθέσιμό εισιτήριο. Για αυτό τον λόγο μπορούμε να προσθέσουμε ένα link μεταξύ των δύο δραστηριοτήτων. Με αυτόν τον τρόπο δημιουργείται μια εξάρτηση μεταξύ των δύο δραστηριοτήτων κατά την οποία η δραστηριότητα η όποια είναι η τελική θα ξεκινήσει την εκτέλεση της αφού η αρχική δραστηριότητα ολοκληρωθεί .

```
<flow ...>
<links>
<link name="checkFlight-To-BookFlight" />
</links>
<documentation>
check availability of a flight, hotel and rental car concurrently
</documentation>
<invoke name="checkFlight" ...>
<sources>
<source linkName="checkFlight-To-BookFlight" />
</sources>
</invoke>
<invoke name="checkHotel" ...
```

```
<invoke name="checkHotel" ... />
<invoke name="checkRentalCar" ... />
<invoke name="bookFlight" ...>
<targets>
<target linkName="checkFlight-To-BookFlight" />
</targets>
</invoke>
</flow>
```

TransitionCondition. Στα χαρακτηριστικά ενός link υπάρχει μια ιδιότητα που ονομάζεται transitionCondition. Ο σκοπός αυτής της ιδιότητας είναι να προσδιορίσει ποιο link θα κληθεί κάθε φορά ανάλογα της τιμής που θα λάβει η ιδιότητα αυτή. Αν αυτό το χαρακτηριστικό δεν αρχικοποιηθεί λαμβάνει την τιμή αληθές ως προεπιλεγμένη τιμή . Αν μια transitionCondition ορισθεί ,τότε θα ορίσει το status του link.

```
<flow ...>
<links>
<link name="request-to-approve" />
<link name="request-to-decline" />
</links>
<receive name="ReceiveCreditRequest"
createInstance="yes"
partnerLink="creditRequestPLT"
operation="creditRequest"
variable="creditVariable">
<sources>
<source linkName="request-to-approve">
<transitionCondition>
$creditVariable/value &lt; 5000
</transitionCondition>
</source>
<source linkName="request-to-decline">
<transitionCondition>
$creditVariable/value >= 5000
</transitionCondition>
</source>
</sources>
</receive>
<invoke name="approveCredit" ...>
<targets>
```

```
<target linkName="request-to-approve" />
</targets>
</invoke>
<invoke name="declineCredit" ...>
<targets>
<target linkName="request-to-decline" />
</targets>
</invoke>
</flow>
```

Το link *request-to-approve* έχει transition condition το οποίο ελέγχει αν η τιμή της μεταβλητής *creditVariable* έχει τιμή η οποία είναι μικρότερη από 5000. Αν ισχύει αυτό τότε το status request-to-approve link θα πάρει την τιμή αληθές και θα κληθεί το αντίστοιχό target, σε αντίθετη περίπτωση το status του request-to-decline θα λάβει την τιμή αληθές και θα καλέσει το αντίστοιχο target.

Το request-to-decline link (μεγαλύτερο η ίσο από 5000), που σημαίνει ότι μια από τις δύο δραστηριότητες θα εκτελεστεί .

Τα **Transition conditions** προσφέρουν ένα μηχανισμό που επιτρέπει τον διαχωρισμό της ροής της εργασίας σε εναλλακτικές ροές ανάλογα της συνθήκης που ορίζουμε κάθε φορά. Ωστόσο θα πρέπει να υπάρχει και ένας μηχανισμός ο οποίος να μας επιτρέπει να επανένωση της ροή εργασίας μας. Αυτό προσφέρεται μέσα από τι λεγόμενες join conditions που θα δούμε στην συνέχεια.

Οι Join conditions σχετίζονται με δραστηριότητες συνήθως εάν οι δραστηριότητες έχουν κοινά εισερχόμενα links. Μια joinCondition προσδιορίζει για μια δραστηριότητα κάτι σαν μιαν αρχική συνθήκη πχ όλες οι εισερχόμενα link πρέπει να έχουν status true ώστε να εκτελεστεί η δραστηριότητα , ή τουλάχιστον ένα εισερχόμενο link πρέπει να έχει status true. Το παραπάνω φαίνεται στο παράδειγμα που ακολουθεί:

```
<flow ...>
<links>
<link name="request-to-approve" />
<link name="request-to-decline" />
<link name="approve-to-notify" />
<link name="decline-to-notify" />
</links>
<receive name="ReceiveCreditRequest"
createInstance="yes"
partnerLink="creditRequestPLT"
operation="creditRequest"
variable="creditVariable">
<sources>
<source linkName="request-to-approve">
<transitionCondition>
$creditVariable/value &lt; 5000
</transitionCondition>
</source>
<source linkName="request-to-decline">
<transitionCondition>
$creditVariable/value >= 5000
</transitionCondition>
</source>
</sources>
</receive>
<invoke name="approveCredit" ...>
<source linkName="approve-to-notify" />
```

```
<targets>
<target linkName="request-to-approve" />
</targets>
</invoke>
<invoke name="declineCredit" ...>
<source linkName="approve-to-notify" />
<targets>
<target linkName="request-to-decline" />
</targets>
</invoke>
<reply name="notifyApplicant" ...>
<targets>
<joinCondition>
$approve-to-notify or $decline-to-notify
</joinCondition>
<target linkName="approve-to-notify" />
<target linkName="decline-to-notify" />
</targets>
</invoke>
</reply>
</flow>
```

Ας υποθέσουμε ότι συμβαίνει ένα λάθος στην δραστηριότητα *approveCredit*, το εξερχόμενο link της δραστηριότητας αυτής το οποίο είναι το *approve-to-notify* θα πάρει τιμή false από το περιβάλλον εκτέλεσης . Αυτό μπορεί να οδηγήσει σε μια κατάσταση όπου το join condition λαμβάνει τιμή false επίσης . Η WS-BPEL παρέχει δύο μηχανισμούς για να χειρίζεται με τις false join conditions. Εξ ορισμού ένα joinFailure μήνυμα λάθους παράγεται και ένας κατάλληλος fault handler (που θα δούμε στην συνέχεια) καλείται να το χειριστεί ή εναλλακτικά , όταν η ιδιότητα suppressJoinFailure στο process ή σε μια εσωτερική δραστηριότητα λαμβάνει την τιμή yes, η δραστηριότητα που σχετίζεται με το false join condition παρακάμπτεται.

