ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

# ΓΛΩΣΣΕΣ ΚΑΙ ΠΛΑΤΦΟΡΜΕΣ ΣΥΝΘΕΣΗΣ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΗΡΕΣΙΩΝ

**Σταμάτης Καρβουναράκης**

Μεταπτυχιακή Εργασία

Ηράκλειο, Μάρτιος 2004

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

# ΓΛΩΣΣΕΣ ΚΑΙ ΠΛΑΤΦΟΡΜΕΣ ΣΥΝΘΕΣΗΣ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΗΡΕΣΙΩΝ

Εργασία που υποβλήθηκε από τον

**Σταμάτη Καρβουναράκη**

ως μερική εκπλήρωση των απαιτήσεων για την απόκτηση

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΕΙΔΙΚΕΥΣΗΣ

Συγγραφέας:

_____
Σταμάτης Καρβουναράκης
Τμήμα Επιστήμης Υπολογιστών

Εισηγητική Επιτροπή:

_____
Βασίλης Χριστοφίδης, Επίκουρος Καθηγητής, Επόπτης

_____
Δημήτρης Πλεξουσάκης, Αναπληρωτής Καθηγητής, Μέλος

_____
Rick Hull, Διευθυντής Ερευνών Bell Labs, Μέλος

Δεκτή:

_____
Δημήτρης Πλεξουσάκης, Αναπληρωτής Καθηγητής,
Πρόεδρος Επιτροπής Μεταπτυχιακών Σπουδών

Ηράκλειο, Νοέμβριος 2002

# ΓΛΩΣΣΕΣ ΚΑΙ ΠΛΑΤΦΟΡΜΕΣ ΣΥΝΘΕΣΗΣ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΗΡΕΣΙΩΝ

Σταμάτης Καρβουναράκης

Μεταπτυχιακή Εργασία

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

## Περίληψη

Ο συνεχώς αυξανόμενος ανταγωνισμός της αγοράς επιβάλει στις εταιρίες να επιζητούν όλο και μεγαλύτερη αποδοτικότητα και παραγωγικότητα, χαρακτηριστικά που επιτυγχάνονται με την υιοθέτηση του μοντέλου ροών εργασίας για την σχεδίαση και εκτέλεση των επιχειρησιακών τους διαδικασιών. Παράλληλα η εκμετάλλευση του ανερχόμενου «παραδείγματος» Ηλεκτρονικών Υπηρεσιών τις βοήθησε να «εξάγουν» «δευτερεύοντες» επιχειρησιακούς τους στόχους σε άλλους οργανισμούς που αποτελούν τους «επιχειρησιακούς τους συνεργάτες» επικεντρώνοντας σε ένα κύριο «επιχειρησιακό στόχο». Κατά αυτό τον τρόπο οι λειτουργίες μίας επιχείρησης, ενός οργανισμού ή απλά μίας εφαρμογής γίνονται προσβάσιμες μέσω του Διαδικτύου με την μορφή Ηλεκτρονικών Υπηρεσιών

Οι Ηλεκτρονικές Υπηρεσίες μπορούν να χωριστούν σε δύο μεγάλες κατηγορίες, τις διακριτές (π.χ. προσθήκη ενός αντικειμένου σε ένα «καλάθι αγορών, χρέωση μίας πιστωτικής κάρτας κ.τ.λ) και τις Ηλεκτρονικές Υπηρεσίες που αναπτύσσουν συνεδρίες (π.χ. τηλεδιάσκεψη, συνεργατική πλοήγηση στο Διαδίκτυο, αλληλεπιδραστικά παιχνίδια κ.λ.π). Οι διακριτές Ηλεκτρονικές Υπηρεσίες έχουν τυπικά μικρή διάρκεια και δεν αποκρίνονται σε εξωτερικά ασύγχρονα γεγονότα. Αντίθετα οι Ηλεκτρονικές Υπηρεσίες που αναπτύσσουν συνεδρίες έχουν μεγάλη διάρκεια και έχουν την ικανότητα τόσο να αποκρίνονται σε ασύγχρονα γεγονότα (π.χ. την προσθήκη ενός συμμετέχοντα σε μία ήδη ενεργή τηλεδιάσκεψη) όσο και την ιδιότητα να δημιουργούν ασύγχρονα γεγονότα (π.χ. όταν ένας συμμετέχων

αποφασίσει να εγκαταλείψει την τηλεδιάσκεψη ή χάσει την σύνδεση του). Οι περισσότερες γλώσσες σύνθεσης Ηλεκτρονικών Υπηρεσιών υποστηρίζουν τον ορισμό μίας διαδικασίας υψηλού επιπέδου η οποία χειρίζεται τις κλήσεις των «επιμέρους» Ηλεκτρονικών Υπηρεσιών σαν ανεξάρτητες μεταξύ τους ενέργειες. Ενώ αυτό το μοντέλο αρκεί στην περίπτωση των διακριτών Ηλεκτρονικών Υπηρεσιών, δεν ισχύει το ίδιο στην περίπτωση που γίνεται σύνθεση Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες. Στην περίπτωση των Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες η «μηχανή εκτέλεσης» πρέπει να «αποκρίνεται» σε ασύγχρονα γεγονότα και να αποφασίζει πώς αυτά τα γεγονότα θα επηρεάζουν τις εν'εξελίξει συνεδρίες (για παράδειγμα αν ένας συμμετέχων σε μία ηλεκτρονική τηλεδιάσκεψη χάσει την σύνδεσή του, ίσως οι εναπομείναντες πρέπει να ειδοποιηθούν για αυτό με ένα γραπτό μήνυμα).

Η εργασία αυτή μελετάει και επεκτείνει ένα μοντέλο διαδικασιών και μία πλατφόρμα σύνθεσης και εκτέλεσης τόσο διακριτών όσο και Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες. Αντίθετα με τις υπόλοιπες προτάσεις, το μοντέλο «ενεργών διαγραμμάτων ροής» που μελετάμε δεν περιγράφει τον ορισμό μίας υψηλού επιπέδου διαδικασίας αλλά περιγράφει το «σχήμα» μίας διαδικασίας σαν ένα σύνολο απο ενεργά διαγράμματα ροής, δηλαδή διαγράμματα ροής «ζευγαρωμένα» με τον τύπο του γεγονότος που τα ενεργοποιεί. Η ενεργοποίηση ενός διαγράμματος ροής μπορεί να έχει αντίκτυπο στην εξέλιξη μίας ενεργής συνεδρίας τροποποιώντας την κατάσταση της ή και τερματίζοντας την. Η πλατφόρμα εκτέλεσης Ηλεκτρονικών Υπηρεσιών AZTEC είναι οδηγούμενη απο γεγονότα, υποστηρίζει τον ρητό ορισμό προτεραιοτήτων στην εκτέλεση των ενεργών διαγραμμάτων ροής και επιτρέπει την εκτέλεση πολλών διαδικασιών ταυτόχρονα. Οι μετρήσεις απόδοσης που εκτελέσαμε δείχνουν ότι η πλατφόρμα AZTEC επιδεικνύει σταθερή συμπεριφορά στον χρόνο απόκρισης τόσο με μικρούς όσο και με μεγάλους ρυθμούς λήψης γεγονότων.

**Επόπτης**: Βασίλης Χριστοφίδης,

Επίκουρος Καθηγητής

# E-SERVICES COMPOSITION LANGUAGES AND PLATFORMS

Stamatis Karvounarakis

Master's Thesis

Computer Science Department, University of Crete

## Abstract

The growing market competition enforces companies to pursue increased efficiency and productivity, which is achieved by adopting workflow models for designing and executing their business processes. At the same time exploitation of the arising E-Service paradigm helps companies to focus on a principal business goal "outsourcing" secondary business goals to other organizations which constitute their "business partners". This way operations of an enterprise or an organization become accessible via the web in the form of E-Services.

E-Services can be distinguished into two broad categories, discrete (i.e add an object in a shopping cart, charge a credit card e.t.c) and session-oriented E-Services (i.e. teleconference, collaborative web browsing, interactive games e.t.c). Discrete E-Services have typically short duration and do not respond to asynchronous events. In the other hand session-oriented e-services have long duration and also the ability to both create (i.e. when a participant drops out of a teleconference) and respond (i.e. add a new participant to an already active teleconference session) to asynchronous events. Most of the E-Service composition languages support the definition of a top-level process which handles the interactions with the various E-Services as independent actions. Even though this model suffices in the case of discrete E-Services this is not the case for session-oriented E-Services. In the case of session-oriented E-Services the "execution engine" must be able to "respond" to asynchronous events and

decide how they affect the active sessions (i.e. when a participant in an audio conference loses his connection the rest of the participants may have to be notified with a text message).

This thesis studies and extends a process model and a composition and execution platform for both discrete and session-oriented E-Services. In contrast to other models the “active flowchart model” defines a process schema as a collection of “active flowcharts”, that is flowcharts coupled with the event class that enacts them, rather than defining a top-level process. A flowchart enactment can affect the state of a session modifying or event terminating it. The AZTEC platform for E-Service composition and execution is event-driven, supports explicit prioritization in the execution of active flowcharts and allows the concurrent execution of both multiple process instances as well as multiple flowcharts of the same process instance. Finally performance measures show that the system exhibits stable behaviour and low response times for both small and high event creation arrival rate.

**Supervisor**: Vassilis Christophides,

Assistant Professor

*Αφιερωμένο στην οικογένεια μου*

# Ευχαριστίες

Ολοκληρώνοντας αυτή την εργασία θα ήθελα να ευχαριστήσω θερμά τον επόπτη καθηγητή μου κ. Βασίλη Χριστοφίδη ο οποίος δεν άφησε σε καμία στιγμή το μυαλό μου να εφησυχαστεί και ήταν πάντα εκεί με νέα ερεθίσματα για σκέψη, προβληματισμό (και δουλειά). Θέλω να ευχαριστήσω επίσης τον κ. Rick Hull αφενός γιατί είχα την τιμή να τον έχω στην εισηγητική μου επιτροπή και αφετέρου γιατί οι συζητήσεις μαζί του τις μέρες της διαμονής του στην Κρήτη ήταν κάτι παραπάνω από επικοδομητικές. Ακόμα θέλω να ευχαριστήσω τον κ. Δημήτρη Πλεξουσάκη, με τον οποίο είχαμε μία πολύ καλή συνεργασία όλα αυτά τα χρόνια, για την συμμετοχή του στην εισηγητική μου επιτροπή. Ένα ευχαριστώ αξίζει και στον Λευτέρη Σιδηρουργό ο οποίος συνέβαλε στην ολοκλήρωση της εργασίας μου και έκανε αρκετές πολύ χρήσιμες παρατηρήσεις. Ευχαριστώ επίσης το Εργαστήριο Πληροφοριακών Συστημάτων του Ιδρύματος Τεχνολογίας και Έρευνας για την «φιλοξενία» του όλο αυτό τον καιρό.

Θα ήθελα να πω ένα πολύ μεγάλο ευχαριστώ σε όλους τους φίλους μου για την συμπαράσταση και την συντροφιά τους. Αν επιχειρήσω να τους κατονομάσω πιθανότατα θα αδικήσω κάποιους, γι'αυτό θα ευχαριστήσω «ονομαστικά» μόνο τον Γιώργο Ζαχαριουδάκη και τον Λευτέρη Κουμάκη με τους οποίους μας δένει μακρόχρονη φιλία, χωρίς πάντως να μειώνω αυτά που νιώθω για τους υπόλοιπους.

Σαν ελάχιστο δείγμα σεβασμού και ευγνωμοσύνης για όλες τις θυσίες που έχουν κάνει για μένα και για την ψυχική και υλική τους συμπαράσταση θέλω να αφιερώσω αυτή την εργασία στους γονείς μου Μιχάλη και Αριέττα. Επίσης θέλω να την αφιερώσω στα αγαπημένα μου αδέρφια Γρηγόρη, Γιώργο και Κρύστη. Ο Γρηγόρης ήταν και εξακολουθεί να είναι πρότυπο για μένα και ήταν πάντα παρόν στις δυσκολότερες στιγμές της φοιτητικής μου ζωής. Ο Γιώργος είναι πάντα πηγή γέλιου και συμπαράστασης ενώ η Κρύστη είναι απλά η αγαπημένη μας αδερφούλα.

# Περιεχόμενα

# Κατάλογος Εικόνων

# *Κεφάλαιο 1*

# *Εισαγωγή*

## *1.1 Ροές Εργασίας και Ηλεκτρονικές Υπηρεσίες*

Ο συνεχώς αυξανόμενος ανταγωνισμός της αγοράς επιβάλει στις επιχειρήσεις και τους οργανισμούς να αυξήσουν την αποδοτικότητα και παραγωγικότητα τους για να μπορούν να ανταποκρίνονται στις απαιτήσεις των πελατών τους. Αυτή η ανάγκη τους ώθησε παραδοσιακά στην χρήση Ροών Εργασίας για την οργάνωση και εκτέλεση των επιχειρησιακών τους διαδικασιών. Χρησιμοποιώντας το μοντέλο ροών εργασίας επιτυγχάνεται ο επιζητούμενος αυτοματισμός της εκτέλεσης των ενεργειών ελαχιστοποιώντας έτσι τον «χαμένο» χρόνο. Παράλληλα επιτυγχάνεται ο μεγαλύτερος δυνατός βαθμός «παραλληλισμού» στην εκτέλεση των διαφόρων ενεργειών εκμεταλλευόμενοι έτσι τις δυνατότητες που παρέχουν τα σύγχρονα Πληροφοριακά Συστήματα.

Ακολουθώντας το «ρεύμα» «κατακόρυφης οργάνωσης» στο οποίο οι επιχειρήσεις και οι οργανισμοί επικεντρώνονται σε ένα «κύριο» επιχειρησιακό στόχο «εξάγοντας» τυχόν δευτερεύοντες στόχους σε άλλους οργανισμούς, υιοθέτησαν ένα μοντέλο παραγωγού/ καταναλωτή στο οποίο κάθε επιχείρηση έχει ένα σύνολο από «επιχειρησιακούς συνεργάτες» που εκτελούν ενέργειες για λογαριασμό της. Αυτή η τάση διευκολύνθηκε κατά πολύ με την ανάπτυξη του «παραδείγματος» Ηλεκτρονικών Υπηρεσιών σύμφωνα με το οποίο υπηρεσίες μπορούν να δημιουργούνται από ένα «παροχέα υπηρεσιών», να διαφημίζονται σε ένα «διαμεσολαβητή υπηρεσιών» και να επερωτούνται και έπειτα να επικαλούνται από ένα «αιτών υπηρεσιών» χρησιμοποιώντας «προσυμφωνημένα» (standard) πρωτόκολλα περιγραφής και επικοινωνίας. Ακολουθώντας αυτή την μόδα οι οργανισμοί (και οι επιχειρήσεις) δεν χρειάζεται να συνάψουν «εξ'αρχής» συμφωνίες με τους «επιχειρησιακούς συνεργάτες» (business partners) στους οποίους θα «εξάγουν» κάποιες υπηρεσίες τους, αλλά αντίθετα θα αναζητήσουν σε κάποιο «διαμεσολαβητή υπηρεσιών», τους «επιχειρησιακούς συνεργάτες» που παρέχουν τις υπηρεσίες που είναι πιο κοντά στις ανάγκες τους (τόσο απο άποψη λειτουργικότητας όσο και απο άποψη «κόστους») υιοθετώντας έτσι ένα

μοντέλο «χαλαρής σύνδεσης» και «δυναμικής δέσμευσης» μεταξύ ενός οργανισμού ή επιχείρησης και των υπηρεσιών που χρησιμοποιεί.

## *1.2 Σύνθεση Ηλεκτρονικών Υπηρεσιών*

Το «παράδειγμα» των Ηλεκτρονικών Υπηρεσιών ενθαρρύνει την «επαναχρησιμοποίηση» υπαρχόντων «απλών» Ηλεκτρονικών Υπηρεσιών για την σύνθεση πολύπλοκων τονίζοντας έτσι τα χαρακτηριστικά της «άρθρωσης» (modularity) και της «διαλειτουργικότητας» (interoperability) που διέπουν τις Ηλεκτρονικές Υπηρεσίες. Για την δημιουργία πολύπλοκων Ηλεκτρονικών Υπηρεσιών απο απλούστερες, χρησιμοποιούνται γλώσσες σύνθεσης Ηλεκτρονικών Υπηρεσιών οι οποίες υιοθετούν πολλά απο τα χαρακτηριστικά των γλωσσών ορισμού Ροών Εργασίας. Το μεγαλύτερο μέρος της έρευνας σε αυτό τον τομέα έχει επικεντρωθεί στην σύνθεση διακριτών (discrete) ηλεκτρονικών υπηρεσιών με μικρή διάρκεια. Παραδείγματα τέτοιων υπηρεσιών είναι η προσθήκη ενός αντικειμένου σε ένα «καλάθι» αγορών, η χρέωση μίας πιστωτική κάρτας ή ο έλεγχος της διαθεσιμότητας ενός εισιτηρίου. Κύριο χαρακτηριστικό των γλωσσών που συνθέτούν τέτοιες υπηρεσίες είναι ότι ακολουθούν μία προσέγγιση «διαμεσολαβητή» (mediator) κατά την οποία μόνο η «σύνθετη υπηρεσία» μπορεί να επικοινωνήσει με τις απλές και οι απλές Ηλεκτρονικές Υπηρεσίες δεν αλληλεπιδρούν μεταξύ τους. Χαρακτηριστικό των διακριτών Ηλεκτρονικών Υπηρεσιών είναι ότι ούτε δημιουργούν ούτε έχουν την δυνατότητα να αποκρίνονται σε ασύγχρονα γεγονότα και για αυτό το λόγο τις ονομάζουμε «*απομονωμένες*» (insular) Ηλεκτρονικές Υπηρεσίες.

Εκτός απο τις «διακριτές» Ηλεκτρονικές Υπηρεσίες υπάρχει και μία άλλη κατηγορία Ηλεκτρονικών Υπηρεσιών οι οποίες αναπτύσσουν συνεδρίες στις οποίες πιθανώς παρευρίσκονται κάποιοι συμμετέχοντες (session-oriented services). Οι υπηρεσίες αυτές έχουν κύκλο ζωής (δηλαδή έχουν μεγάλη διάρκεια) κατά τον οποίο ασύγχρονα γεγονότα μπορεί να δημιουργηθούν απο τις συνεδρίες ή οι συνεδρίες μπορεί να χρειαστεί να αποκριθούν σε «εξωτερικά» ασύγχρονα γεγονότα. Τέτοιου είδους υπηρεσίες εμφανίζονται συχνά στον χώρο των τηλεπικοινωνιών, των συνεργατικών συνεδριών μέσω του διαδικτύου, των αλληλεπιδραστικών παιχνιδιών κ.τ.λ. Εξ'αιτίας της απαίτησης για απόκριση σε ασύγχρονα γεγονότα οι Ηλεκτρονικές Υπηρεσίες αυτές ονομάζονται «*αποκριτικές*» (responsive) Ηλεκτρονικές Υπηρεσίες.

## *1.3 Συνεισφορά της εργασίας*

Τα περισσότερα υπάρχοντα μοντέλα σύνθεσης Ηλεκτρονικών Υπηρεσιών περιγράφουν με ικανοποιητικό τρόπο την σύνθεση διακριτών υπηρεσιών χωρίς να ισχύει το ίδιο και για την περίπτωση σύνθεσης Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες. Για την σύνθεση τέτοιου είδους υπηρεσιών σχεδιάστηκε τον Ιούνιο του 2001 στα "Bell Labs" [31] το μοντέλο ενεργών διαγραμμάτων ροής και η γλώσσα σύνθεσης Ηλεκτρονικών Υπηρεσιών XASC τα οποίο υιοθετούμε και επεκτείνουμε στα πλαίσια αυτής της εργασίας. Ένα σχήμα διαδικασίας (process schema) στο μοντέλο «ενεργών διαγραμμάτων ροής» δεν αποτελείται απο μια «υψηλού επιπέδου» διαδικασία (top level process) όπως συμβαίνει σε γλώσσες όπως η BPEL4WS ή την BPML αλλά αποτελείται απο μία συλλογή απο ενεργά διαγράμματα ροής «ζευγαρωμένα» (coupled) με τον τύπο γεγονότος που τα ενεργοποιεί. Δομικά στοιχεία τύπου ροής εργασίας (workflow-like constructs) χρησιμοποιούνται για να περιγραφεί η «αντίδραση» του συστήματος στην δημιουργία ενός γεγονότος. Η δημιουργία γεγονότων δεν μπορεί να προβλεφθεί εξαρχής πότε θα συμβεί και αυτό τονίζει την «πλήρως ασύγχρονη» φύση του μοντέλου «ενεργών διαγραμμάτων ροής». Οι ενεργοποιήσεις διαγραμμάτων ροής στα πλαίσια της ίδιας διαδικασίας μπορούν να αλληλεπιδρούν μεταξύ τους με διάφορους τρόπους συντονίζοντας έτσι την εκτέλεση τους. Απο αυτή την προσέγγιση ακολουθείται μία παραλλαγή της προσέγγισης για την ανταλλαγή δεδομένων απο αυτή που ακολουθείται στα άλλα μοντέλα, μίας και οι «session oriented» Ηλεκτρονικές Υπηρεσίες μπορούν να ανταλλάσσουν δεδομένα τόσο άμεσα μεταξύ τους (π.χ «δεδομένα πολυμέσων» μέσω ενός RTP πρωτοκόλλου) όσο και με την «σύνθετη» Ηλεκτρονική Υπηρεσία (δηλαδή την διαδικασία XASC).

Στα πλαίσια της υλοποίησης του μοντέλου «ενεργών διαγραμμάτων ροής» υλοποιήθηκε η πλατφόρμα AZTEC για την σύνθεση και εκτέλεση τόσο «διακριτών» Ηλεκτρονικών Υπηρεσιών όσο και Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες. Η πλατφόρμα AZTEC είναι «οδηγούμενη απο γεγονότα» (event driven) και επιτρέπει τον ρητό ορισμό προτεραιοτήτων στην εκτέλεση των ενεργών διαγραμμάτων ροής αλλά και την ταυτόχρονη εκτέλεση τόσο πολλών διαδικασιών XASC όσο και πολλών διαγραμμάτων ροής στα πλαίσια της ίδιας διαδικασίας

Στην εργασία αυτή μελετήσαμε και επεκτείναμε τόσο το μοντέλο ενεργών διαγραμμάτων ροής όσο και την πλατφόρμα AZTEC η οποία βρισκόταν σε αρκετά πρώιμη φάση. Συνοψίζοντας την συνεισφορά της εργασίας μας μπορούμε να αναγνωρίσουμε τέσσερις κατευθύνσεις.

- Καταρχήν μελετήσαμε την εκφραστικότητα του μοντέλου ενεργών διαγραμμάτων ροής και της γλώσσας XASC αναπαριστώντας παραδείγματα σύνθεσης τόσο διακριτών

Ηλεκτρονικών Υπηρεσιών όσο και Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες. Έπειτα μελετήσαμε το μοντέλο ενεργών διαγραμμάτων ροής σε συνδυασμό με ένα σύνολο δομικών και επικοινωνιακών «προτύπων» ροών εργασίας όπως αυτά περιγράφονται στο [32]. Η μελέτη αυτή έγινε σε σύγκριση με τις πρότυπες γλώσσες σύγκρισης Ηλεκτρονικών Υπηρεσιών BPEL4WS [34] και BPML[33].

- Η μελέτη του μοντέλου ενεργών διαγραμμάτων ροής με τους παραπάνω τρόπους μας βοήθησε να αναγνωρίσουμε τα σημεία στα οποία το μοντέλο μπορούσε να επεκταθεί. Πρώτη επέκταση που υλοποιήθηκε ήταν μετατροπή του «Context Repository» σε μέσο συγχρονισμού για τα ενεργά διαγράμματα ροής υλοποιώντας «blocking» ενέργειες πάνω σε «διαμοιραζόμενες» μεταβλητές μεταξύ των διαφόρων ενεργών διαγραμμάτων ροής. Έπειτα προστέθηκαν δύο νέα δομικά στοιχεία το «Hparallel» και το «Switch», το πρώτο για την εκτέλεση πολλών παράλληλων στιγμιοτύπων της ίδιας ενέργειας και το δεύτερο για την δημιουργία πιο «συμπαγών» ορισμών στην περίπτωση επιλογής ενός από πολλά εναλλακτικά «μονοπάτια». Μάλιστα μία επιπλέον επέκταση της σημασιολογίας του στοιχείου «Switch» θα μας επιτρέψει τον συσχετισμό ενός διαγράμματος ροής με παραπάνω του ενός γεγονότα.
- Σημαντικό μέρος της εργασίας κατέλαβε η ολοκληρωτική υλοποίηση και «βελτιστοποίηση» της πλατφόρμας AZTEC όσων αφορά την αναπαράσταση των ενεργών διαγραμμάτων ροής στο εσωτερικό της. Υλοποιήσαμε το αντικειμενοστρεφές μοντέλο αναπαράστασης των διαγραμμάτων ροής και τον μηχανισμό μετατροπής κειμένων XML σύμφωνων με το «σχήμα» XML της γλώσσας XASC σε αντικείμενα αυτού του μοντέλου. Ένα πολύ σημαντικό στοιχείο είναι ότι η αντικειμενοστρεφής αυτή αναπαράσταση είναι «χαλαρά συνδεδεμένη» με το «σχήμα» XML της γλώσσας XASC. Αυτό σημαίνει ότι διαδικασίες ορισμένες σε διαφορετικές γλώσσες μπορούν να αναπαρασταθούν στο εσωτερικό μας μοντέλο αρκεί να οριστούν οι κατάλληλες αντιστοιχήσεις (mappings) μεταξύ των στοιχείων τους και των αντικειμένων του αντικειμενοστρεφούς μοντέλου. Αυτό πρακτικά καθιστά την πλατφόρμα AZTEC ανεξάρτητη γλώσσας. Ως αποτέλεσμα της υιοθέτησης αυτής της αντικειμενοστρεφούς αναπαράστασης καθίσταται ευκολότερος ο χειρισμός σύνθετων τύπων δεδομένων στα πλαίσια του διαγράμματος ροής και η μετατροπή τους σε «custom» αντικείμενα της επιλογής μας. Παράλληλα υλοποιήσαμε τον μηχανισμό «αποτίμησης» των διαγραμμάτων ροής όντας στην αντικειμενοστρεφή τους αναπαράσταση. Η υλοποίηση της πλατφόρμας AZTEC βρίσκεται αυτή την στιγμή σε μία αρκετά «πλήρη» και «σταθερή» κατάσταση και υποστηρίζεται η ταυτόχρονη εκτέλεση πολλών στιγμιοτύπων διαδικασιών, τόσο του ίδιου όσο και διαφορετικών τύπων.
- Τέλος στα πλαίσια υλοποίησης του παραδείγματος Ηλεκτρονικού Εμπορίου υλοποιήσαμε μετρήσεις απόδοσης της πλατφόρμας AZTEC η οποία δείχνει μεγάλη

σταθερότητα στον χρόνο απόκρισης τόσο με μικρούς όσο και με μεγάλους ρυθμούς λήψης αιτήσεων για δημιουργία νέων διαδικασιών.

## *1.4 Δομή της εργασίας*

Στο **κεφάλαιο 2** περιγράφουμε τα βασικά χαρακτηριστικά των Ροών Εργασίας και τα οφέλη που αποκομίζονται απο την χρήση τους στην οργάνωση επιχειρησιακών διαδικασιών. Θα παρουσιάσουμε ένα τυπικό σενάριο Ηλεκτρονικού Εμπορίου το οποίο θα χρησιμοποιούμε σαν παράδειγμα (case study) κατά την μελέτη των διάφορων γλωσσών και φορμαλισμών περιγραφής Ροών Εργασίας. Θα δούμε δύο τυπικούς φορμαλισμούς για τον ορισμό Ροών Εργασίας, τα Δίκτυα Petri και τους Χάρτες Καταστάσεων και θα δούμε που υστερεί και που υπερτερεί ο καθένας. Έπειτα θα δούμε συνοπτικά τις ανάγκες που ώθησαν στην εξέλιξη απο τις «ενδο-επιχειρησιακές» Ροές Εργασίας στις «δια-επιχειρησιακές» και πώς αυτές επωφελήθηκαν στο έπακρο απο την άνθηση του «παραδείγματος» Ηλεκτρονικών Υπηρεσιών. Θα περιγράψουμε συνοπτικά το παράδειγμα Ηλεκτρονικών Υπηρεσιών και τα βασικά πρωτόκολλα που χρησιμοποιεί. Τέλος θα δώσουμε δύο παραδείγματα (case studies) σύνθεσης Ηλεκτρονικών Υπηρεσιών ένα «διακριτών» και ένα Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες.

Στο **κεφάλαιο 3** θα δούμε τα βασικά χαρακτηριστικά τόσο του μοντέλου «ενεργών διαγραμμάτων» ροής και της γλώσσας XASC όσο και των γλωσσών BPEL4WS και BPML που θεωρούνται «πρότυπες» (standards) γλώσσες σύνθεσης Ηλεκτρονικών Υπηρεσιών. Θα παρουσιάσουμε ένα σύνολο «προτύπων» (patterns) Ροών Εργασίας και επικοινωνίας σαν ένα σύνολο απο απαιτήσεις απο τις γλώσσες ορισμού ροών εργασίας και σύνθεσης Ηλεκτρονικών Υπηρεσιών στα πλαίσια των οποίων θα μελετήσουμε την εκφραστικότητα τόσο του μοντέλου μας όσο και των «προτύπων» (standard) γλωσσών. Τέλος θα αναπαραστήσουμε τα παραδείγματα που είδαμε στο τέλος του προηγούμενου κεφαλαίου με το μοντέλο «ενεργών διαγραμμάτων ροής».

Στο **κεφάλαιο 4** θα παρουσιάσουμε την αρχιτεκτονική της πλατφόρμας AZTEC για την σύνθεση και την εκτέλεση Ηλεκτρονικών Υπηρεσιών. Θα παρουσιάσουμε την «αντικειμενοστρεφή» αναπαράσταση των ενεργών διαγραμμάτων ροής στο εσωτερικό της Μηχανής Εκτέλεσης του AZTEC και την «οδηγούμενη απο γεγονότα» φύση της Μηχανής Εκτέλεσης. Τέλος θα παρουσιάσουμε κάποιες επιλογές όσων αφορά την ανάθεση προτεραιοτήτων σε διαγράμματα ροής αλλά και την επιλογή ενός προκατασκευασμένου αλλά δυναμικού συνόλου «εξυπηρετητών» (pool of threads) για

την ταυτόχρονη εκτέλεση πολλών διαδικασιών αντί της αυθαίρετης δημιουργίας/καταστροφής «εξυπηρετητών».

Τέλος στο **κεφάλαιο 5** θα συνοψίσουμε την συνεισφορά της συγκεκριμένης μεταπτυχιακής εργασίας τονίζοντας κάποιες κατευθύνσεις για επεκτάσεις και μελλοντική εργασία.

# Κεφάλαιο 2

# Ροές Εργασίας και Ηλεκτρονικές Υπηρεσίες

Οι επιχειρησιακές διαδικασίες βρίσκονται στον πυρήνα της λειτουργίας όλων των μεγάλων οργανισμών και επιχειρήσεων. Μεγάλοι οίκοι ανάπτυξης λογισμικού, τράπεζες, ή ασφαλιστικές εταιρίες είναι μερικά μόνο παραδείγματα εταιριών με πολλές και περίπλοκες επιχειρησιακές διαδικασίες. Όσο καλύτερα δομούνται οι επιχειρησιακές διαδικασίες τόσο πιο εύκολα μπορούν να γίνουν κατανοητές από νέους υπαλλήλους καθώς επίσης και να εμπλουτιστούν με καινούργια λειτουργικότητα καθώς ο οργανισμός ή η επιχείρηση εξελίσσεται. Η απαραίτητη οργάνωση και τεκμηρίωση των σύγχρονων επιχειρήσεων υποστηρίζεται σε μεγάλο βαθμό από την χρήση ροών εργασίας που επιτρέπουν την αναπαράσταση/ οργάνωση πολύπλοκων επιχειρησιακών διαδικασιών. Επιπλέον, τροχοπέδη για πολλές εταιρίες αποτελεί το γεγονός ότι σημαντικές επιχειρησιακές διαδικασίες σχετιζόμενες κυρίως με την γραμμή παραγωγής τους, είναι αργές περιορίζοντας σημαντικά την δυνατότητα τους να αναπτυχθούν περαιτέρω. Η χρήση της τεχνολογίας των Ροών Εργασίας οργανώνει αυστηρά την σειρά εκτέλεσης ενεργειών με αποτέλεσμα να ελαττώνεται στο ελάχιστο ο «χαμένος» χρόνος ανάμεσα σε δυο ενέργειες που πρέπει να γίνουν αφαιρώντας παράλληλα τυχόν περιττές ενέργειες.

Η ραγδαία εξάπλωση του Διαδικτύου και η ανάγκη για όλο και μεγαλύτερη συνεργασία ώθησε τις εταιρίες και τους οργανισμούς να επεκτείνουν την «δράση» των επιχειρησιακών τους διαδικασιών και εκτός των αυστηρών τους ορίων. Η εξέλιξη προτύπων για την κωδικοποίηση δεδομένων όπως η γλώσσα δεικτοδότησης XML[XML]( eXtensible Markup Language) κατέστησε δυνατό τον διαμοιρασμό δεδομένων και γενικότερα πόρων μεταξύ ανθρώπων ή εφαρμογών τόσο στον ίδιο οργανισμό όσο και μεταξύ διαφορετικών οργανισμών. Κάτι που αργά ή γρήγορα έγινε αντιληπτό από τις εταιρίες και τους οργανισμούς είναι ότι η πραγματική πρόκληση δεν έγκειται στην ανταλλαγή αρχείων, αλλά στην ανταλλαγή *ενεργειών* (ή υπηρεσιών), δηλαδή στον διαμοιρασμό και συντονισμό εργασιών ανάμεσα σε διαφορετικούς οργανισμούς. Κάθε οργανισμός πλέον θα μπορεί να χρησιμοποιεί στις επιχειρησιακές του διαδικασίες υπηρεσίες υλοποιημένες από διαφορετικούς οργανισμούς χρησιμοποιώντας πιθανώς αρκετά ετερογενή εσωτερικά περιβάλλοντα ανάπτυξης (πλατφόρμες, συστήματα κ.λ.π). Κάθε οργανισμός που επιθυμεί οι υπηρεσίες που

υλοποιεί να μπορούν να χρησιμοποιηθούν από τρίτους οφείλει να παρέχει μία προσυμφωνημένη «διεπαφή» των υπηρεσιών προς τα έξω σε μορφή κατανοητή από όλους. Ο τρόπος με τον οποίο είναι υλοποιημένες οι υπηρεσίες εσωτερικά του οργανισμού δεν έχει κανένα αντίκτυπο σε αυτούς που τις χρησιμοποιούν. Τέτοιου είδους υπηρεσίες ονομάζονται *Ηλεκτρονικές Υπηρεσίες*.