Χειρισμός Δεδομένων

Η assign activity περιέχει μια η περισσότερες λειτουργίες αντιγραφής δεδομένων.
Κάθε λειτουργία αντιγραφής έχει ένα αρχικό σημείο από το οποίο αντιγράφονται τα δεδομένα και ένα τελικό σημείο στο οποίο πάνε και γράφονται (from και to) αντίστοιχα . Στο παράδειγμα που ακολουθεί μια μεταβλητή αντιγράφεται σε μία άλλη μεταβλητή του ίδιου πάντα τύπου.:

```
<assign>
<copy>
```

```
<from variable="TimesheetSubmissionFailedMessage" />
<to variable="EmployeeNotificationMessage" />
</copy>
</assign>
```

Ωστόσο το παραπάνω παράδειγμα δεν επαρκεί για να δείξει τις δυνατότητες της assign . Συχνά υπάρχει η ανάγκη να αντιγράψουμε ένα μόνο κομμάτι μιας μεταβλητής σε ένα άλλο , η ακόμα τμήματα μια μεταβλητής σε ένα μία μόνο μεταβλητή κοκ. Για αυτό το λόγο χρησιμοποιείται η XPath και η XSLT.

```
<assign>
<copy>
<from variable="Input" part="operation1Parameter">
<query>
creditCardInformation
</query>
</from>
<to variable="CreditCardServiceInput" />
</copy>
</assign>
```

Χειρισμός Εξαιρέσεων

Όπως σε κάθε προγραμματιστικό περιβάλλον έτσι και στην BPEL θα πρέπει να υπάρχει η θεωρήσει «τι γίνεται όταν τα πράγματα δεν κυλήσουν όπως τα έχουμε σχεδιάσει». Θα πρέπει να υπάρχει ένας κατάλληλος μηχανισμός ο οποίος θα μας επιτρέπει να χειριστούμε τυχόν εξαιρέσεις στην εξέλιξη της πού εργασίας μας. Στην BPEL προσφέρονται ή έννοια του χειριστή λαθών fault handlers. Ένας χειριστής λαθών μπορεί να συμπεριληφθεί σε ένα scope , σε ένα process, η απευθείας σε μια invoke δραστηριότητα.

```
<faultHandlers>
<catch faultName="BookOutOfStockException"
faultVariable="BookOutOfStockVariable">
...
</catch>
<catchAll>...</catchAll>
</faultHandlers>
```

Όπως μπορεί να φανεί από το παράδειγμα ένας fault handler μπορεί να έχει δύο τύπους παιδιών

- Ένα ή περισσότερα catch ,
- Ένα τουλάχιστον catchAll.

Κάθε catch παρέχει μια δραστηριότητα ή οποία εκτελείται για ένα συγκεκριμένο τύπο εξαίρεσης. Στο παραπάνω παράδειγμα καλείται το web service checkAvailability για να ελεγχθεί αν ένα συγκεκριμένο βιβλίο είναι στο απόθεμα και η απάντηση του web service μπορεί να είναι μια BookOutOfStockException η οποία θα πρέπει να έχει οριστεί στο αντίστοιχο WSDL interface της υπηρεσίας.
Μια δομή catch έχει τις εξής προαιρετικές ιδιότητες :

- faultName το οποίο αναφέρεται στο όνομα του fault το οποίο θα πρέπει να εντοπίσει
- faultVariable .Όταν μια faultVariable χρησιμοποιείται τότε είτε ένα faultMessageType ή faultElement θα πρέπει να προσδιοριστεί και η ιδιότητα αυτή θα αναφερθεί στις ιδιότητες αυτές.

Κατ' επιλογή ένας fault hander μπορεί να τελειώνει με μια δομή catchAll όπως σε όλες τις γλώσσες προγραμματισμού και να αναφέρεται σε όλα εκείνα τα γεγονότα που δεν έχουν προβληθεί από τον προγραμματιστή – αναλυτή.

Web Service Description Language

Εισαγωγή

Η WSDL είναι μια προδιαγραφή που δημιουργήθηκε με σκοπό να περιγράψει υπηρεσίες διαδικτύου (Web Services) σε μορφή XML κειμένων , προσδιορίζοντας επιπρόσθετα και την τοποθεσία που βρίσκονται αυτές οι υπηρεσίες.
Αναλυτικότερα σε ένα WSDL κείμενο περιγράφονται τέσσερα βασικά κομμάτια δεδομένων:

- Πληροφορίες για τις υπηρεσίες που περιγράφονται από το WSDL κείμενο.
- Πληροφορίες τύπων δεδομένων για όλα τα μηνύματα αίτησης και απάντησης της υπηρεσίας.
- Πληροφορίες που αφορούν τον εντοπισμό της συγκεκριμένης υπηρεσίας παρέχοντας την διεύθυνση της.
- Πληροφορίες σχετικά με τα πρωτόκολλά που χρησιμοποιούνται για την μεταφορά.

Η WSDL 2.0 αποτελεί πρότυπο της W3C από τον Ιούνιο του 2007.