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τα βασικά χαρακτηριστικά των Ροών Εργασίας, τα οφέλη που αποκομίζονται από την χρήση τους για την υλοποίηση επιχειρησιακών διαδικασιών καθώς και κάποιους τυπικούς φορμαλισμούς για την αναπαράσταση Ροών Εργασίας. Θα δούμε επίσης πώς η ανάπτυξη διεπιχειρησιακών Ροών Εργασίας ώθησε στην υιοθέτηση του παραδείγματος Ηλεκτρονικών Υπηρεσιών για την πιο ευέλικτη συνεργασία μεταξύ οργανισμών. Τέλος θα παρουσιαστούν τα βασικά πρωτόκολλα και γλώσσες για την περιγραφή και επικοινωνία μεταξύ Ηλεκτρονικών Υπηρεσιών και τα οφέλη που αποκομίζονται από την χρήση τους.

## *2.1 Επιχειρησιακές διεργασίες και Ροές Εργασίας*

Μια Ροή Εργασίας ασχολείται με την αυτοματοποίηση μιας διαδικασίας κατά την οποία έγγραφα, πληροφορία ή *έργα* ανταλλάσσονται ανάμεσα σε κάποιους συμμετέχοντες με σκοπό να επιτύχουν ένα σύνολο κανόνων ή να συμβάλουν στην επίτευξη ενός επιχειρησιακού στόχου. Παρόλο που μια Ροή Εργασίας μπορεί να οργανωθεί χειρωνακτικά (στο χαρτί για παράδειγμα), στην πράξη οι ροές εργασίας συνήθως οργανώνονται στα πλαίσια κάποιου πληροφοριακού συστήματος με σκοπό να συμβάλουν υπολογιστικά στον αυτοματισμό κάποιας διαδικασίας [WFMC]. Λίγο πιο διαισθητικά, μια Ροή Εργασίας μπορεί να περιγραφεί ως η αυτοματοποίηση της ροής του έλεγχου και της πληροφορίας ανάμεσα στα «δομικά στοιχεία» μιας επιχειρησιακής διαδικασίας [WT]. Κάθε εταιρία ή οργανισμός μπορεί να αποτελείται από ένα μεγάλο αριθμό από επιχειρησιακές διαδικασίες κάθε μια από τις οποίες αποσκοπεί στην πραγμάτωση ενός επιμέρους στόχου.

Οι ροές εργασίας είναι στενά συνυφασμένες με την διαδικασία της επανασχεδίασης (re-engineering) κάποιας επιχειρησιακής διαδικασίας. Η επανασχεδίαση σχετίζεται με την αποτίμηση, την ανάλυση, την μοντελοποίηση και έπειτα την σχεδίαση και υλοποίηση της βασικής επιχειρησιακής διαδικασίας ενός οργανισμού. Παρόλο που όλες επιχειρησιακές διαδικασίες δεν συνεπάγονται την δημιουργία ροών εργασίας, η τεχνολογία των συστημάτων διαχείρισης ροών εργασίας (WFMS) είναι συχνά η ενδεδειγμένη λύση μιας και ξεχωρίζει την επιχειρησιακή

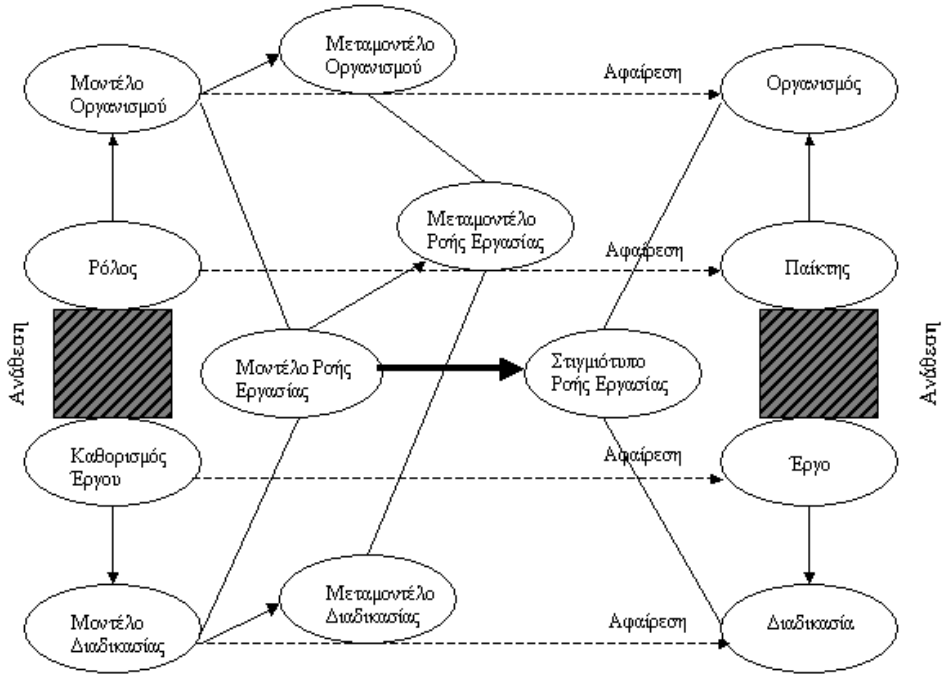
λογική από την υλοποίηση της στα πλαίσια ενός συγκεκριμένου πληροφοριακού συστήματος

Το σύνολο των βασικών στοιχείων τα οποία συνθέτουν μια Ροή Εργασίας αποτελούν το μετα-μοντέλο της Ροής Εργασίας. Διαφορετικά μετα-μοντέλα έχουν προταθεί στην βιβλιογραφία τα οποία τις περισσότερες φορές χρησιμοποιούν διαφορετικούς όρους για να περιγράψουν τα ίδια πράγματα. Παρόλα αυτά, ορισμένοι όροι είναι πολύ δημοφιλείς στην βιβλιογραφία και θα τους παρουσιάσουμε παρακάτω ονομαστικά για γίνει πιο κατανοητό το πλαίσιο δημιουργίας και εκτέλεσης μιας Ροής Εργασίας[LM95].

- Η *ενέργεια* (activity) είναι η βασική αφαίρεση μιας Ροής Εργασίας. Μια Ροή Εργασίας αποτελείται από μια ή περισσότερες ενέργειες.
- Ένα *έργο* (task) είναι ένα σαφώς καθορισμένο κομμάτι εργασίας. Μια ενέργεια περιλαμβάνει ένα έργο.
- Ένας *παίκτης* (actor) είναι ένας άνθρωπος ή μια μηχανή (ένα πρόγραμμα) που του ανατίθεται να εκτελέσει ένα έργο.
- Ένας *ρόλος* (role) είναι μια λογική αφαίρεση που αποτελείται από έναν ή περισσότερους παίκτες με σκοπό να επιτύχουν μια λειτουργικότητα δηλαδή ένα πιο πολύπλοκο πιθανώς έργο..
- Μια *διαδικασία* (process) είναι μια επιχειρησιακή διαδικασία (business process) στην οποία ένα σύνολο από έργα είναι δομημένα με τον κατάλληλο τρόπο.
- Ένα *μουτέλο διαδικασίας* (process model) είναι μια αφαιρετική όψη μιας επιχειρησιακής διαδικασίας στην οποία έμφαση δίνεται στον συντονισμό των διαφόρων έργων τονίζοντας τις εξαρτήσεις μεταξύ τους.
- Ένα *οργανισμός* (organization) είναι ένα σύνολο από δομημένες ομάδες από παίκτες.
- Ένα *μουντέλο οργανισμού* (organization model) είναι μια αφαιρετική όψη ενός οργανισμού.
- Ένα *στιγμιότυπο μιας Ροής Εργασίας* (workflow instance) είναι μια διαδικασία που σχετίζεται με ένα οργανισμό και αναθέτει έργα προς εκτέλεση στους «παίκτες» του οργανισμού.
- Ένα *μουντέλο Ροής Εργασίας* (workflow model) συνδυάζει ένα μοντέλο διαδικασίας με ένα μοντέλο οργανισμού
- Ένα *μετα-μουντέλο Ροής Εργασίας* (workflow metamodel) είναι μια γλώσσα με την οποία ορίζουμε μοντέλα Ροής Εργασίας.

Μπορούμε να δούμε διαγραμματικά τις εξαρτήσεις των παραπάνω στοιχείων στην **Εικόνα 2.1**

Από τα παραπάνω γίνεται κατανοητό ότι η έννοια της Ροής Εργασίας και της επιχειρηματικής διαδικασίας είναι στενά συνδεδεμένες. Στο εξής θα αναφερόμαστε σε Ροές Εργασίας και θα εννοούμε μια επιχειρησιακή διαδικασία η οποία είναι υλοποιημένη με Ροή Εργασίας. Σύμφωνα με τα παραπάνω, μπορούμε να δούμε ένα μοντέλο ροών εργασίας σαν ένα «πρότυπο» (schema) για την δημιουργία συγκεκριμένων στιγμιότυπων (enactment) ροών εργασίας. Μια Ροή Εργασίας μπορεί να θεωρηθεί σαν μια «εκδοχή» του μοντέλου Ροής Εργασίας δημιουργούμενη δυναμικά, ανάλογα με τις συνθήκες, δηλαδή τις «παραμέτρους» με τις οποίες θα εκτελεστεί η Ροή Εργασίας και τα αποτελέσματα εκτέλεσης των ενεργειών της όπως θα εξηγηθεί καλύτερα παρακάτω. Μπορούμε δηλαδή να πούμε ότι το μοντέλο Ροής Εργασίας περιλαμβάνει όλες τις διαφορετικές «εκδοχές» που μπορεί να έχει η Ροή Εργασίας.




**Εικόνα 2.1:Το πλαίσιο ορισμού μιας Ροής Εργασίας**

### 2.1.1 Το Μοντέλο μίας Ροής Εργασίας

Το βασικό συστατικό στοιχείο ενός μοντέλου ροών εργασίας είναι η *ενέργεια* (activity). Μια *ενέργεια* αντιπροσωπεύει μια σημασιολογικά ανεξάρτητη πράξη, η οποία μπορεί να είναι είτε «απλή» δηλαδή να εκτελείται (από ένα *παίκτη*) σε ένα βήμα, είτε «πολύπλοκη» δηλαδή να έχει μια δομή οπότε ουσιωδώς να αποτελεί μια ξεχωριστή Ροή Εργασίας. Το γεγονός αυτής της δόμησης μας επιτρέπει να ακολουθούμε προσεγγίσεις είτε «από πάνω προς τα κάτω» (top-down) είτε «από κάτω προς τα πάνω» (bottom-up) στη σχεδίαση των ροών εργασίας.

Κάθε *ενέργεια*, περιλαμβάνει ένα *έργο* (όπως αυτό ορίστηκε παραπάνω) το οποίο πρέπει να εκτελεστεί και το οποίο πιθανώς να παράγει αποτελέσματα. Τα αποτελέσματα αυτά αποτελούν παραμέτρους των *ενεργειών*. Αντίστοιχα κάποιες *ενέργειες* προσπελαύνουν αποτελέσματα άλλων, προηγούμενων *ενεργειών* τα οποία πιθανώς χρειάζονται σαν παραμέτρους για την εκτέλεση του *έργου* τους. Έτσι γενικά μπορούμε να πούμε ότι οι *ενέργειες* έχουν παραμέτρους εισόδου και παραμέτρους εξόδου. Όπως είδαμε μια Ροή Εργασίας μπορεί να αποτελεί το έργο κάποιας ενέργειας μιας «υψηλότερου επιπέδου» Ροής Εργασίας. Με αυτή την έννοια μπορεί και μια Ροή Εργασίας να έχει παραμέτρους εισόδου και εξόδου οι οποίες αποτελούν το καθολικό πλαίσιο (global context) της Ροής Εργασίας. Δίνοντας διαφορετικές τιμές στις παραμέτρους εισόδου μιας ενέργειας δημιουργούνται διαφορετικές τιμές για τις παραμέτρους εξόδου, δηλαδή η *ενέργεια* έχει διαφορετική συμπεριφορά. Έτσι μπορούμε να περιγράψουμε μια ενέργεια σαν μια «σχέση» μεταξύ των παραμέτρων εισόδου και εξόδου της.

Όπως αναφέραμε παραπάνω κάποια παράμετρος εισόδου μιας *ενέργειας* μπορεί να χρειάζεται να πάρει την τιμή της από μια παράμετρο εξόδου μιας άλλης *ενέργειας*, όχι αναγκαστικά της άμεσα προηγούμενης της. Τέτοιου είδους συσχετίσεις μεταξύ των *ενεργειών* της Ροής Εργασίας προσδιορίζουν την ροή των δεδομένων (data flow). Η ροή των δεδομένων μπορεί να αναπαρασταθεί σαν ένα δίκτυο από ακμές όπου μία ακμή υπάρχει ανάμεσα σε κάθε δύο *ενέργειες* όπου η μία «περνάει» δεδομένα στην άλλη.

Οι *ενέργειες* μιας Ροής Εργασίας δεν πρέπει να εκτελούνται σε τυχαία σειρά. Κάποια *ενέργεια* απαιτείται για να ξεκινήσει η Ροή Εργασίας ενώ κάποια ίσως να μπορεί να εκτελεστεί μόνο αφού κάποια ή κάποιες άλλες έχουν ήδη εκτελεστεί. Έτσι

οι ενέργειες σχηματίζουν ένα δίκτυο από μεταβάσεις όπου υπάρχει μια μετάβαση ανάμεσα σε κάθε *ενέργεια* και σε όλες τις πιθανές επόμενες της. Μια συγκεκριμένη Ροή Εργασίας η οποία όπως προαναφέραμε αποτελεί ένα στιγμιότυπο ενός μοντέλου Ροών Εργασίας περιλαμβάνει ένα σημείο από αυτό το δίκτυο από μεταβάσεις το οποίο ονομάζεται ροή του ελέγχου (control flow) της Ροής Εργασίας.

Μια μετάβαση από μια *ενέργεια* σε μια επόμενη της μπορεί να συσχετιστεί με μια δυαδική μέθοδο η οποία ονομάζεται *συνθήκη ροής* (flow condition). Η λογική αυτή μέθοδος έχει επίσης ένα σύνολο από παραμέτρους εισόδου που μπορεί να παίρνουν τιμές είτε από τις παραμέτρους εξόδου της προηγούμενης ενέργειας είτε από τις «καθολικές» παραμέτρους της Ροής Εργασίας. Αν η συνθήκη ροής πάρει την τιμή «Αληθής» τότε εκτελείται η ενέργεια που ακολουθεί ειδάλλως δεν εκτελείται.

Μια ενέργεια μπορεί να έχει παραπάνω από μια μεταβάσεις οι οποίες καταλήγουν σε αυτή. Σε αυτή την περίπτωση η ενέργεια λειτουργεί σαν *ένωση* (join). Στην περίπτωση αυτή συσχετίζεται με την ενέργεια μια δυαδική μέθοδος η οποία ονομάζεται *συνθήκη συγχρονισμού* (synchronization expression). Αν η συνθήκη συγχρονισμού πάρει τιμή «Αληθής» εκτελείται η ενέργεια ειδάλλως δεν εκτελείται. Η συνθήκη συγχρονισμού μπορεί να είναι είτε η «τομή» των συνθηκών ροής των διαφόρων «εισερχόμενων» μεταβάσεων είτε όχι. Αντίστοιχα μια *ενέργεια* μπορεί να έχει «εξερχόμενες» μεταβάσεις προς παραπάνω από μια ενέργεια. Σε περίπτωση που οι μεταβάσεις έχουν συνθήκες και οι συνθήκες είναι αληθείς τότε η ενέργεια λειτουργεί σαν *διακλάδωση* (fork). Σε αυτή την περίπτωση γίνεται δυνατή η παράλληλη εκτέλεση ενεργειών. Σε επόμενη υποενότητα θα παρουσιαστεί ένα παράδειγμα Ροής Εργασίας όπου θα ξαναδούμε γραφικά τα παραπάνω στοιχεία, με έμφαση στην παρουσίαση της ροής του έλεγχου και των δεδομένων.

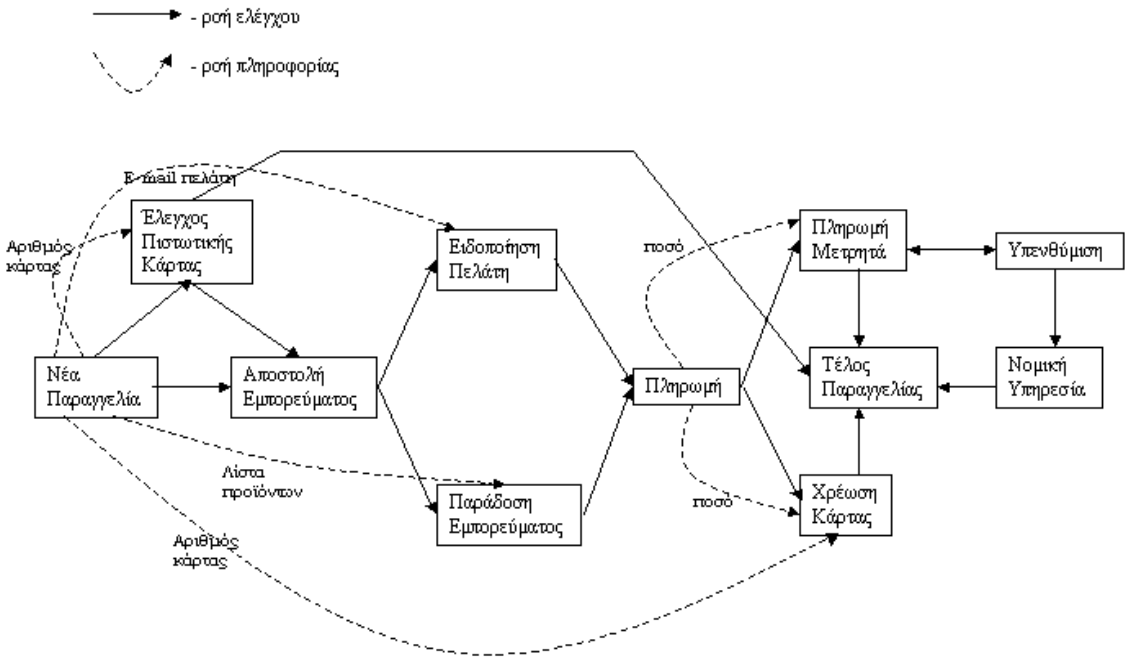
Αν και τα παραπάνω αποτελούν στην ουσία την περιγραφή του μοντέλου Ροής Εργασίας ενός συγκεκριμένου συστήματος ([LD94],[LA94]), εντούτοις αποτελούν μια αρκετά γενική και αφαιρετική σύνοψη των βασικών στοιχείων ενός μοντέλου Ροής Εργασίας. Κάποια άλλα χαρακτηριστικά που ίσως συναντήσουμε σε άλλες σχετικές εργασίες [FGP+97] περιλαμβάνουν τον ορισμό *προσυνθηκών* (preconditions) και *μετασυνθηκών* (postconditions) για τις *ενέργειες* και κανόνες για τον χειρισμό λαθών.

## 2.1.2 Παράδειγμα Ροής Εργασίας

Σε αυτή την ενότητα θα δώσουμε ένα συγκεκριμένο παράδειγμα Ροής Εργασίας εμπνευσμένο από ένα τυπικό σενάριο ηλεκτρονικού εμπορίου. Όπως μπορούμε να δούμε από την **Εικόνα 2.2**, το παράδειγμα είναι απλό αλλά καλύπτει όλα τα βασικά χαρακτηριστικά μιας Ροής Εργασίας. Η *ενέργεια* «ΝέαΠαραγγελία» εκκινεί την Ροή Εργασίας. Έπειτα η ροή του έλεγχου χωρίζεται. Αν ο πελάτης επιθυμεί να πληρώσει με πιστωτική κάρτα, τότε γίνεται μετάβαση στην *ενέργεια* «ΈλεγχοςΠιστωτικήςΚάρτας» η οποία ελέγχει την εγκυρότητα του αριθμού πιστωτικής κάρτας που έχει δώσει ο πελάτης. Αν ο αριθμός δεν αντιστοιχεί σε κάποια έγκυρη πιστωτική κάρτα τότε η Ροή Εργασίας τερματίζεται. Η επόμενη *ενέργεια* και στις δυο περιπτώσεις πληρωμής είναι η «ΑποστολήΕμπορεύματος» η οποία έχει δυο εξερχόμενες μεταβάσεις, μια προς την ενέργεια «ΕιδοποίησηΠελάτη» και μια προς την *ενέργεια* «ΠαράδοσηΕμπορεύματος». Η *ενέργεια* «ΕιδοποίησηΠελάτη» στέλνει ένα ενημερωτικό e-mail στον πελάτη. Η *ενέργεια* «ΠαράδοσηΕμπορεύματος» είναι μια υποδιαδικασία . Η υποδιαδικασία αυτή περιλαμβάνει τις εξής *ενέργειες*: Η *ενέργεια* «ΕκκίνησηΠαράδοσης» εκκινεί τη νέα Ροή Εργασίας. Στην συνέχεια η *ενέργεια* «παραγγελία» αναθέτει στο επιλεγμένο κατάστημα να κάνει την παράδοση του προϊόντος. Αυτή η διαδικασία γίνεται επαναληπτικά για κάθε προϊόν που περιλαμβάνει η παραγγελία. Για να γυρίσουμε στην Ροή Εργασίας ηλεκτρονικού εμπορίου , οι *ενέργειες* «ΕιδοποίησηΠελάτη» και «ΠαράδοσηΕμπορεύματος» εκτελούνται παράλληλα. Η *ενέργεια* «ΠαράδοσηΕμπορεύματος» έχει δυο εξερχόμενες μεταβάσεις ανάλογα με τον τρόπο πληρωμής που έχει επιλέξει ο πελάτης. Η μια μετάβαση είναι προς την *ενέργεια* «Πληρωμή» και η άλλη προς την *ενέργεια* «ΧρέωσηΚάρτας». Την *ενέργεια* «Πληρωμή» ακολουθεί η *ενέργεια* «Υπενθύμιση» η οποία πρακτικά θέτει την διορία που έχει ο πελάτης για να πληρώσει και τον ενημερώνει σε περίπτωση που καθυστερήσει. Αν ο πελάτης αγνοήσει τρεις υπενθυμίσεις, γίνεται μετάβαση στην *ενέργεια* «ΝομικήΥπηρεσία» που αναλαμβάνει την υπόθεση. Στην **Εικόνα 2.2**, μπορούμε να δούμε διαγραμματικά τη Ροή Εργασίας για αυτή την επιχειρησιακή διαδικασία. Για συντομία παρουσιάζουμε μόνο την ροή υψηλού επιπέδου και όχι την υποδιαδικασία «ΠαράδοσηΕμπορεύματος».

Η ροή του ελέγχου απεικονίζεται με τα συνεχόμενα βέλη ενώ η ροή της πληροφορίας με τα διακεκομμένα. Είναι προφανές ότι η ροή της πληροφορίας

ακολουθεί σε μεγάλο βαθμό την ροή του ελέγχου – το οποίο είναι λογικό και δείχνει την εξάρτηση κάθε *ενέργειας* από τα αποτελέσματα που έδωσε η προηγούμενη – αλλά υπάρχουν και ενέργειες οι οποίες παίρνουν σαν είσοδο πληροφορία από *ενέργειες* όχι «άμεσα» προηγούμενες. Έτσι για παράδειγμα, η *ενέργεια* «ΑποστολήΕμπορεύματος» παίρνει πληροφορία από την προηγούμενη της («ΈλεγχοςΠιστωτικήςΚάρτας») γιατί η αποστολή του εμπορεύματος εξαρτάται από το αν είναι έγκυρη η κάρτα (στην περίπτωση της πληρωμής με κάρτα). Αντίστοιχα η *ενέργεια* «ΧρέωσηΚάρτας» χρειάζεται σαν πληροφορία τον αριθμό της πιστωτικής κάρτας τον οποίο δεν θα τον πάρει από την προηγούμενη *ενέργεια* της αλλά από την *ενέργεια* «ΝέαΠαραγγελία» όπου ο πελάτης συμπλήρωσε τα στοιχεία της παραγγελίας του.




**Εικόνα 2.2:Ροή Εργασίας για το παράδειγμα Ηλεκτρονικού Εμπορίου.**

### 2.1.3 Οφέλη από την χρήση Ροών Εργασίας

Μια από τις βασικές επιδιώξεις κατά την επανασχεδίαση μιας επιχειρησιακής διαδικασίας είναι η ελαχιστοποίηση του χρόνου εκτέλεσης της. Αυτό μπορεί να επιτευχθεί αφενός αφαιρώντας όλες τις «περιττές» ενέργειες αφετέρου επιτυγχάνοντας το μεγαλύτερο δυνατό βαθμό παραλληλισμού όσων αφορά την εκτέλεση των ενεργειών. Αυτό

καθίσταται εφικτό μιας και διαφορετικές ενέργειες μπορούν να εκτελούνται από διαφορετικές οντότητες (ανθρώπους ή προγράμματα) [LD97].

Παραδοσιακές «μονολιθικές» εφαρμογές αναπτύσσονταν σαν ένα μεγάλο κομμάτι κώδικα με κάποια εσωτερική δομή , μέσα στο οποίο δεν υπήρχε σαφής διαχωρισμός μεταξύ της διαχείρισης των δεδομένων που χρησιμοποιεί η εφαρμογή ή που υλοποιείται της υλοποίησης της επιχειρησιακής «λογική» (business logic) της εφαρμογής. Μια αλλαγή στο «σχήμα» των δεδομένων όπως η προσθήκη ενός πεδίου ή η αλλαγή ενός μονοπατιού απαιτούσε από ολόκληρη την εφαρμογή να αλλάξει για να γίνει συμβατή με τα νέα δεδομένα. Αυτό καθιστούσε τις εφαρμογές «εξαρτώμενες από τα δεδομένα» (data dependent). Αντίστοιχα όταν γινόταν κάποια αλλαγή όσων αφορά την ανάθεση έργων προς εκτέλεση προς κάποιες οντότητες (ανθρώπους ή προγράμματα) αυτό επίσης απαιτούσε αντίστοιχες αλλαγές από την εφαρμογή, καθιστώντας τις εφαρμογές «εξαρτώμενες από τη ροή» (flow dependent). Το πρώτο πρόβλημα λύθηκε με την ανάπτυξη Συστημάτων Διαχείρισης Βάσεων Δεδομένων με αποτέλεσμα οι εφαρμογές να έχουν την ίδια εικόνα για τα δεδομένα ασχετα με την εσωτερική αναπαράσταση των δεδομένων. Το δεύτερο πρόβλημα λύθηκε με την ανάπτυξη Συστημάτων Διαχείρισης Ροών Εργασίας που επιτρέπει τον διαχωρισμό της επιχειρησιακής λογικής μιας εφαρμογής από την ανάθεση έργων σε «εκτελεστές έργων». Η «λογική αφαίρεση» ενός στοιχειώδους μέρους δουλείας σε μια Ροή Εργασίας γίνεται με την έννοια *ενέργεια* ή οποία έπειτα αναθέτει ένα συγκεκριμένο *έργο* σε μια οντότητα που θα το εκτελέσει. Η Ροή Εργασίας προσδιορίζει την ροή του ελέγχου και των δεδομένων (control and data flow) ανάμεσα στις διάφορες ενέργειες. Παρακάτω αναφέρονται τα τέσσερα βασικότερα οφέλη από την χρήση ροών εργασίας για την οργάνωση επιχειρηματικών διαδικασιών.

- Ευελιξία στην αλλαγή του μοντέλου της επιχειρησιακής διαδικασίας

Το μοντέλο Ροής Εργασίας είναι ένα μοντέλο δύο επιπέδων Ο διαχωρισμός της ροής του ελέγχου και της ανάθεσης των ενεργειών στις οντότητες που τις εκτελούν παρέχει επιπλέον ευελιξία. Η ροή του ελέγχου μπορεί να αλλάζει χωρίς να επηρεάζεται η ανάθεση έργων σε οντότητες και αντίστοιχα η ανάθεση έργων σε οντότητες που τα εκτελούν μπορεί να αλλάξει χωρίς να επηρεάζεται η ροή του ελέγχου.

- *Δυνατότητα ενοποίησης ακόμα και μη συμβατών/ ετερογενών εφαρμογών.*

Το γεγονός της αφαίρεσης της *ενέργειας* η οποία έπειτα αναθέτει το έργο προς εκτέλεση σε μια οντότητα που μπορεί να είναι στο ίδιο ή σε διαφορετικό οργανισμό, που χρησιμοποιεί το ίδιο ή διαφορετικό λειτουργικό σύστημα κ.τ.λ μας δίνει την δυνατότητα να χρησιμοποιήσουμε στην ίδια Ροή Εργασίας εφαρμογές οι οποίες θεωρούνται μη

συμβατές. Οι ετερογενείς αυτές εφαρμογές ανταλλάσσουν πληροφορία μέσω της ροής δεδομένων της Ροής Εργασίας.

- *Επαναχρησιμοποίηση ενεργειών ή και ολόκληρων μοντέλων διεργασιών.*

Το γεγονός της αφαίρεσης της *ενέργειας* είναι άρρηκτα συνδεδεμένο με την υλοποίηση του έργου της από κάποια οντότητα. Όπως έχουμε προαναφέρει ένα έργο μίας ενέργειας μπορεί να είναι είτε ένα απλό «βήμα εκτέλεσης» μιας εφαρμογής ή ενός ανθρώπου, είτε και μια σύνθετη επιχειρησιακή διαδικασία. Σε αυτή την περίπτωση η υλοποίηση της ενέργειας έγκειται στην υλοποίηση ολόκληρης της επιχειρησιακής διαδικασίας. Υλοποιώντας το έργο μιας ενέργειας σαν επιχειρησιακή διαδικασία λεμε ότι αυτό είναι μια *υποδιεργασία* (subprocess). Όπως προαναφέραμε η ύπαρξη υποδιεργασιών επιτρέπει την μοντελοποίηση διεργασιών με την προσέγγιση «από πάνω προς τα κάτω» ή και «από κάτω προς τα πάνω» Και στις δυο περιπτώσεις μπορούμε να επωφεληθούμε από το ότι έχουμε την υλοποίηση μιας ενέργειας «έτοιμη» και μπορούμε να την επαναχρησιμοποιήσουμε είτε στην ίδια είτε σε κάποια άλλη Ροή Εργασίας του ίδιου οργανισμού.

- *Κλιμάκωση στην ανάπτυξη εφαρμογών*

Ένα ακόμα πλεονέκτημα από την χρήση Ροών Εργασίας για την υλοποίηση επιχειρησιακών διαδικασιών είναι η κλιμάκωση τους τόσο σε σχέση με την σχεδίαση όσο και με την εκτέλεση των διαδικασιών Η αρχιτεκτονική δυο επιπέδων που εισάγεται διαχωρίζει την σχεδίαση της επιχειρησιακής λογικής (πρακτικά της ροής ελέγχου) με την υλοποίηση των ενεργειών με αποτέλεσμα να μειώνεται κατά πολύ η πολυπλοκότητα της σχεδίασης και ανάπτυξης σύνθετων επιχειρησιακών διαδικασιών. Η χρήση υποδιεργασιών επιτρέπει την μοντελοποίηση «από πάνω προς τα κάτω» δίνοντας την δυνατότητα στην διαδικασία κάθε επιπέδου να σχεδιαστεί και να ελεγχθεί ανεξάρτητα από τις υπόλοιπες, διευκολύνοντας έτσι την ταυτόχρονη ανάπτυξη επιχειρησιακών εφαρμογών από πιθανώς διαφορετικές ομάδες εργασίας. Η κλιμάκωση της εφαρμογής κατά την διάρκεια της εκτέλεσης μπορεί να διευκολυνθεί με την ανάθεση έργων σε οσοδήποτε κατανεμημένους «εξυπηρετητές» επιτυγχάνοντας κατά αυτό τον τρόπο εξισορρόπηση του φόρτου εργασίας και καλύτερη εκμετάλλευση την υπαρχόντων υπολογιστικών πόρων.

## 2.1.4 Τυπικά Μοντέλα Ροών Εργασίας

Η ανάλυση και μοντελοποίηση Ροών Εργασίας πρέπει να γίνεται με τρόπο τυπικό, ούτως ώστε να εξασφαλίζεται η ορθότητα της εκτέλεσης τους αποφεύγοντας ορισμούς με τυχόν ασάφειες ή αντιφάσεις (σε αντίθεση με αρκετές διαγραμματικές τεχνικές που στερούνται κάποιας μαθηματικής υποδομής). Διάφοροι τυπικοί φορμαλισμοί μπορούν να χρησιμοποιηθούν για την μοντελοποίηση και την επαλήθευση μιας διαδικασίας (δηλαδή να μπορούν να αποφανθούν για το αν η διαδικασία διατηρεί ιδιότητες όπως αυτή της *ασφάλειας*[1] ή της *ζωτικότητας* [2]). Δύο καλά ορισμένοι τυπικοί φορμαλισμοί -τους οποίους θα περιγράψουμε παρακάτω- είναι τα Δίκτυα Petri[M89] (Petri Nets) και οι Χάρτες Καταστάσεων[H87] (*Statecharts).* Πολλές από τις γλώσσες που έχουν δημιουργηθεί για την περιγραφή Ροών Εργασίας έχουν στηριχθεί σε αυτούς τους φορμαλισμούς για να αποκτήσουν κάποιο θεωρητικό υπόβαθρο.

### 2.1.4.1 Δίκτυα Petri

Ένας καλά ορισμένος τυπικός φορμαλισμός για την μοντελοποίηση διαδικασιών είναι τα Δίκτυα Petri. Υπάρχουν αρκετοί λόγοι που συνηγορούν στην χρήση Δικτύων Petri για την μοντελοποίηση διαδικασιών και επομένως Ροών Εργασίας που περιγράφουν διαδικασίες [A98].

- *Τυπική Σημασιολογία*

Μια Ροή Εργασίας μοντελοποιημένη με Δίκτυα Petri έχει μια σαφή και ακριβή περιγραφή.