Δομή ενός WSDL κειμένου

Όπως προαναφέρθηκε ένα WSDL κείμενο είναι XML γραμματική που χρησιμοποιείται για την περιγραφή Web Services, επομένως η σύνταξη και η δομή του δεν διαφέρει και πολύ από αυτήν ενός απλού XML κειμένου.
Η WSDL δεν θεωρεί ότι η ανταλλαγή της πληροφορίας θα γίνει με ένα συγκεκριμένο τύπο επικοινωνίας, ενώ είναι σχεδόν βέβαιο ότι διαφορετικές υπηρεσίες θα έχουν διαφορετικές διεπαφές (interfaces) και θα βρίσκονται σε διάφορες διευθύνσεις.
Γενικά η δομή ενός WSDL κειμένου αποτελείται από δύο βασικά κομμάτια :

- Το abstract κομμάτι το οποίο περιγράφει:
  - Τα μηνύματα που ανταλλάσσονται .
  - Την λειτουργία που συνδέει ένα πρότυπο ανταλλαγής μηνυμάτων με ένα η περισσότερα μηνύματα.
- Το concrete κομμάτι στο οποίο ορίζονται:

- Την μεταφορά και το τύπο μετάδοσης πάνω στο μέσο για να μία ή περισσότερες διεπαφές.
- Ένα τελικό σημείο (endpoint) που συνδέει μια ηλεκτρονική διεύθυνση ένα σημείο αναφοράς.
- Μια υπηρεσία η οποία ομαδοποιεί τελικά σημεία μιας κοινής διεπαφής.

Συγκεκριμένα ένα WSDL κείμενο χρησιμοποιεί τους παρακάτω τύπους δεδομένων για την περιγραφή ενός Web Service.

Στοιχείο	**Προσδιορίζει**
<types>	Προσδιορίζει τους τύπους δεδομένων που χρησιμοποιούνται από το **Web Service.**
<Interface>	Προσδιορίζει μια σειρά μηνυμάτων που ανταλλάσσονται**.**
<binding>	Καθορίζει τα πρωτόκολλα επικοινωνίας που χρησιμοποιούνται**.**
<service>	Καθορίζει που βρίσκεται η υπηρεσία**.**

Η δομή ενός κειμένου WSDL σε μορφή κώδικα θα είναι όπως παρακάτω.

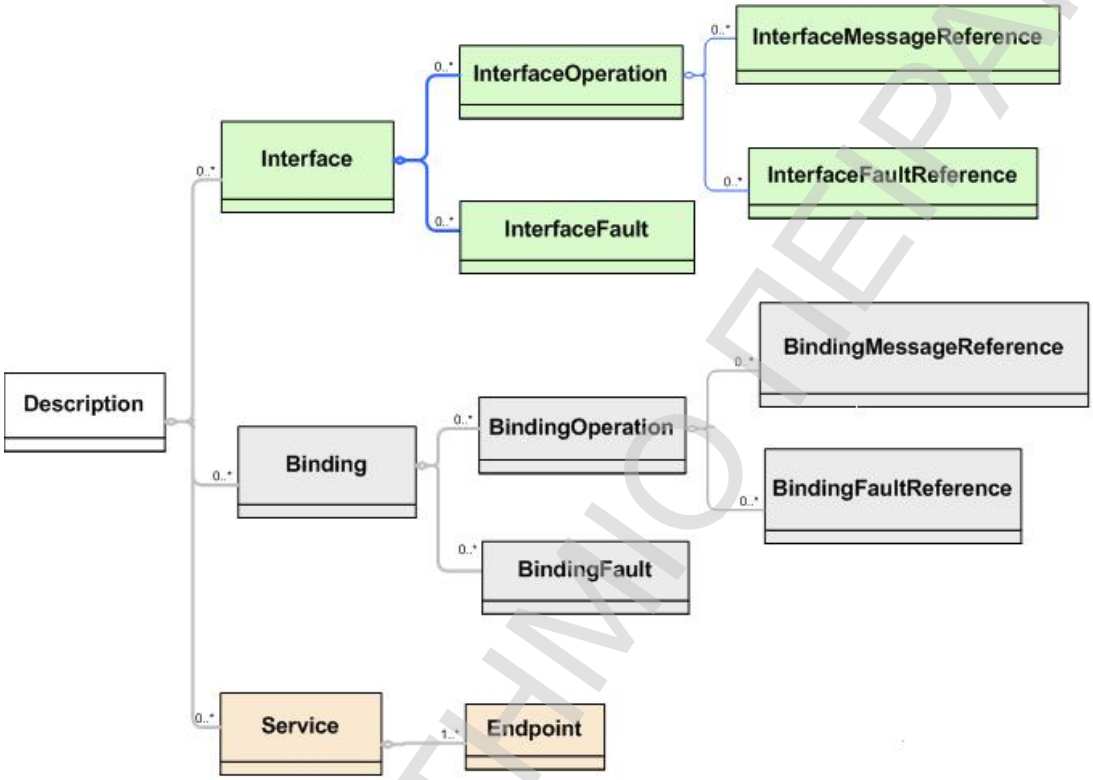
```
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl">
  <wsdl:types/>
  <wsdl:interface/>
  <wsdl:binding/>
  <wsdl:service/>
</wsdl:description>
```

Το στοιχείο <types> περιλαμβάνει όλο το σχήμα XML περιγραφής ενός element και δηλώσεις τύπου που περιγράφουν τα μηνύματα του Web Service.
Η WSDL 2.0 είναι ανοικτή ως πρότυπο για την χρήση και άλλων σχημάτων για την περιγραφεί των μηνυμάτων πέραν αυτού της XML που χρησιμοποιείται ως προεπιλογή.

Το στοιχείο **<Interface>** όπως προαναφέρθηκε περιγράφει μια σειρά μηνυμάτων που ανταλλάσσονται ομαδοποιώντας τα συσχετιζόμενα μηνύματα σε λειτουργίες (operations). Οι λειτουργίες είναι μια ακολουθία εισερχομένων και εξερχόμενων μηνυμάτων , ενώ ένα interface είναι ένα σύνολο από operations.

Το στοιχείο **<binding>** καθορίζει το πρωτόκολλο επικοινωνίας του client με το Web Service, ενώ το στοιχείο **<service>** καθορίζει την διεύθυνση της υπηρεσίας παρέχοντας το URL αυτής καθώς και ένα συγκεκριμένο <interface> όπως και <binding>.

Ενώ ένα αναλυτικό σχήμα πάνω στην δομή ενός WSDL κειμένου είναι αυτό παρακάτω:



Για να γίνει περισσότερο κατανοητή η δομή και ο τρόπος με τον οποίο δομείται ένα WSDL 2.0 κείμενο θα ακολουθήσει ένα παράδειγμα στο οποίο θα εξεταστεί το σύνολο των χαρακτηριστικών της συγκεκριμένης γλώσσας.

Παράδειγμα

Στο παράδειγμα αυτό υποθέτουμε ότι είμαστε στο τμήμα IT ενός ξενοδοχείου με την επωνυμία GreatH . Η διεύθυνση του ξενοδοχείου εξέφρασε την επιθυμία να δημιουργηθεί μια υπηρεσία για την κράτηση δωματίων μέσω Internet.
Το τμήμα μας μετά από μελέτη στο συγκεκριμένο πρόβλημα κατέληξε ότι το Web Service που θα δημιουργήσει θα πρέπει να έχει τις παρακάτω λειτουργικότητες:

- CheckAvailability . Για τον έλεγχο της διαθεσιμότητας δωματίου. Ο πελάτης θα πρέπει να προσδιορίσει την ημερομηνία προσέλευσης (check-in date) , την ημερομηνία αναχώρησης (check-out date) καθώς και τον τύπο του δωματίου που επιθυμεί. Εάν

κάποιο στοιχείο που καταχωρεί είναι άκυρο η υπηρεσία θα επιστρέφει ένα μήνυμα λάθους, ενώ στην περίπτωση που τα εισαγόμενα στοιχεία είναι έγκυρα και υπάρχει διαθεσιμότητα δωματίου στις συγκεκριμένες ημερομηνίες θα επιστρέφει θα επιστρέφεται το αριθμό κινητής υποδιαστολής και σε αντίθετη περίπτωση μηδέν. Άρα η υπηρεσία θα δέχεται ένα μήνυμα checkAvailability και θα επιστρέφει ένα μήνυμα checkAvailabilityResponse ή σε περίπτωση λάθους invalidDataFault μήνυμα.

- MakeReservation. Για την κράτηση ενός δωματίου , ο πελάτης θα πρέπει να καταχωρήσει ένα όνομα , μια διεύθυνση και τον αριθμό της πιστωτικής του κάρτας , ενώ η υπηρεσία θα του επιστρέψει ένα αριθμό επιβεβαίωσης εάν η κράτηση ολοκληρωθεί με επιτυχία. Στην περίπτωση που συμβεί το αντίθετο θα επιστρέφεται ένα μήνυμα λάθους. Άρα η υπηρεσία θα δέχεται σαν είσοδο ένα μήνυμα makeReservation και θα επιστρέφει ένα μήνυμα makeReservationResponse είτε invalidCreditCardFault σε περίπτωση αποτυχίας.

Για λόγους απλότητας και συντομίας θα δοθεί η ελάχιστη λειτουργικότητα στο παράδειγμα μας χωρίς να εξετασθούν θέματα που αφορούν την ασφάλεια και την επεξεργασίας των δεδομένων ενώ παράλληλα θα αναπτυχθεί μόνο η λειτουργία *CheckAvailability.*

Ορίζοντας το Target Namespace

Το *target namespace* ενός WSDL 2.0 κειμένου είναι ένα URI , ανάλογο με το XMLSchema target namespace. Το Interface, binding και service names που θα οριστούν στο κείμενο μας θα σχετίζονται με το συγκεκριμένο target namespace, και έτσι θα προσδιορίζονται μοναδικά από παρόμοια σε διαφορετικά target namespace.

Ο ορισμός του URI θα είναι όπως παρακάτω

Ένα αρχικό WSDL Κείμενο

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    . . . >

  . . .
</description>
```

Εξήγηση του παραδείγματος

`<description`

Κάθε WSDL 2.0 κείμενο έχει ένα στοιχείο description σαν αρχικό στοιχείο. Αυτό λειτουργεί σαν container για το υπόλοιπο κείμενο και χρησιμοποιείται για την δήλωση των namespaces που θα χρησιμοποιηθούν.

`xmlns="http://www.w3.org/ns/wsdl"`

Αυτό είναι XML namespace για τον ορισμό του ίδιου του WSDL 2.0. Κάθε στοιχείο χωρίς prefix στο κείμενο μας θα αναφέρεται σε στοιχείο WSDL 2.0 .

`targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"`

Εδώ ορίζεται το target namespace που επιλέξαμε για την υπηρεσία μας .Η λειτουργία του είναι ανάλογη του σε ένα XML Schema target namespace.

`xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"`

Αυτό είναι ένα πραγματικό ορισμός XML namespace για χρήση της υπηρεσίας μας . Θα πρέπει να σημειωθεί ότι είναι η ίδια τιμή που ορίστηκε παραπάνω.

Ορίζοντας τα Message Types

Για τον ορισμό των μηνυμάτων θα χρησιμοποιήσουμε το XML Schema μιας και αυτό αποτελεί προεπιλογή για την WSDL 2.0 χωρίς αυτό να είναι δεσμευτικό για την χρήση άλλων σχημάτων. Η WSDL 2.0 επιτρέπει τα message types να ορίζονται απευθείας στο WSDL 2.0 κείμενο , μέσα στο types element, το οποίο είναι «παιδί» του description element.

GreatH Message Types

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    . . . >

  ...

  <types>
    <xs:schema
```

```
        xmlns:xs="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://greath.example.com/2004/schemas/resSvc"
        xmlns="http://greath.example.com/2004/schemas/resSvc">

      <xs:element name="checkAvailability" type="tCheckAvailability"/>
      <xs:complexType name="tCheckAvailability">
        <xs:sequence>
          <xs:element  name="checkInDate" type="xs:date"/>
          <xs:element  name="checkOutDate" type="xs:date"/>
          <xs:element  name="roomType" type="xs:string"/>
        </xs:sequence>
      </xs:complexType>

      <xs:element name="checkAvailabilityResponse" type="xs:double"/>

      <xs:element name="invalidDataError" type="xs:string"/>

    </xs:schema>
  </types>
  . . .
</description>
```