- *Γραφική Αναπαράσταση*

Τα Δίκτυα Petri είναι μια γραφική γλώσσα με αποτέλεσμα να είναι ιδιαίτερα διαισθητικά και εύκολα στην εκμάθηση και κατανόηση.

- *Εκφραστικότητα*

Τα Δίκτυα Petri υποστηρίζουν τα απαραίτητα αξιώματα ώστε να μοντελοποιήσουν μια Ροή Εργασίας. Οι περισσότερες δομές πλοήγησης που χρησιμοποιούνται από τα διάφορα

[1] Ότι αποφεύγονται δηλαδή καταστάσεις όπως το αδιέξοδο κλπ

[2] Ότι υπάρχει δηλαδή κάποια ακολουθία εκτέλεσης που οδηγεί σε επιτυχή κατάληξη

συστήματα διαχείρισης Ροών Εργασίας και από τις διάφορες γλώσσες και φορμαλισμούς για την περιγραφή Ροών Εργασίας μπορούν να αναπαρασταθούν με την χρήση Δικτύων Petri.

- *Ανάλυση*

Η χρήση Δικτύων Petri για την μοντελοποίηση Ροών Εργασίας διευκολύνει την ανάλυση της Ροής Εργασίας και την τεκμηρίωση διάφορων ιδιοτήτων της όπως την ιδιότητα της ασφάλειας και της ζωτικότητας όπως αναφέραμε παραπάνω. Επίσης διευκολύνει την διεξαγωγή υπολογισμών όσων αφορά την απόδοση του συστήματος (χρόνοι απάντησης, χρόνοι αναμονής κ.τ.λ)

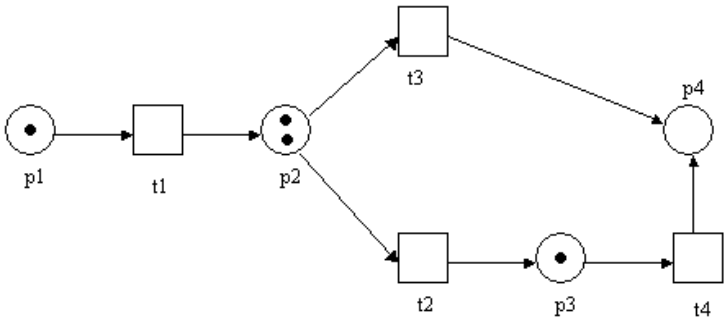
- *Ανεξαρτησία*

Τα Δίκτυα Petri παρέχουν ένα πλαίσιο Εργασίας για την μοντελοποίηση και την ανάλυση διαδικασιών ανεξάρτητο από συγκεκριμένα εμπορικά προϊόντα.

### *2.1.4.1.1 Περιγραφή ενός Δικτύου Petri*

Ένα Δίκτυο Petri είναι ένας κατευθυνόμενος γράφος ο οποίος έχει κόμβους δυο τύπων, *σημεία* (places) και *μεταβάσεις* (transitions). Οι κόμβοι ενώνονται μεταξύ τους με κατευθυνόμενες ακμές και δεν επιτρέπεται να ενωθούν δυο κόμβοι του ίδιου τύπου (δηλαδή δεν επιτρέπεται να ενωθούν με μια ακμή δυο *σημεία* ή δυο *μεταβάσεις*. Όπως μπορούμε να δούμε στην **Εικόνα 2.3**, ένα *σημείο* αναπαρίσταται με ένα κύκλο ενώ μια *μετάβαση* αναπαρίσταται με ένα ορθογώνιο. Ένα *σημείο* ονομάζεται *σημείο εισόδου* μιας *μετάβασης* αν υπάρχει μια ακμή που να ξεκινάει από το *σημείο* αυτό και να καταλήγει στην *μετάβαση*. Αντίστοιχα ένα *σημείο* ονομάζεται *σημείο εξόδου* μιας *μετάβασης* αν υπάρχει μια ακμή που να ξεκινάει από την *μετάβαση* και να καταλήγει σε αυτό το *σημείο*. Μια μετάβαση μπορεί να έχει πολλά *σημεία εισόδου* και πολλά *σημεία εξόδου*. Τα *σημεία* σε ένα Δίκτυο Petri μπορούν να περιέχουν *κουπόνια* (tokens).

Η δομή ενός Δικτύου Petri είναι σταθερή και καθορισμένη εξ'αρχής, ενώ η κατανομή των *κουπονιών* στα διάφορα *σημεία* καθορίζει την *κατάσταση* (state) στην οποία βρίσκεται το Δίκτυο Petri. Μία *κατάσταση* μπορεί να περιγραφεί για παράδειγμα ως $1p_1+2p_2+1p_3+0p_4$ ή εναλλακτικά ως $p_1+2p_2+p_3$ το οποίο σημαίνει ότι υπάρχει ένα *κουπόνι* στο *σημείο* $p_1$, δυο *κουπόνια* στο *σημείο* $p_2$ και 1 *κουπόνι* στο *σημείο* $p_3$. Ένα Δίκτυο Petri σε αυτή την κατάσταση είναι και αυτό της **Εικόνας 2.3**.




**Εικόνα 2.3: Παράδειγμα Δικτύου Petri**

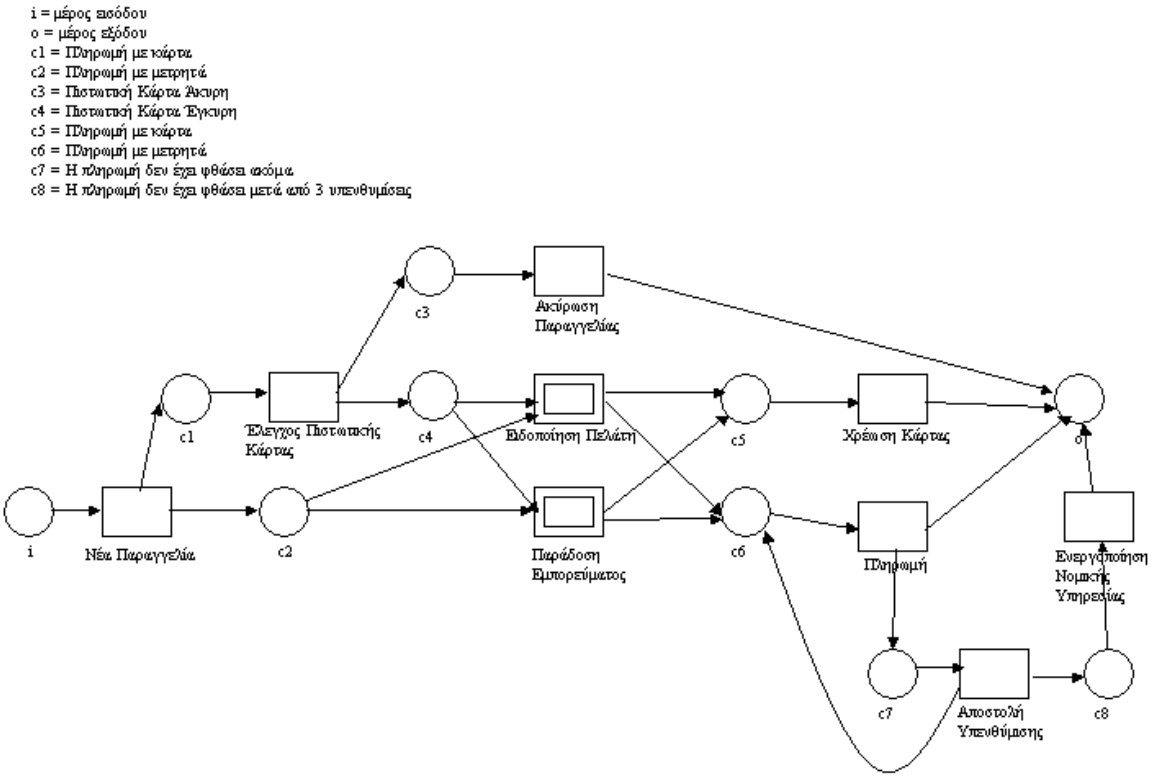
Για να ενεργοποιηθεί μια *μετάβαση* πρέπει να υπάρχει τουλάχιστον ένα *κουπόνι* σε κάθε *σημείο εισόδου* της. Όταν ενεργοποιείται μια μετάβαση τότε μεταφέρονται *κουπόνια* από τα *σημεία εισόδου* προς τα *σημεία εξόδου* της. Γι'αυτό το λόγο λεμε ότι μια *μετάβαση* καταναλώνει *κουπόνια* από τα *σημεία εισόδου* της και παράγει κουπόνια στα *σημεία εξόδου* της. Εξ'αιτίας αυτής της μεταφοράς κουπονιών είναι λογικό ότι αλλάζει η *κατάσταση* του Δικτύου Petri, αφού αλλάζει η κατανομή των *κουπονιών* στα διάφορα *σημεία*. Οι *μεταβάσεις* λέγεται ότι είναι το ενεργό κομμάτι των Δικτύων Petri ακριβώς γιατί προκαλούν την μετάβαση από την μια κατάσταση στην άλλη, ενώ τα *σημεία* αποτελούν το ανενεργό κομμάτι τους. Έτσι μπορεί να γίνει η διαισθητική αντιστοίχηση των *μεταβάσεων* σε γεγονότα ή λειτουργίες που αλλάζουν την κατάσταση ενός συστήματος, των *σημείων* σε συνθήκες που ισχύουν σε κάποια συγκεκριμένη στιγμή, ενώ τα *κουπόνια* μπορούν να θεωρηθούν σαν τα αντικείμενα ή τα δεδομένα που ανταλλάσσονται.

Τα παραπάνω χαρακτηριστικά συνθέτουν τα Δίκτυα Petri χαμηλού επιπέδου. Παρόλο που με την χρήση των Δικτύων Petri χαμηλού επιπέδου μπορούμε να περιγράψουμε σχεδόν οποιαδήποτε επιχειρηματική διαδικασία, δεν παύουν να υπάρχουν κάποιοι περιορισμοί. Καταρχήν πραγματικές διαδικασίες μπορεί να οδηγήσουν στην δημιουργία πολύ μεγάλων και πολύπλοκων Δικτύων Petri τα οποία είναι δύσκολο να γίνουν εύκολα κατανοητά. Έπειτα, αν θεωρήσουμε ότι τα *κουπόνια* αντιστοιχούν σε αντικείμενα/ δεδομένα υπάρχουν φορές που θέλουμε να μπορούμε να ξεχωρίσουμε τα διάφορα χαρακτηριστικά (attributes) αυτών των δεδομένων. Αυτό με την χρήση των Δικτύων Petri χαμηλού επιπέδου δεν είναι εφικτό αφού σε αυτά όλα τα *κουπόνια* είναι όμοια. Τέλος υπάρχουν φορές που θέλουμε να μελετήσουμε την «χρονική συμπεριφορά» μιας διαδικασίας. Η έννοια του χρόνου δεν υπάρχει στα Δίκτυα Petri χαμηλού επιπέδου. Για την αντιμετώπιση των προβλημάτων αυτών έχουν προταθεί ορισμένες επεκτάσεις στο βασικό μοντέλο των Δικτύων Petri.

Έτσι έχουμε τα *Ιεραρχικά Δίκτυα Petri (Hierarchical Petri Nets)* [F93][HJS91] στα οποία μπορούμε να έχουμε εκτός από απλές *μεταβάσεις*, και σύνθετες οι οποίες ονομάζονται *υποδίκτυα* (subnets). Ένα υποδίκτυο είναι κι αυτό ένα Δίκτυο Petri το οποίο μπορεί με την σειρά του να περιλαμβάνει και άλλα υποδίκτυα. Με αυτό τον τρόπο έχουμε ιεραρχημένη «από πάνω προς τα κάτω» ή «από κάτω προς τα πάνω» σχεδίαση των Δικτύων Petri το οποίο αφενός τα καθιστά απλούστερα στην κατανόηση και την χρήση, αφετέρου επιτρέπει την επαναχρησιμοποίηση των υποδικτύων. Εν συνεχεία έχουμε τα *Χρωματισμένα Δίκτυα Petri (Colored Petri Nets)* [J92][J94][J97] στα οποία τα *κουπόνια* δεν είναι όλα ίδια αλλά καθένα έχει μια τιμή – που το κάνει να ξεχωρίζει από τα αλλά – και συνήθως την ονομάζουμε «χρώμα» του *κουπονιού*. Τέλος υπάρχουν τα *Χρονικά Δίκτυα Petri (Temporal Petri Nets)* [R74] στα οποία έχει εισαχθεί η έννοια του χρόνου, στα *κουπόνια*, στα *σημεία* ή και στις *μεταβάσεις*. Δίκτυα Petri με τα παραπάνω χαρακτηριστικά ονομάζονται Δίκτυα Petri υψηλού επιπέδου.

### *2.1.4.1.2 Μοντελοποίηση μιας Ροής Εργασίας με Δίκτυα Petri.*

Η μοντελοποίηση μιας Ροής Εργασίας με την χρήση Δικτύων Petri είναι μία αρκετά τετριμμένη διαδικασία. Ουσιαστικά, οι *ενέργειες* αντιστοιχούν σε *μεταβάσεις*, οι διάφορες συνθήκες σε *σημεία* και τα διαφορετικά στιγμιότυπα εκτέλεσης της ίδιας Ροής Εργασίας εκφράζονται με την κατανομή των κουπονιών στο Δίκτυο Petri. Ένα Δίκτυο Petri το οποίο μοντελοποιεί μια Ροή Εργασίας ονομάζεται *WF-net* (Workflow Net). Δύο ειναι οι βασικές απαιτήσεις από ένα WF-net. Πρώτον, να έχει ένα αρχικό και ένα τελικό *σημείο* από τα οποία να ξεκινάει και να καταλήγει η Ροή Εργασίας και δεύτερον κάθε *σημείο* και κάθε *μετάβαση* να ανήκουν σε ένα τουλάχιστον «μονοπάτι» το οποίο ξεκινάει από το αρχικό *σημείο* και καταλήγει στο τελικό, σε κάποιο στιγμιότυπο εκτέλεσης. Αξίζει να σημειωθεί ότι τα διπλά ορθογώνια στην περίπτωση των *μεταβάσεων* «Ειδοποίηση Πελάτη» και «Παράδοση Εμπορεύματος» είναι χαρακτηριστικά των Ιεραρχικών Δικτύων Petri. Οι *μεταβάσεις* αυτές δηλαδή είναι στην ουσία *υποδίκτυα* τα οποία χάριν συντομίας δεν έχουμε περιγράψει στην **Εικόνα 2.4**. Με βάσει τα παραπάνω, η Ροή Εργασίας που περιγράφηκε στην παράγραφο 2.1.5 μοντελοποιείται από το Δίκτυο Petri της **Εικόνας 2.4.**




**Εικόνα 2.4:Μοντελοποίηση μίας Ροής Εργασίας ηλεκτρονικού εμπορίου με ενα Δίκτυο Petri.**

### 2.1.4.2 Χάρτες Καταστάσεων

Οι Χάρτες Καταστάσεων είναι ένας φορμαλισμός ευρέως γνωστός από το πεδίο των αναδραστικών (reactive) συστημάτων, ο οποίος μπορεί να εφαρμοστεί όμως και για την περιγραφή Ροών Εργασίας

Η χρήση Χαρτών Καταστάσεων για την μοντελοποίηση Ροών Εργασίας μπορεί να δικαιολογηθεί από τους παρακάτω λόγους:

- Ο φορμαλισμός των Χαρτών Καταστάσεων είναι βασισμένος στις μηχανές πεπερασμένων καταστάσεων, έχει μια –μαθηματικά- καλά ορισμένη σημασιολογία, επομένως αποτελεί μια «αυστηρή» και τυπική μέθοδο περιγραφής (συστημάτων, διαδικασιών κλπ)
- Οι Χάρτες Καταστάσεων μπορούν να αναπαρασταθούν γραφικά με ένα κομψό και διαισθητικό τρόπο, που αφενός κάνει την κατανόηση τους ευκολότερη, αφετέρου διευκολύνει την οπτικοποίηση της εκτέλεσης τους. Κάθε στιγμή η *κατάσταση* του μοντέλου εκφράζεται από ένα από τα στοιχεία της γραφικής του αναπαράστασης καθιστώντας εφικτή την παρακολούθηση (monitoring) της εκτέλεσης του συστήματος.

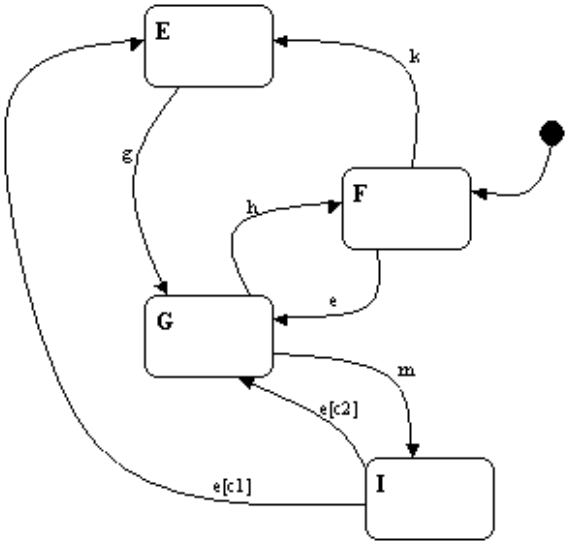
### *2.1.4.2.1 Περιγραφή ενός Χάρτη Κατάστασης*

Οι Χάρτες Καταστάσεων και οι Χάρτες Ενεργειών είναι οι δυο όψεις του ίδιου μοντέλου. Οι *ενέργειες* (activities) αποτελούν το ενεργό κομμάτι μιας περιγραφής και ο Χάρτης Ενεργειών πρακτικά περιγράφει την ροή των δεδομένων ανάμεσα στις διάφορες *ενέργειες* με την μορφή ενός κατευθυνόμενου γράφου. Οι κόμβοι αντιπροσωπεύουν τις διάφορες *ενέργειες* ενώ οι ακμές είναι *σημειωμένες* με δεδομένα.