Επεξήγηση του παραδείγματος

`xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"`

όπως μπορεί να παρατηρήσει κανείς προστέθηκε άλλο ένας ορισμός namespace.Το ghns namespace prefix μας επιτρέπει να αναφερθούμε στο XML Schema target namespace που θα ορίσουμε τα δικά μας message types. Το URI που ορίζουμε θα πρέπει να είναι το ίδιο με το URI που θα ορίσαμε σαν target namespace για το XML Schema types και όχι με target namespace του ίδιου του WSDL 2.0 κειμένου.

`targetNamespace="http://greath.example.com/2004/schemas/resSvc"`

Αυτό είναι το XML Schema target namespace που δημιουργήσαμε για χρήση από την υπηρεσία μας. Τα στοιχεία (elements) checkAvailability , checkAvailabilityResponse και invalidDataError αναφέρονται σε αυτό το XML Schema target namespace.

`checkAvailability`, `checkAvailabilityResponse` και `invalidDataError`

Αυτά είναι οι τύποι μηνυμάτων που θα χρησιμοποιηθούν , ενώ όπως προαναφέρθηκε ορίστηκαν σαν XML *elements* . Παρόλο όμως που έχουμε ορίσει διάφορους τύπους δεν έχει προσδιοριστεί ακόμα ποιοι από αυτούς τους τύπους θα χρησιμοποιηθούν σαν τύποι μηνυμάτων στην υπηρεσία μας.

Ορίζοντας ένα Interface

Ένα WSDL 2.0 interface ορίζει το abstract κομμάτι μιας υπηρεσίας περιλαμβάνοντας ένα σύνολο από λειτουργίες (operations) , κάθε μια από τις οποίες αντιπροσωπεύει μια απλή διεπαφή μεταξύ του client και της υπηρεσίας. Σε κάθε operation καθορίζονται οι τύποι μηνυμάτων που μπορεί να δεχθεί ή να στείλει η υπηρεσία ως τμήμα του συγκεκριμένου operation. Επίσης κάθε υπηρεσία ορίζει ένα τύπο ανταλλαγής μηνυμάτων που καθορίζει την σειρά με την οποία θα γίνει η ανταλλαγή των μηνυμάτων μεταξύ των συμμετεχόντων μερών. Για την υπηρεσία GreatH , θα ορίσουμε αρχικά ένα interface το οποίο θα περιέχει μια απλή λειτουργία , την opCheckAvailability, η οποία θα χρησιμοποιεί του τύπους μηνυμάτων checkAvailability και checkAvailabilityResponse που ορίσαμε νωρίτερα στο τμήμα types. Επιπλέον των μηνυμάτων αυτών που θα λειτουργούν στο πρότυπο request-response , θα πρέπει να οριστεί επίσης και ένα μήνυμα χειρισμού λαθών . Η WSDL 2.0 επιτρέπει μηνύματα λάθους να ορίζονται εντός ενός interface με σκοπό την επαναχρησιμοποίηση στο σύνολο των operations του συγκεκριμένου interface. .

GreatH Interface Definition

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    . . .
    xmlns:wsdlx="http://www.w3.org/ns/wsdl-extensions">

  . . .
  <types>
        ...
  </types>

  <interface  name = "reservationInterface" >

    <fault name = "invalidDataFault"
            element = "ghns:invalidDataError"/>
```

```
    <operation name="opCheckAvailability"
            pattern="http://www.w3.org/ns/wsdl/in-out"
            style="http://www.w3.org/ns/wsdl/style/iri"
            wsdlx:safe = "true">
        <input messageLabel="In"
              element="ghns:checkAvailability" />
        <output messageLabel="Out"
              element="ghns:checkAvailabilityResponse" />
        <outfault ref="tns:invalidDataFault" messageLabel="Out"/>
    </operation>

  </interface>

  . . .
</description>
```

Επεξήγηση του παραδείγματος

```
<interface name = "reservationInterface" >
```

Το στοιχείο interface ορίζεται μέσα στο στοιχείο description . Μπορούμε να ορίσουμε όσα στοιχεία interface θέλουμε , όμως κάθε ένα θα πρέπει να έχει ένα μοναδικό όνομα που θα το προσδιορίζει μέσα στο συγκεκριμένο WSDL 2.0 target namespace. Τα ονόματα που δίνουμε δεν θα πρέπει να περιέχουν κενά ή (":").

<fault name = "invalidDataFault"

Η ιδιότητα name ορίζει ένα όνομα για το χειρισμό λαθών. Το όνομα απαιτείται ώστε όταν ορίζουμε ένα operation να μπορούμε να αναφερθούμε σε αυτό .Τέλος το όνομα που θα δώσουμε πρέπει να είναι μοναδικό μέσα στο interface.

element = "ghns:invalidDataError"/>

Η ιδιότητα element ορίζει το σχήμα του μηνύματος λάθους που ορίστηκε νωρίτερα στο τμήμα types.

<operation name="opCheckAvailability"

Η ιδιότητα name ορίζει ένα όνομα για το operation ,ώστε να μπορούμε αν αναφερθούμε σε αυτό αργότερα όταν ορίζονται τα bindings. Τα ονόματα των operation θα πρέπει να είναι μοναδικά μέσα σε ένα interface.

pattern="http://www.w3.org/ns/wsdl/in-out"

Σε αυτό το τμήμα του κώδικα ορίζουμε ότι θα χρησιμοποιήσουμε το τρόπο επικοινωνίας in-out δηλαδή επικοινωνία request-response .

style="http://www.w3.org/ns/wsdl/style/iri"

Σε αυτό το τμήμα του κώδικα φαίνεται ότι το XML schema ορίζει ότι το εισερχόμενο μήνυμα ακολουθεί ένα σύνολο κανόνων όπως αυτοί καθορίζονται στο IRI Style εξασφαλίζοντας ότι το μήνυμα μπορεί να σειριοποιηθεί σαν IRI.

wsdlx:safe="true" >

Αυτό το τμήμα κώδικα σημαίνει ότι το operation δεν είναι υποχρεωτικό και ότι ο πελάτης δεν είναι υποχρεωμένος π.χ να κλείσει ένα δωμάτιο ξενοδοχείου ή να αγοράσει κάτι.

<input messageLabel="In"

Το στοιχείο input ορίζει ένα μήνυμα εισαγωγής .Παρόλο που ορίσαμε τον τρόπο ανταλλαγής της πληροφορίας , ο τρόπος αυτός είναι γενικός και θεωρητικά μπορεί να αποτελείται από πολλά μηνύματα εισόδου , έτσι είναι ανάγκη να προσδιορίσουμε μοναδικά το μήνυμα εισόδου.

element="ghns:checkAvailability" />

Εδώ ορίζεται ο τύπος μηνύματος για το μήνυμα εισόδου όπως προηγουμένως ορίστηκε στο τμήμα types.

<output messageLabel="Out" . . .

Παρόμοια με το μήνυμα εισόδου.

<outfault ref="tns:invalidDataFault" messageLabel="Out"/>

Εδώ συσχετίζεται ένα εξερχόμενο μήνυμα λάθους με το operation. Τα μηνύματα ορίζονται λίγο διαφορετικά από τα κανονικά μηνύματα. Η ιδιότητα ref αναφέρεται στο όνομα ενός προηγούμενα ορισμένου λάθους στο interface και όχι σε ένα τύπο προσδιορισμένο από κάποιο σχήμα .Επειδή κατά την σειρά εκτέλεσης της ανταλλαγής της πληροφορίας ένα λάθος μπορεί να εμφανιστεί σε διάφορα σημεία της και επειδή ίσως να υπάρχει η ανάγκη να συσχετιστούν διαφορετικά μηνύματα , χρησιμοποιείται το messageLabel για να δείξει το επιθυμητό μήνυμα

που θα δώσει το λάθος ή της θα αντικαταστήσει αυτό το λάθος , ανάλογα τον τύπο ανταλλαγής της πληροφορίας .

Ορίζοντας ένα Binding

Παρόλο που έχουν οριστεί τα abstract μηνύματα που θα ανταλλάσσονται από την υπηρεσία μας , δεν έχει οριστεί ακόμα το πώς θα ανταλλάσσονται αυτά τα μηνύματα . Αυτό είναι και ο σκοπός του *binding*. Ένα binding ορίζει επακριβώς το format των μηνυμάτων και λεπτομέρειες για το πρωτόκολλο μεταφοράς για ένα interface, και θα πρέπει να παρέχει αυτά τα στοιχεία για κάθε operation και fault σε ένα interface.

Στην γενική περίπτωση , οι λεπτομέρειες οι λεπτομέρειες ενός binding για κάθε operation και fault είναι ορισμένες χρησιμοποιώντας τα στοιχεία operation and fault μέσα σε στοιχείο binding , όπως φαίνεται και στο παρακάτω παράδειγμα. Ωστόσο , σε κάποιες περιπτώσεις είναι πιθανόν να χρησιμοποιηθούν defaulting rules για να παρέχουμε αυτή την πληροφορία. The WSDL 2.0 SOAP binding extension, for example, defines some defaulting rules for operations.

For the GreatH service, we will use SOAP 1.2 as our concrete message format and HTTP as our underlying transmission protocol, as shown below.

GreatH Binding Definition

```
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  . . .

  <types>
    . . .
  </types>

  <interface  name = "reservationInterface" >
       ...
  </interface>

  <binding name="reservationSOAPBinding"
         interface="tns:reservationInterface"
```

```
          type="http://www.w3.org/ns/wsdl/soap"

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">

    <operation ref="tns:opCheckAvailability"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>