Οι Χάρτες Καταστάσεων από την άλλη απεικονίζουν την ροή του έλεγχου ανάμεσα στις διάφορες ενέργειες. Ένας Χάρτης Καταστάσεων είναι ουσιαστικά μια μηχανή πεπερασμένων καταστάσεων όπου οι μεταβάσεις καθοδηγούνται από κανόνες της μορφής Γεγονός-Συνθήκη-Δράση (Event-Condition-Action,ECA). Για κάθε Χάρτη Καταστάσεων πρέπει να οριστεί μια αρχική *κατάσταση*. Κάθε ακμή μετάβασης ανάμεσα σε δυο *καταστάσεις* είναι «σημειωμένη» με ένα κανόνα ECA. Μια μετάβαση από την *κατάσταση* X στην *κατάσταση* Y ενεργοποιείται, αν το *γεγονός* E συμβεί και η *συνθήκη* C ισχύει. Σαν αποτέλεσμα, η *κατάσταση* X εγκαταλείπεται, η *κατάσταση* Y εισάγεται και η *δράση* A συμβαίνει. Οι *συνθήκες* και οι *δράσεις* εκφράζονται με την μορφή συγκεκριμένων μεταβλητών, αυτών που ορίζονται στην ροή των δεδομένων (στο Χάρτη Ενεργειών). Επίσης για τον ορισμό *γεγονότων*, *συνθηκών* ή *δράσεων* μπορούν να χρησιμοποιηθούν κάποια κατηγορήματα όπως en(S) (γίνεται εισαγωγή στην κατάσταση S) [WW96], in(S) (αληθές όταν το σύστημα είναι στην κατάσταση S) κ.λ.π

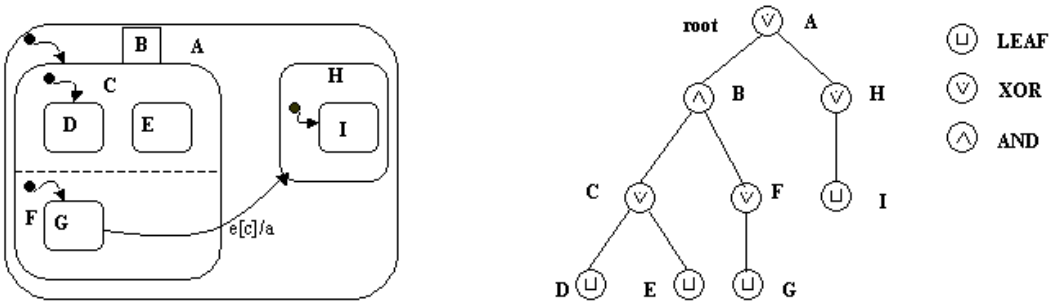
Μια *δράση* A μπορεί να ξεκινήσει ή ρητά να τερματίσει μια *ενέργεια*, μπορεί να δημιουργήσει ένα γεγονός E ή να θέσει την τιμή μιας συνθήκης C. Οι κανόνες ECA απεικονίζονται ως E[C]/A. Οποιοδήποτε από τα τρία μέρη του κανόνα μπορεί να μην είναι ορισμένο. Αν κανένα από τα τρία μέρη του κανόνα δεν είναι ορισμένο τότε η μετάβαση ενεργοποιείται αυτόματα. Η μετάβαση μεταξύ δυο *καταστάσεων* θεωρείται ότι διαρκεί μια μονάδα χρόνου, οπότε υπό αυτή την έννοια η μετάβαση μεταξύ καταστάσεων εισάγει μια έννοια διακριτού χρόνου. Ένας απλός Χάρτης Καταστάσεων είναι αυτός της **Εικόνας 2.5**.

Η αρχική του *κατάσταση* είναι η F. Βλέπουμε ότι προς την F υπάρχει μια μετάβαση η οποία δεν προέρχεται από κάποια άλλη κατάσταση. Αντίστοιχα βλέπουμε ότι από την *κατάσταση* F υπάρχουν δυο εξερχόμενες μεταβάσεις προς τις *καταστάσεις* E και G αντίστοιχα. Η μια μετάβαση έχει τον κανόνα k και η άλλη τον κανόνα e. Ανάλογα με το ποιο γεγονός θα συμβεί πρώτο, το σύστημα θα μεταβεί στην αντίστοιχη *κατάσταση*. Όταν το σύστημα βρεθεί στην *κατάσταση* I υπάρχουν δυο εξερχόμενες μεταβάσεις οι οποίες ενεργοποιούνται με το ίδιο γεγονός e. Αυτό που θα καθορίσει ποια μετάβαση θα επιλεχθεί τελικά είναι ποια *συνθήκη* εκ των c1,c2 θα είναι αληθής.




**Εικόνα 2.5: Παράδειγμα Χάρτη Καταστάσεων**

Δυο σημαντικά χαρακτηριστικά στην δομή ενός Χάρτη Καταστάσεων είναι οι *εμφωλευμένες καταστάσεις* (nested states) και οι *ορθογώνιες συνιστώσες* (orthogonal components). Η εμφώλευση «καταστάσεων» πρακτικά σημαίνει ότι μια «κατάσταση» μπορεί να «περιέχει» έναν ολόκληρο Χάρτη Καταστάσεων. Η σημασιολογία αυτής της εμφώλευσης είναι ότι με την είσοδο στην *κατάσταση* υψηλού επιπέδου αυτόματα γίνεται είσοδος και στην *κατάσταση* χαμηλού επιπέδου. Η εμφώλευση *καταστάσεων* συντελεί στην δημιουργία κομψότερων και απλούστερων ορισμών κατά την διαδικασία της σχεδίασης. Η ύπαρξη ορθογώνιων συνιστωσών δηλώνει την παράλληλη εκτέλεση δυο Χαρτών Καταστάσεων που περιέχονται στον ίδιο υψηλού επιπέδου Χάρτη. Και οι




**Εικόνα 2.6: Αναπαράσταση ενός Χάρτη Καταστάσεων σαν Δένδρο Καταστάσεων**

δυο συνιστώσες εισέρχονται στην αρχική τους κατάσταση ταυτόχρονα και οι μεταβάσεις τους εκτελούνται παράλληλα, ανάλογα βέβαια με τους κανόνες που τις διέπουν.

Ένας Χάρτης Καταστάσεων μπορεί να αναπαρασταθεί και ως ένα *δένδρο καταστάσεων* (**Εικόνα 2.6**). Σε αυτό το δένδρο οι *απλές* ή *βασικές* καταστάσεις όπως ονομάζονται (αυτές δηλαδή που απλά αντιστοιχούν στην εκτέλεση κάποιας *ενέργειας*) αντιστοιχούν σε κόμβους *φύλλα* (LEAF), οι *εμφωλευμένες καταστάσεις* αντιστοιχούν σε κόμβους *αποκλειστικής διάζευξης* (XOR) και οι *ορθογώνιες συνιστώσες* σε *κόμβους σύζευξης* (AND).Η διακεκομμένη γραμμή δηλώνει ότι οι καταστάσεις C και D είναι ορθογώνιες συνιστώσες της υψηλότερου επιπέδου κατάστασής B. Επίσης η *κατάσταση* I είναι εμφωλευμένη μέσα στην υψηλότερου επιπέδου *κατάσταση* H.

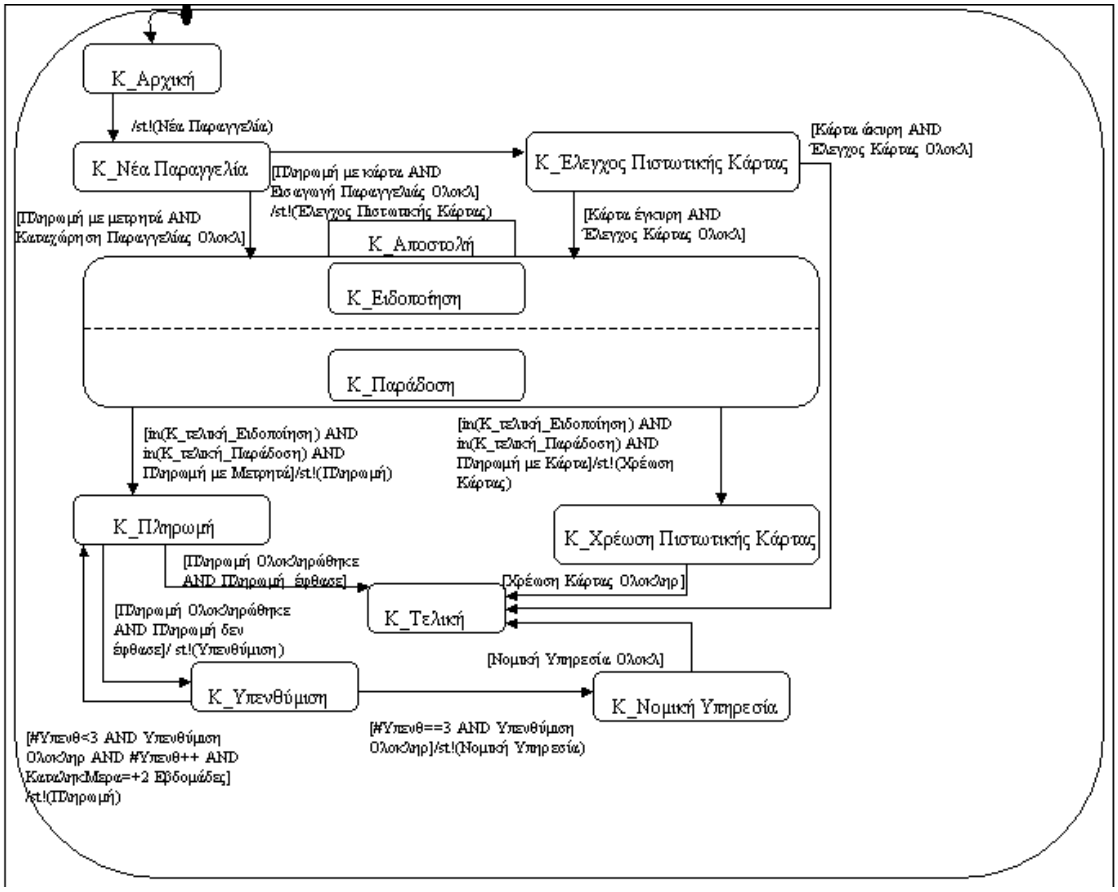
#### *2.1.4.2.2 Μοντελοποίηση μιας Ροής Εργασίας με Χάρτες Καταστάσεων.*

Στην **Εικόνα 2.7** παρουσιάζουμε το παράδειγμα ηλεκτρονικού εμπορίου της **Εικόνας 2.2** μοντελοποιημένο με την χρήση Χάρτη Καταστάσεων. Οι *ενέργειες* που εκτελούνται για την ολοκλήρωση της Ροής είναι οι «Έλεγχος Πιστωτικής Κάρτας», «Πληρωμή», «Χρέωση Κάρτας», «Υπενθύμιση» και «Νομική Υπηρεσία». Η ονομασία Κ_ΌνομαΚατάστασης είναι συντομογραφική για το «Κατάσταση ΌνομαΚατάστασης». Όπως βλέπουμε δεν υπάρχουν *γεγονότα* που να πυροδοτούν τις μεταβάσεις αλλά αντίθετα αυτές πυροδοτούνται όταν οι συνθήκες τους γίνουν αληθείς.

Σε μια *δράση* το κατηγόρημα !st(A) όπου A είναι μια *ενέργεια* σημαίνει την εκτέλεση αυτής της *ενέργειας*. Σε μια *συνθήκη* το κατηγόρημα in(S) -όπου S είναι μια *κατάσταση* - είναι «Αληθές» όσο το σύστημα βρίσκεται στην *κατάσταση* S. Στην **Εικόνα 2.7** η *κατάσταση* Κ_Αποστολή περιλαμβάνει τις ορθογώνιες συνιστώσες Κ_Ειδοποίηση και Κ_Παράδοση των οποίων οι Χάρτες Καταστάσεων δεν δίνονται χάριν συντομίας. Όπως έχει προαναφερθεί οι δυο αυτοί Χάρτες Καταστάσεων θα εκτελεστούν παράλληλα. Οι *καταστάσεις* Κ_Τελική_Ειδοποίηση και Κ_Τελική_Παράδοση, αναφέρονται στις τελικές *καταστάσεις* του καθενός εκ των δυο αυτών Χαρτών.

### 2.1.4.3 Σύγκριση Χαρτών Καταστάσεων και Δικτύων Petri

Τόσο ο φορμαλισμός των Δικτύων Petri όσο και των Χαρτών Καταστάσεων είναι τυπικοί και με καλά ορισμένο μαθηματικό υπόβαθρο εξασφαλίζοντας μας έτσι σημασιολογικά ορθούς ορισμούς για την περιγραφή Ροών Εργασίας. Εντούτοις τα ιδιαίτερα




**Εικόνα 2.7: Μοντελοποίηση της Ροής Εργασίας του παραδείγματος ηλεκτρονικού εμπορίου με Χάρτη Κατάστασης.**

χαρακτηριστικά του καθένας τον καθιστούν λιγότερο ή περισσότερο κατάλληλο για συγκεκριμένες κατηγορίες προβλημάτων.

Μία από τις βασικές επιδιώξεις από την χρήση Ροών Εργασίας για την υλοποίηση επιχειρησιακών διαδικασιών είναι ή επίτευξη του μεγαλύτερου δυνατού βαθμού παραλληλισμού στην εκτέλεση των ενεργειών. Μια διαφορά ανάμεσα στα Δίκτυα Petri και στους Χάρτες Καταστάσεων έγκειται στον τρόπο με τον οποίο αναπαριστούν τον παραλληλισμό ενεργειών. Τα Δικτύα Petri παρέχουν εγγενή υποστήριξη του παραλληλισμού. Οποιεσδήποτε δύο *μεταβάσεις* έχουν ένα κοινό *σημείο εισόδου* με αρκετά *κουπόνια* ώστε να ενεργοποιήσει και τις δύο *μεταβάσεις*, αυτές θα εκτελεστούν παράλληλα. Οι Χάρτες Καταστάσεων μοντελοποιούν τον παραλληλισμό με την χρήση ορθογώνιων συνιστωσών. Κάθε ορθογώνια συνιστώσα αποτελεί ένα ξεχωριστό Χάρτη Καταστάσεων και δεν επιτρέπεται να γίνει μετάβαση από μία κατάσταση τής μίας συνιστώσας σε μία κατάσταση της άλλης. Η επικοινωνία μεταξύ των δύο συνιστωσών μπορεί να γίνει μόνο μέσω μηνυμάτων ( δηλαδή

*γεγονότων*). Αυτά τα χαρακτηριστικά περιορίζουν την ευελιξία της χρήσης παραλληλισμού σε Χάρτες Καταστάσεων σε σύγκριση με τα Δίκτυα Petri.

Η ύπαρξη των *κουπονιών* σε ένα Δίκτυο Petri καθιστά δυνατή την ταυτόχρονη εκτέλεση πολλών περιπτώσεων (cases) του ίδιου Δικτύου. Πρακτικά ο αριθμός των *κουπονιών* στο *αρχικό σημείο* του Δικτύου Petri απεικονίζει τον αριθμό των σεναρίων που μπορεί το σύστημα να εκτελέσει ταυτόχρονα. Ο διαχωρισμός των διαφορετικών περιπτώσεων μεταξύ τους βάσει των κουπονιών έγκειται στον διαχωρισμό των διάφορων κουπονιών μεταξύ τους χρησιμοποιώντας την έννοια του χρώματος, χαρακτηριστικό των Χρωματισμένων Δικτύων Petri. Από την άλλη, με την χρήση Χαρτών Καταστάσεων μπορεί να προσομοιωθεί η εκτέλεση μόνο ενός σεναρίου κάθε φορά.

Κατά την διάρκεια της εκτέλεσης ενός Χάρτη Καταστάσεων, κάθε στιγμή η κατάσταση του συστήματος απεικονίζεται από ένα από τα στοιχεία τής διαγραμματικής απεικόνισης του Χάρτη. Αυτό σημαίνει ότι κάθε στιγμή η κατάσταση της εκτέλεσης του Χάρτη Καταστάσεων και επομένως της Ροής Εργασίας που υλοποιείται με αυτόν είναι άμεσα καθορισμένη με ένα προφανή και διαισθητικό τρόπο. Αντίθετα σε ένα Δίκτυο Petri η κατάσταση εκτέλεσης μίας *περίπτωσης* απεικονίζεται απο την κατανομή των κουπονιών της *περίπτωσης* στο Δίκτυο. Από την άποψη λοιπόν της παρακολούθησης της εκτέλεσης μίας Ροής Εργασίας οι Χάρτες Καταστάσεων πλεονεκτούν των Δικτύων Petri.

Σημαντικός παράγοντας στην επιλογή ενός τυπικού φορμαλισμού για την σχεδίαση και την υλοποίηση Ροών Εργασίας είναι το πλήθος τών εργαλείων που έχουν αναπτυχθεί και τον υποστηρίζουν. Εξαιρουμένου του εργαλείου STATEMATE [H90] οι Χάρτες Καταστάσεων δεν χαίρουν ιδιαίτερης υποστήριξης από εμπορικά ή ερευνητικά εργαλεία. Αντίθετα υπάρχει πληθώρα εργαλείων τόσο για την σχεδίαση, τον έλεγχο και την προσομοίωση Δικτύων Petri χαμηλού επιπέδου όσο και για τις επεκτάσεις τους, δηλαδή τα Χρωματισμένα, τα Χρονικά και τα Ιεραρχικά Δίκτυα Petri [PN].

### 2.1.5 Διεπιχειρησιακές Ροές Εργασίας.

Παραδοσιακά τα Συστήματα Διαχείρισης Ροών Εργασίας εστίαζαν την λειτουργικότητα τους σε ομογενή περιβάλλοντα μέσα στα όρια ενός οργανισμού.

Στις μέρες μας όμως πολλοί οργανισμοί συνάπτουν συνεργασίες για ανταποκριθούν στις αυξανόμενες απαιτήσεις της αγοράς για πολύπλοκες και εξειδικευμένες υπηρεσίες. Οι εταιρίες εστιάζουν τις επιχειρησιακές τους διαδικασίες στον κύριο επιχειρησιακό τους

στόχο «προωθώντας» (outsourcing) ενέργειες -βοηθητικές ή δευτερεύουσας σημασίας- σε άλλες εταιρίες ή οργανισμούς. Με αυτό τον τρόπο δημιουργείται μια αλυσίδα από εταιρίες ή οργανισμούς οι οποίοι συνεργάζονται στενά για την επίτευξη ενός κοινού συνολικού στόχου συνθέτοντας έτσι μία *Εικονική Εταιρία* ή ένα *Εικονικό Οργανισμό*.

Στα πλαίσια μίας στενής συνεργασίας μεταξύ εταιριών που συνδυάζουν τις προσπάθειες τους με σκοπό την επίτευξη ενός κοινού στόχου, οι επιχειρησιακές διαδικασίες πρέπει να ξεπερνάνε τα όρια ενός οργανισμού. Συστήματα Διαχείρισης Ροών Εργασίας σε διαφορετικούς οργανισμούς πρέπει να έχουν την δυνατότητα να «συνεργάζονται» στην δημιουργία διεπιχειρησιακών Ροών Εργασίας (cross-organizational workflows). Το νέο αυτό πλαίσιο δημιουργίας Ροών Εργασίας πρέπει να επιτρέπει την «ενοποίηση» Ροών Εργασίας διαφορετικών οργανισμών οι οποίοι μπορεί να έχουν ετερογενή περιβάλλοντα διατηρώντας πάντα την αυτονομία του κάθε οργανισμού.

Αυτή η πρακτική, προώθησης δευτερεύουσας σημασίας ενεργειών, σε άλλους οργανισμούς, εισάγει ένα παράδειγμά καταναλωτή/ παραγωγού μεταξύ των οργανισμών. Ο οργανισμός που θέλει μια ενέργεια να εκτελεστεί εκ μέρους του μπορεί να θεωρηθεί ο καταναλωτής ενώ ο οργανισμός που εκτελεί μια ενέργεια εκ μέρους κάποιου άλλου οργανισμού είναι ο παραγωγός. Σε ένα γενικότερο πλαίσιο όπου μία εταιρία αναζητά επιχειρησιακούς συνεργάτες (business partners) για να αναθέσει κάποια δευτερεύουσα ενέργεια της (ή υπηρεσία της όπως συχνά αποκαλείται), απαιτούνται ευκολίες ταιριάσματος (matchmaking facility) προμηθευτών και καταναλωτών υπηρεσιών συγκροτώντας έτσι «αγορές υπηρεσιών» (service marketplaces). Οι παραγωγοί υπηρεσιών «διαφημίζουν» τις υπηρεσίες που παρέχουν ενώ αντίστοιχα οι καταναλωτές υπηρεσιών επερωτούν υπηρεσίες οι οποίες ταιριάζουν στις προδιαγραφές τους. Σε αυτή την αλυσίδα από οργανισμούς είναι επόμενο ότι κάποιοι οργανισμοί που παίζουν τον ρόλο παραγώγου κάποιας υπηρεσίας να παίζουν παράλληλα και τον ρόλο καταναλωτή μίας άλλης υπηρεσίας την οποία χρειάζονται οι ίδιοι για να υλοποιήσουν την επιχειρησιακή τους διαδικασία. Για να γίνει εφικτό αυτό το ταίριασμα ζητούμενων και προσφερόμενων υπηρεσιών είναι απαραίτητο οι καταναλωτές και οι παραγωγοί να έχουν συμφωνήσει στις γλώσσες και στους τρόπους περιγραφής των υπηρεσιών τους, τον τρόπο με τον οποίο δηλαδή οι υπηρεσίες θα είναι προσβάσιμες στους άλλους. Τέτοιου είδους συμφωνίες επιτυγχάνονται με την μορφή «*συμβολαίων*»[NJLRC].

Από εδώ και στο εξής, τις υπηρεσίες αυτές οι οποίες είναι προσβάσιμες μέσω του διαδικτύου με μια δεδομένη και προσυμφωνημένη διεπαφή, και με ένα δεδομένο και

προσυμφωνημένο πρωτόκολλο και που μπορούν να εντοπιστούν σε κάποια από τις δεδομένες και προσυμφωνημένες «αγορές υπηρεσιών» θα τις ονομάζουμε Ηλεκτρονικές Υπηρεσίες.

## *2.2 Ηλεκτρονικές Υπηρεσίες*

Η ανάπτυξη των τεχνολογιών του Διαδικτύου έχει αλλάξει σημαντικά τον τρόπο με τον οποίο λειτουργούν οι επιχειρήσεις. Οι εταιρίες «εξάγουν» τις κύριες λειτουργίες τους στον Παγκόσμιο Ιστό προσδοκώντας επιπλέον αυτοματοποίηση, αποδοτικότερες επιχειρησιακές διαδικασίες και καθολική ορατότητα των λειτουργιών τους. Για να επιζήσουν οι εταιρίες στις συνθήκες πολύ σκληρού ανταγωνισμού που έχουν δημιουργηθεί πρέπει να επιλέξουν τις σωστότερες και πιο ευέλικτες επιχειρησιακές λύσεις που θα τους επιτρέψουν αφενός να χρησιμοποιήσουν τις υπάρχουσες εφαρμογές με τον καλύτερο δυνατό τρόπο και αφετέρου θα τους επιτρέψουν να εξελίσσονται και να αλλάζουν.

Στις μέρες, υπάρχει μια αλλαγή κατεύθυνσης, όσων αφορά την συνεργασία εφαρμογών, από τα στενά συνεργαζόμενα συστήματα(π.χ. DCOM[DCOM]) σε συστήματα με πιο χαλαρά συνδεδεμένες και δυναμικά προσαρμοζόμενες συνιστώσες (π.χ Jini[JINI], Enterprise Java Beans[EJB]) με τελευταία παρουσία σε αυτή την κατηγορία το παράδειγμα των Ηλεκτρονικών Υπηρεσιών.

Οι Ηλεκτρονικές Υπηρεσίες είναι *αυτόνομες* (self-contained) και *αρθρωτές* (modular) εφαρμογές, προσβάσιμες μέσω του Παγκόσμιου Ιστού παρέχοντας ένα σύνολο από λειτουργικότητες σε επιχειρήσεις ή άτομα. Αυτό που κάνει τις Ηλεκτρονικές Υπηρεσίες να ξεχωρίζουν σαν πρόταση είναι η δυνατότητα *ανακάλυψης* Ηλεκτρονικών Υπηρεσιών που να ταιριάζουν στις απαιτήσεις μίας εφαρμογής, η δυνατότητα διαπραγμάτευσης των συμβολαίων χρήσης τους και η καθολική προσβασιμότητα τους από οποιοδήποτε σημείο, οποιαδήποτε στιγμή.

Το παράδειγμα των Ηλεκτρονικών Υπηρεσιών μπορεί να θεωρηθεί σαν επέκταση του οντοκεντρικού παραδείγματος αφού στοιχεία όπως η ενθυλάκωση, το πέρασμα μηνυμάτων και η δυναμική δέσμευση υπηρεσιών, είναι πρωτεύουσας σημασίας. Εντούτοις το παράδειγμα τους επεκτείνεται πέρα από τις απλές «υπογραφές μεθόδων» (method signatures) αφού η περιγραφή (ή διεπαφή) μιας Ηλεκτρονικής Υπηρεσίας περιλαμβάνει πληροφορία για την λειτουργικότητα της υπηρεσίας (τι κάνει), την τοποθεσία της υπηρεσίας (που είναι), τον τρόπο με τον οποίο επικαλείται η υπηρεσία, την ποιότητα της υπηρεσίας (quality of service) και τις πολιτικές ασφάλειας που

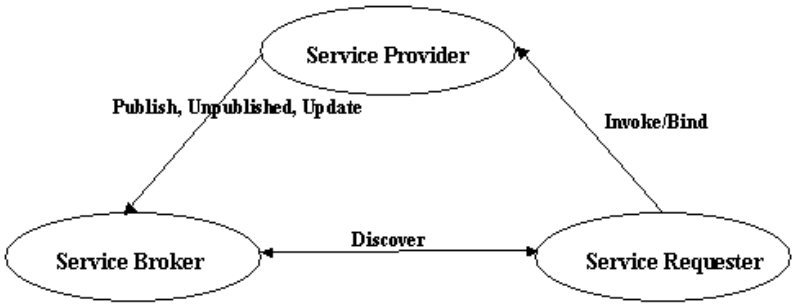
σχετίζονται με αυτήν. Αυτά τα χαρακτηριστικά αποδίδουν στις Ηλεκτρονικές Υπηρεσίες τον χαρακτηρισμό *αυτοπεριγραφόμενες* (self-describing). Αντίστοιχα το παράδειγμα Ηλεκτρονικών Υπηρεσιών μπορεί να θεωρηθεί και σαν επέκταση του παραδείγματος συνιστωσών λογισμικού, αφού κάθε Ηλεκτρονική Υπηρεσία μπορεί να θεωρηθεί σαν μία συνιστώσα, ανεξάρτητη πλατφόρμας που δίνει την δυνατότητα χαλαρής σύνδεσης (loose coupling) με εφαρμογές ή άλλες Ηλεκτρονικές Υπηρεσίες.

### 2.2.1 Η λειτουργικότητα των Ηλεκτρονικών Υπηρεσιών

Οι Ηλεκτρονικές Υπηρεσίες αποτελούν ένα νέο μοντέλο για τη χρήση του Παγκόσμιου Ιστού. Το μοντέλο αυτό επιτρέπει την έκδοση επιχειρησιακών λειτουργιών στο Διαδίκτυο και την καθολική πρόσβαση σε αυτές τις λειτουργίες. Τόσο προγραμματιστές όσο και απλοί χρήστες μπορούν να εκμεταλλευτούν τα οφέλη τών Ηλεκτρονικών Υπηρεσιών. Το μοντέλο των Ηλεκτρονικών Υπηρεσιών απλοποιεί την ανάπτυξη και την διαλειτουργικότητα επιχειρησιακών εφαρμογών ενώ επιτρέπει ακόμα και σε απλούς χρήστες να δημιουργήσουν τις δικές τους Ηλεκτρονικές Υπηρεσίες ή να ανακαλύψουν Ηλεκτρονικές Υπηρεσίες μέσω διαισθητικών και χρηστικών διεπαφών σε κοινούς φυλλομετρητές (browsers) του Διαδικτύου.

Οι βασικές λειτουργίες που σχετίζονται με το μοντέλο Ηλεκτρονικών Υπηρεσιών είναι η δημιουργία της Ηλεκτρονικής Υπηρεσίας (Web Service creation), η περιγραφή της Ηλεκτρονικής Υπηρεσίας (Web Service description), η έκδοση της Ηλεκτρονικής Υπηρεσίας (Web Service publishing), η ανακάλυψη της Ηλεκτρονικής Υπηρεσίας από πιθανούς χρήστες (Web Service discovery), η δέσμευση με τα κατάλληλα πρωτόκολλα και η επίκληση της Ηλεκτρονικής Υπηρεσίας (Web Service binding/invocation) και τέλος η αναίρεση της έκδοσης της Ηλεκτρονικής Υπηρεσίας (Web Service unpublishing) σε περίπτωση που δεν ικανοποιεί πλέον τις απαιτήσεις δημιουργίας της. Σε αυτές τις βασικές λειτουργίες προστίθενται ακόμα αρκετές άλλες έτσι ώστε να παρέχεται η δυνατότητα για ανάπτυξη πολύπλοκων και σύνθετων Ηλεκτρονικών Υπηρεσιών. Τέτοιες ενέργειες είναι η σύνθεση (composition), η διαχείριση (management) και η παρακολούθηση (monitoring) Ηλεκτρονικών Υπηρεσιών. Οι βασικές λειτουργίες μπορούν να οργανωθούν σε τρεις κατηγορίες ανάλογα με το ποίος τις εκτελεί. Αντίστοιχα μπορούν να θεωρηθούν τρεις ρόλοι, δηλαδή τρία «είδη» οντοτήτων οι οποίες αναλαμβάνουν την εκτέλεση ενός συγκεκριμένου είδους ενεργειών. Οι ρόλοι αυτοί είναι, ο αιτών μίας υπηρεσίας (service requester), ο

παροχέας μίας υπηρεσίας (service provider), και ο διαμεσολαβητής υπηρεσιών (service broker)[WSA] (**Εικόνα 2.8**).




**Εικόνα 2.8: Οι αλληλεπιδράσεις των ρόλων στο μοντέλο Ηλεκτρονικών Υπηρεσιών.**

Ο παροχέας μίας υπηρεσίας είναι η οντότητα που παρέχει συγκεκριμένες εφαρμογές λογισμικού με την μορφή Ηλεκτρονικών Υπηρεσιών. Οι παροχείς υπηρεσιών εκδίδουν, αναιρούν την έκδοση και ανανεώνουν τις υπηρεσίες τους έτσι ώστε να είναι διαθέσιμες τις περισσότερες φορές μέσω του Διαδικτύου. Από επιχειρηματική σκοπιά, ο παροχέας μίας υπηρεσίας μπορεί να θεωρηθεί ως ο ιδιοκτήτης των υπηρεσιών ενώ από άποψη αρχιτεκτονικής ο παροχέας είναι η πλατφόρμα που χρησιμοποιείται για την υλοποίηση των εφαρμογών με την μορφή υπηρεσιών.

Ο αιτών μίας υπηρεσίας είναι η οντότητα που έχει μια λειτουργική ανάγκη και την οποία προσπαθεί να ικανοποιήσει αναζητώντας την κατάλληλη υπηρεσία στο Διαδίκτυο. Από επιχειρηματικής σκοπιάς μπορεί να είναι μία εταιρία που θέλει να προωθήσει (outsource) μια ανάγκη της σε ένα τρίτο ενώ από αρχιτεκτονικής σκοπιάς είναι μια εφαρμογή που απλά ψάχνει μια άλλη εφαρμογή για να την χρησιμοποιήσει. Ο αιτών μπορεί να είναι οτιδήποτε, από ένα χρήστη ο οποίος προσπελαύνει μία υπηρεσία μέσω του υπολογιστή του, μέχρι μια εφαρμογή ή και μία άλλη Ηλεκτρονική Υπηρεσία. Ο αιτών αναζητά τις επιθυμητές Ηλεκτρονικές Υπηρεσίες μέσω του διαμεσολαβητή υπηρεσιών και έπειτα τις δεσμεύει στον παροχέα όπως φαίνεται από την **Εικόνα 2.8**.

Ο διαμεσολαβητής υπηρεσιών είναι μια οντότητα η οποία παρέχει μία επερωτήσιμη «αποθήκη περιγραφών υπηρεσιών», δηλαδή μιας μορφής καταλόγου και μπορεί να δεχθεί και να απαντήσει σε επερωτήσεις σχετικές με την ύπαρξη Ηλεκτρονικών Υπηρεσιών που πληρούν συγκεκριμένες προδιαγραφές. Οι παροχείς περιγράφουν τις υπηρεσίες τους σε αυτόν το κατάλογο ενώ οι αιτούντες αναζητούν τις

υπηρεσίες που θέλουν και ανακτούν τόσο την περιγραφή τους όσο και την απαραίτητη πληροφορία για να την δεσμεύσουν από τον παροχέα της. Χαρακτηριστικά παραδείγματα τέτοιων καταλόγων είναι το UDDI ( Universal Description Discovery Integration)[UDDI] και το Xmethods[XM]

### 2.2.2 Οφέλη από τη χρήση Ηλεκτρονικών Υπηρεσιών

Η χρήση του παραδείγματος των Ηλεκτρονικών Υπηρεσιών για την δημιουργία σύνθετων εφαρμογών έχει μια σειρά από πλεονεκτήματα σε σχέση με τον παραδοσιακό τρόπο ανάπτυξης εφαρμογών.