    <fault ref="tns:invalidDataFault"
      wsoap:code="soap:Sender"/>

  </binding>

  . . .
</description>
```

Επεξήγηση του παραδείγματος

xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"

Προσθέσαμε άλλους δύο ορισμούς namespace. Ο παραπάνω αφορά το namespace για SOAP 1.2 binding extension . Τα Elements and attributes με πρόθεμα wsoap: ορίζονται σε αυτό.

xmlns:soap="http://www.w3.org/2003/05/soap-envelope"

Αυτό το namespace ορίζεται μόνο του από SOAP 1.2 specification. Το SOAP 1.2 specification ορίζει συγκεκριμένους όρους εντός αυτού του namespace ώστε να ορίζονται μοναδικά συγκεκριμένα θέματα . όταν αναφερόμαστε σε αυτά τα θα χρησιμοποιείται το πρόθεμα soap: .

<binding name="reservationSOAPBinding"

Τα bindings ορίζονται απευθείας μέσα στο στοιχείο description. Η ιδιότητα name ορίζει το όνομα του . Κάθε όνομα θα πρέπει να είναι μοναδικό ανάμεσα στα άλλα bindings σε αυτό το WSDL 2.0 target namespace, και θα χρησιμοποιηθούν αργότερα όταν ορίζεται ένα service endpoint που αναφέρεται σε αυτό το binding.

interface="tns:reservationInterface"

Αυτό είναι το όνομα του interface του οποίου το message format και το πρωτόκολλό μεταφοράς ορίζουμε.

type="http://www.w3.org/ns/wsdl/soap"

Εδώ ορίζεται το message format που χρησιμοποιείται σε αυτήν την περίπτωση το SOAP 1.2.

wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"

Εδώ ορίζεται το πρωτόκολλο μεταφοράς που τα χρησιμοποιηθεί , που στην προκειμένη περίπτωση είναι το HTTP.

<operation ref="tns:opCheckAvailability"

Εδώ δεν ορίζεται ένα νέο operation αλλά αναφέρεται στο προηγούμενο ορισμένο opCheckAvailability με σκοπό να καθοριστούν λεπτομέρειες που αφορούν το binding .

wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response">

Η παραπάνω γραμμή κώδικα ορίζει το πρότυπο ανταλλαγής μηνυμάτων the SOAP (MEP) που θα χρησιμοποιηθεί για να to implement the abstract WSDL 2.0 message exchange pattern (in-out) that was specified when the opCheckAvailability operation was defined.

Όταν χρησιμοποιείται τοHTTP σαν πρωτόκολλό μεταφοράς η ιδιότητα wsoap:mep ελέγχει επίσης εάν χρησιμοποιηθεί είτε η μέθοδος GET ή η POST σαν μέθοδο HTTP . Στην περίπτωση μας η χρήση του wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"

Δηλώνει ότι θα χρησιμοποιηθεί η GET .

<fault ref="tns:invalidDataFault"

Εδώ δεν δηλώνεται ένα fault αλλά αναφέρεται σε ένα προηγούμενα ορισμένο fault (invalidDataFault) στο opCheckAvailability interface, με σκοπό να οριστούν οι λεπτομέρειες του binding για αυτό.

wsoap:code="soap:Sender"/>

Σε αυτό το σημείο ορίζεται ο SOAP 1.2 κώδικας χειρισμού του fault και προσδιορίζεται το τι θα προκαλέσει το μήνυμα λάθους να σταλεί..

Ορίζοντας ένα Service

Έχοντας ορίσει μέσα από το binding πώς τα μηνύματα θα μεταδίδονται ,πρέπει να ορίσουμε το που η υπηρεσία μπορεί να αναζητηθεί με την χρήση του στοιχείου `service`.

Ένα WSDL 2.0 *service* στοιχείο ορίζει μια απλή διεπαφή την οποία θα υποστηρίζει η υπηρεσία και μια σειρά από *endpoint* locations (τελικές τοποθεσίες) από τις οποίες η υπηρεσία μπορεί να

κληθεί. Κάθε endpoint θα πρέπει να έχει αναφερθεί προηγουμένως στο binding ως προς το ποια πρωτόκολλα θα χρησιμοποιηθούν για την μεταφορά των δεδομένων αλλά και για το format τους στο συγκεκριμένο endpoint . Ένα service επιτρέπεται να έχει ένα και μοναδικό interface.

GreatH Service Definition

```
<?xml version="1.0" encoding="utf-8" ?>
<description
    xmlns="http://www.w3.org/ns/wsdl"
    targetNamespace= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:tns= "http://greath.example.com/2004/wsdl/resSvc"
    xmlns:ghns = "http://greath.example.com/2004/schemas/resSvc"
    xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  . . .

  <types>
    . . .
  </types>

  <interface  name = "reservationInterface" >
    . . .
  </interface>

  <binding name="reservationSOAPBinding"
          interface="tns:reservationInterface"
        . . . >
    . . .
  </binding>

  <service name="reservationService"
       interface="tns:reservationInterface">

     <endpoint name="reservationEndpoint"
               binding="tns:reservationSOAPBinding"
               address ="http://greath.example.com/2004/reservation"/>

  </service>

</description>
```

Επεξήγηση του παραδείγματος

<service name="reservationService"

Εδώ ορίζεται ένα όνομα για την υπηρεσία , το οποίο θα πρέπει να είναι μοναδικό ανάμεσα στα άλλα ονόματα στο WSDL 2.0 target namespace.Ο ορισμός ενός ονόματος είναι υποχρεωτικός.

interface="tns:reservationInterface">

Με την παραπάνω γραμμή κώδικα ορίζεται το endpoint που το service μας θα υποστηρίζει και που έχουμε ορίσει προηγούμενος.

<endpoint name="reservationEndpoint"

Εδώ ορίζεται ένα endpoint για το service, και ένα όνομα γι' αυτό το οποίο θα πρέπει να είναι μοναδικό εντός της υπηρεσίας .

binding="tns:reservationSOAPBinding"

Στο binding ορίζουμε το προηγούμενος ορισμένο binding που θα χρησιμοποιηθεί από το endpoint.

address ="http://greath.example.com/2004/reservation"/>

Εδώ ορίζεται η διεύθυνση στην οποία θα «ακούει» η υπηρεσία μας με την χρήση του binding που ορίσαμε στην σχετική ιδιότητα.

Τεκμηρίωση του Service

Το στοιχείο documentation μας επιτρέπει σε οποιοδήποτε σημείο του κώδικά μας να προσθέσουμε πληροφορίες για την λειτουργικότητα της υπηρεσίας μας η οποία από την οποία ο κάθε προγραμματιστής μπορεί να βγάλει χρήσιμα συμπεράσματα. Παρακάτω ακολουθεί ένα παράδειγμα χρήσης του στοιχείου documentation

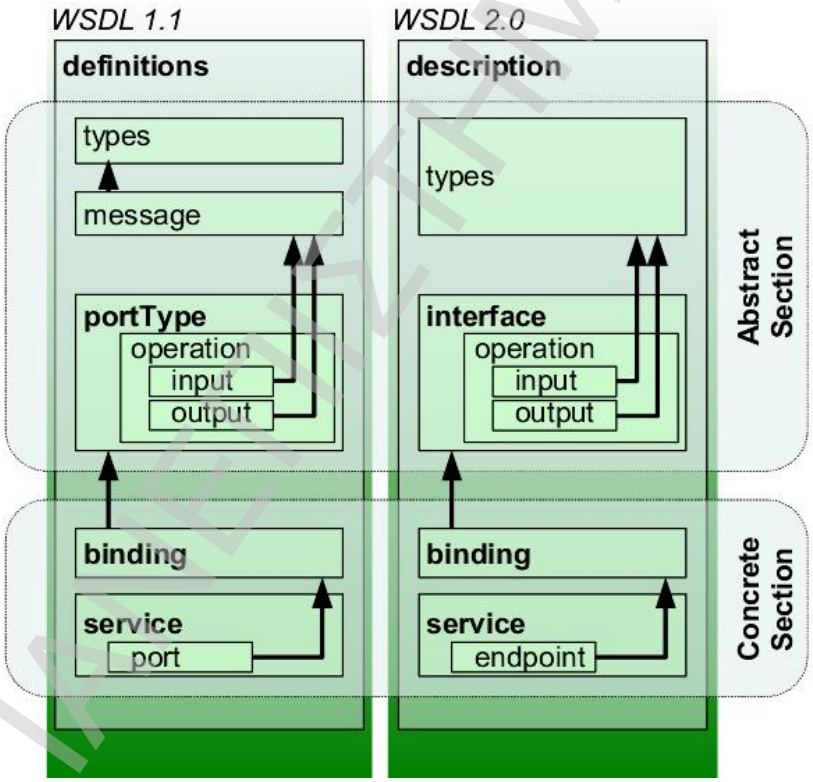
Documenting the GreatH Service

```
<?xml version="1.0" encoding="utf-8" ?>
<description
   . . . >

  <documentation>
    This document describes the GreatH Web service.  Additional
```

```
    application-level requirements for use of this service --
    beyond what WSDL 2.0 is able to describe -- are available
    at http://greath.example.com/2004/reservation-documentation.html
  </documentation>
  . . .
</description>
```

WSDL 1.1 vs WSDL 2.0

Η παρούσα έκδοσή της WSDL είναι η 2.0 η οποία όπως αναφέρθηκε και στην αρχή απαιτεί πρότυπο της W3C από το 2007 και είναι αυτή στην οποία αναφέρεται και το παρόν εγχειρίδιο. Παρόλα αυτά η πρώτη έκδοση της WSDL ήταν η 1.1 . Η βασική διαφορά μεταξύ των δύο εκδόσεων είναι ότι η 1.1 χρησιμοποιούσε για την δημιουργία του binding μόνο τις μεθόδους POST και GET της HTTP ενώ η 2.0 τις χρησιμοποιεί όλες. Ωστόσο η υποστήριξη με εργαλεία ανάπτυξης για την έκδοση 2.0 είναι ακόμα μικρή ενώ αντίθετα για την 1.1 αρκετά μεγάλη. Στην εικόνα που ακολουθεί φαίνονται και οι διαφορές ανάμεσα στις δύο εκδόσεις όσον αφορά την δομή και των ορισμό των στοιχείων τους.



Τα αντικείμενα στην WSDL 1.1/WSDL 2.0

Service/Service: Το στοιχείο service μπορεί να θεωρηθεί σαν ένα «δοχείο» στο οποίο για ένα σύνολο από συναρτήσεις οι οποίες παρέχονται μέσω web πρωτοκόλλων. Η χρήση του συγκεκριμένου στοιχείου είναι η ίδια και για τις δύο εκδόσεις.

Port/Endpoint: Το στοιχείο port στην WSDL 1.1 endpoint στην WSDL 2.0 ορίζει την διεύθυνση ή την σύνδεση σε ένα web service. Συνήθως παρουσιάζεται με ένα http url string.

Binding/Binding: Ορίζει το interface, ορίζει το SOAP binding style (RPC/Document) και το πρωτόκολλο μεταφοράς (SOAP Protocol). Εδώ επίσης ορίζονται και τα operations.

PortType/Interface: Το στοιχείο portType, που μετονομάστηκε σε interface στην WSDL 2.0, ορίζει web service, τα operations που θα εκτελούνται καθώς επίσης και τα messages που χρησιμοποιούνται από τα operation.

Operation/Operation: Κάθε operation μπορεί να συγκριθεί με μια μέθοδο ή function σε παραδοσιακές γλώσσες προγραμματισμού.

Message/N.A.: Το στοιχείο message υπάρχει μόνο στην έκδοση 1.1 και αναφέρεται σε ένα operation. Το message περιέχει πληροφορία η οποία χρειάζεται για να εκτελεστεί το operation. Κάθε message αποτελείται από ένα ή περισσότερα parts. Κάθε part σχετίζεται με μια ιδιότητα του message και αποτελούν λογικές περιγραφές του περιεχομένου ενός message. Το όνομα ενός message παρέχει ένα μοναδικό αναγνωριστικό για αυτό το message που το διαφοροποιεί από άλλα message. Το στοιχείο message έχει αφαιρεθεί από την WSDL 2.0, όπου απλά ορίζονται το XML schema types για τον ορισμό μηνυμάτων εισόδου , εξόδου και λαθών.

Types/Types: Με το στοιχείο αυτό περιγράφονται τα δεδομένα και στις δύο εκδόσεις και γι' αυτό το σκοπό χρησιμοποιείται το XML Schema .