- **Ευκολότερη και ταχύτερη δημιουργία**. Επιχειρήσεις που χρησιμοποιούν το μοντέλο των Ηλεκτρονικών Υπηρεσιών, μπορούν να δημιουργούν νέες υπηρεσίες χωρίς την επένδυση και τις καθυστερήσεις των παραδοσιακών μοντέλων. Αυτό οφείλεται κυρίως στην δημιουργία νέων Ηλεκτρονικών Υπηρεσιών *επαναχρησιμοποιώντας* ή *συνδυάζοντας* ήδη υπάρχουσες Ηλεκτρονικές Υπηρεσίες.
- **Διαλειτουργικότητα**. Μια Ηλεκτρονική Υπηρεσία μπορεί να αλληλεπιδρά με άλλες Ηλεκτρονικές Υπηρεσίες. Χρησιμοποιώντας δεδομένες διεπαφές βασισμένες στην XML (π.χ. WSDL) και δεδομένα πρωτόκολλα για την επικοινωνία, δίνεται η δυνατότητα σε εταιρίες ή απλούς προγραμματιστές να παράγουν ή να καταναλώνουν Ηλεκτρονικές Υπηρεσίες διατηρώντας αμετάβλητο το περιβάλλον ανάπτυξης εφαρμογών τους. Με αυτή τους την ιδιότητα οι Ηλεκτρονικές Υπηρεσίες «ενοποιούν» οσοδήποτε ετερογενή περιβάλλοντα.
- **«Επί τόπου» ανάπτυξη**. Παραδοσιακές συνεργατικές εφαρμογές, δηλαδή εφαρμογές που συνθέτονταν από κάποιες απλούστερες χαρακτηρίζονται από την έλλειψη ευελιξίας που συνεπάγεται τόσο η «σφιχτή σύνδεση» (tight coupling) μεταξύ των εφαρμογών όσο και η στατική δέσμευση (static binding) μεταξύ τους κατά την σχεδίαση (build-time). Οποιαδήποτε αλλαγή σε μία από τις «υπο-εφαρμογές» προκαλούσε κατάρρευση της συνολικής εφαρμογής και ανάγκη επαναπρογραμματισμού της για να γίνει συμβατή με τα νέα δεδομένα. Το μοντέλο των Ηλεκτρονικών Υπηρεσιών υποστηρίζει την *αποσύνδεση* (decoupling) μεταξύ των υπηρεσιών, και την «επί τόπου» ανάπτυξη εφαρμογών και υπηρεσιών *ανακαλύπτοντας* και *συνθέτοντας* δυναμικά (κατά το δυνατό) υπηρεσίες διαθέσιμες στο Διαδίκτυο. Η χαλαρή σύνδεση (loose coupling) και η δυναμική δέσμευση (dynamic binding) συμβάλει στο γεγονός ότι μία υπηρεσία που χρησιμοποιεί κάποια άλλη υπηρεσία για να ικανοποιήσει μια ανάγκη της, μπορεί ανά πάσα στιγμή να την αντικαταστήσει με κάποια άλλη που την ικανοποιεί.

- **Ελάττωση της πολυπλοκότητας λόγω ενθυλάκωσης**. Σε ένα περιβάλλον Ηλεκτρονικών Υπηρεσιών, όλες οι συνιστώσες είναι Ηλεκτρονικές Υπηρεσίες. Σημασία έχει η «συμπεριφορά» που έχει η υπηρεσία και όχι ο τρόπος που είναι υλοποιημένη αυτή η συμπεριφορά. Αυτό το στοιχείο ενθυλάκωσης, η εικόνα δηλαδή των Ηλεκτρονικών Υπηρεσιών σαν «μαύρα κουτιά» με συγκεκριμένη συμπεριφορά, διευκολύνει κατά πολύ το έργο των προγραμματιστών που δεν ασχολούνται πλέον με τις λεπτομέρειες των εφαρμογών που επικαλούνται. Έτσι η πολυπλοκότητα της ανάπτυξης Ηλεκτρονικών Υπηρεσιών ελαττώνεται σημαντικά.

### 2.2.3 Πρότυπα σχετικά με Ηλεκτρονικές Υπηρεσίες

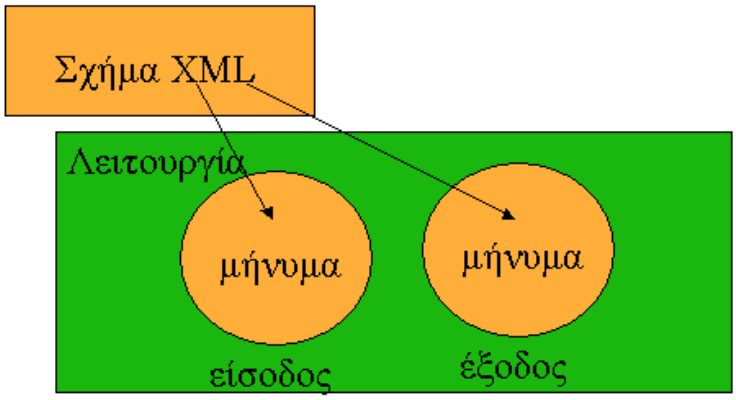
Είναι προφανές ότι από την στιγμή που οι παροχείς, οι αιτούντες, και οι διαμεσολαβητές υπηρεσιών αλληλεπιδρούν, πρέπει να υποστηρίζουν όλοι τις ίδιες «πρότυπες» (standard) τεχνολογίες για περιγραφή υπηρεσιών, επικοινωνία και δέσμευση δεδομένων. Η ύπαρξη των «προτύπων» (standards) επιτρέπει στους προγραμματιστές να δημιουργούν Ηλεκτρονικές Υπηρεσίες με τρόπο ανεξάρτητο πλατφόρμας και γλώσσας υλοποίησης. Τα τρία κυριότερα πρότυπα, σχετικά με τις Ηλεκτρονικές Υπηρεσίες είναι το πρωτόκολλο SOAP[SOAP] για την επικοινωνία μεταξύ υπηρεσιών, η γλώσσα WSDL [WSDL] για την περιγραφή υπηρεσιών και το UDDI για την αποθήκευση περιγραφών υπηρεσιών.

#### 2.2.3.1 WSDL

Η καθολική ορατότητα (global visibility) και προσβασιμότητα (reachability) μίας Ηλεκτρονικής Υπηρεσίας προϋποθέτει μία αυστηρά καθορισμένη και προσυμφωνημένη προγραμματιστική διεπαφή όπως η WSDL. Η WSDL είναι μια XML γραμματική που καθορίζει ιδιότητες μίας Ηλεκτρονικής Υπηρεσίας όπως *τι* κάνει, *πού* βρίσκεται και *πώς* μπορεί να επικληθεί. Ένα WSDL έγγραφο περιγράφει τις υπηρεσίες σαν συλλογές από σημεία στο δίκτυο (network endpoints) που ονομάζονται πόρτες (ports). Στην WSDL τα δεδομένα που ανταλλάσσονται αναπαρίστανται με την μορφή μηνυμάτων (messages) και σύνολα από *λειτουργίες* ομαδοποιούνται με την αφαίρεση του *τύπου θύρας* (port type). Οι ορισμοί των *μηνυμάτων* και των *τύπων θύρας* είναι διαχωρισμένοι με την δέσμευση τους σε συγκεκριμένους τύπους δεδομένων και στοιχείων δικτύου αντίστοιχα, επιτρέποντας έτσι την επαναχρησιμοποίηση των ορισμών τους. Ο συνδυασμός συγκεκριμένων στοιχείων δικτύου (πρωτοκόλλων) με ένα ορισμό τύπου δεδομένων, για ένα συγκεκριμένο *τύπο θύρας* αποτελεί μία (επαναχρησιμοποιήσιμη) *δέσμευση* (binding). Σαν παράδειγμα

της «επαναχρησιμοποίησης», ο ίδιος τύπος θύρας μπορεί να χρησιμοποιηθεί για την δημιουργία δύο δεσμεύσεων για την ίδια λειτουργία, μία με το πρωτόκολλο HTTP και μία με το πρωτόκολλο FTP Ο συνδυασμός μίας *δέσμευσης* με ένα *διεύθυνση δικτύου* συνθέτουν μία *θύρα* ενώ μια συλλογή από *πόρτες* ορίζει πρακτικά μία *Ηλεκτρονική Υπηρεσία*. Επομένως ένα WSDL έγγραφο σε XML χρησιμοποιεί τα παρακάτω εφτά βασικά στοιχεία για να περιγράψει Ηλεκτρονικές Υπηρεσίες.

- **Τύπος** (Type). Το στοιχείο *τύπος* καθορίζει τον τύπο δεδομένων που χρησιμοποιείται από την Ηλεκτρονική Υπηρεσία και παίρνει τιμές από ένα σύστημα τύπων. Για επιπλέον ανεξαρτησία η WSDL χρησιμοποιεί το σύστημα τύπων του *Σχήματος XML*. Ο *τύπος* μπορεί να είναι είτε απλός (συμβολοσειρά, ακέραιος) είτε σύνθετος (στοιχείο XML).
- **Μήνυμα** (Message). Το *μήνυμα* αποτελεί την αφαίρεση για τα δεδομένα που ανταλλάσσονται μεταξύ υπηρεσιών. Ένα *μήνυμα* μπορεί να αποτελείται από ένα ή περισσότερα *μέρη (parts)* τα οποία έχουν κατάλληλους τύπους δεδομένων. Τα *μέρη* μπορούν να αντιστοιχισθούν διαισθητικά στις παραμέτρους κατά την κλήση μίας συνάρτησης σε ένα προγραμματιστικό περιβάλλον.
- **Λειτουργία** (Operation). Το στοιχείο *λειτουργία* είναι η αφαίρεση για μία από τις λειτουργίες που παρέχει η υπηρεσία. Η σύνθεση μίας λειτουργίας από μηνύματα και τύπους φαίνεται στην **Εικόνα 2.9**. Το επίπεδο της αφαίρεσης που χρησιμοποιείται διατηρεί μια «χαλαρή σύνδεση» μεταξύ λειτουργίας και μηνυμάτων και μεταξύ μηνυμάτων και τύπων δεδομένων.



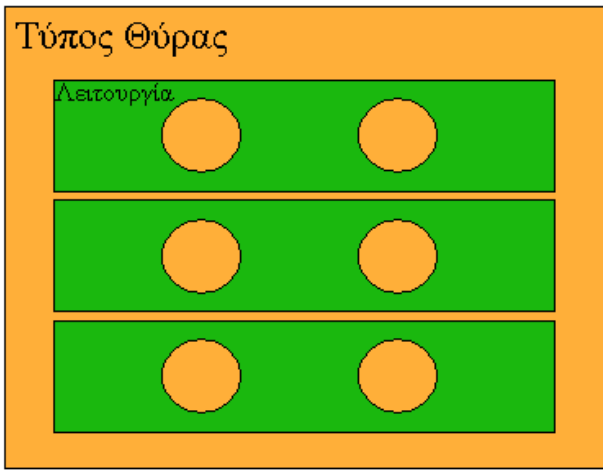

```
<message name="InBidRequest">
   <part name="symbol" type="xsd:string"/>
   <part name="shareCount" type="xsd:int"/>
</message>
```

```
<operation name="bid">
   <input message="InBidRequest"/>
   <output message="OutBidResponse"/>
</operation>
```

**Εικόνα 2.9: Ορισμός μίας λειτουργία στην WSDL.**

- **Τύπος Θύρας.** (Port type). Το στοιχείο *τύπος θύρας* είναι η βασική αφαίρεση της WSDL. Ο *τύπος θύρας* περιγράφει ένα σύνολο από επιμέρους λειτουργίες, δηλαδή πρακτικά την λειτουργικότητα της Ηλεκτρονικής Υπηρεσίας. Αυτό που μένει πλέον για την σύνθεση μίας ολοκληρωμένης Ηλεκτρονικής Υπηρεσίας είναι ο συνδυασμός του τύπου θύρας με πραγματικά πρωτόκολλα επικοινωνίας και τύπους δεδομένων. Στην **Εικόνα 2.10** βλέπουμε ένα *τύπο θύρας*.

```
<portType name="StockPortfolioPortType">
  <operation name="bid">
    <input message="InBidRequest"/>
    <output message="OutBidResponse"/>
  </operation>
  <operation name="verifyCredit">
    <input message="InCheckCreditRequest"/>
    <output message="OutCheckCreditResponse"/>
  </operation>
  <operation name="retrieveStockQuote">
    <input message="InRetrieveStockQuote"/>
    <output message="OutRetrieveStockQuote"/>
  </operation>
</portType>
```



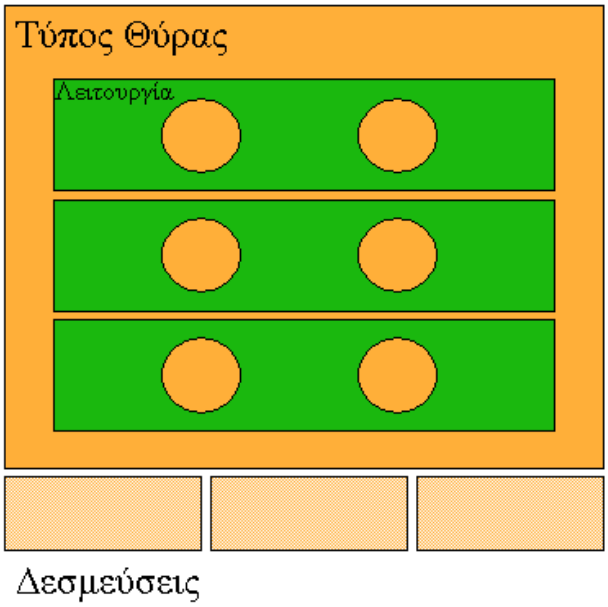

**Εικόνα 2.10: Ορισμός ενός τύπου θύρας στην WSDL**

Στον τύπο θύρας της **Εικόνας 2.10** περιγράφονται τρεις λειτουργίες καθεμία από τις οποίες, δέχεται ένα μήνυμα εισόδου (από αυτόν που την καλεί) και παράγει ένα μήνυμα εξόδου το οποίο θα επιστραφεί σε αυτόν που την κάλεσε. Αυτός ο τρόπος αλληλεπίδρασης που ονομάζεται αίτηση-απάντηση (request-response) είναι ο ποιο «κοινός» τρόπος αλληλεπίδρασης με μία λειτουργία (operation) μίας Ηλεκτρονικής Υπηρεσίας, χωρίς όμως αυτό να σημαίνει ότι είναι και ο μόνος. Τέσσερις τρόποι αλληλεπίδρασης με μία λειτουργία μίας Ηλεκτρονικής Υπηρεσίας μπορούν να αναγνωρισθούν από τους οποίους οι δύο απαιτούν την αποστολή ενός μόνο μηνύματος ενώ οι άλλοι δύο απαιτούν την ανταλλαγή μηνυμάτων.

1. **Μονόδρομη** (One-way) Η λειτουργία δέχεται ένα μήνυμα χωρίς να χρειάζεται να επιστρέψει μία απάντηση.
2. **Αίτηση-Απάντηση** (Request-response) Η λειτουργία *δέχεται* ένα μήνυμα και *επιστρέφει* μία απάντηση
3. **Αποστολή Αίτησης-Απάντηση** (Solicit-response) Η λειτουργία *στέλνει* ένα μήνυμα και περιμένει να *δεχθεί* μία απάντηση.
4. **Ειδοποίηση** (Notification) Η λειτουργία στέλνει ένα μήνυμα χωρίς να περιμένει για απάντηση.

Όλες οι γλώσσες που χρησιμοποιούνται για την περιγραφή και τη σύνθεση πολύπλοκων Ηλεκτρονικών Υπηρεσιών βασίζονται σε αυτούς τους τρόπους αλληλεπίδρασης για την περιγραφή της αλληλεπίδρασης των «διαδικασιών» με τις Ηλεκτρονικές Υπηρεσίες, αλλά και για τον συντονισμό της εκτέλεσης διαδικασιών τους με την ανταλλαγή μηνυμάτων.

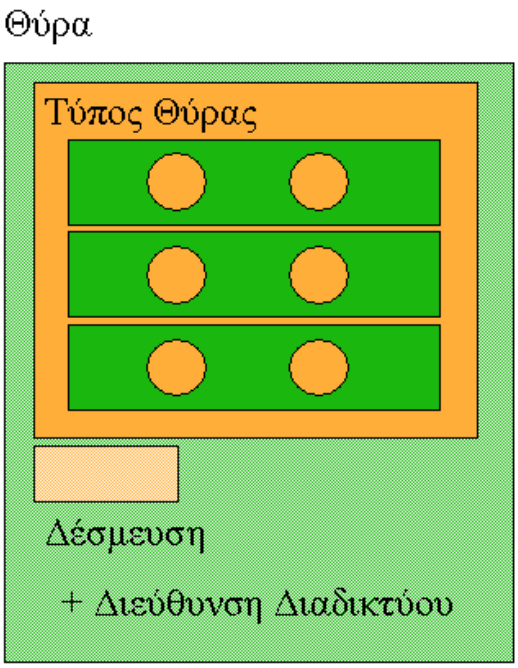
- **Δέσμευση.** (Binding) Το στοιχείο αυτό προσδιορίζει ένα συγκεκριμένο πρωτόκολλο και ορισμό τύπου δεδομένων για ένα συγκεκριμένο τύπο θύρας. Μπορούν να οριστούν παραπάνω από ένα διαφορετικά στοιχεία δέσμευσης για τον ίδιο τύπο θύρας. Στην **Εικόνα 2.11** βλέπουμε ένα παράδειγμα δέσμευσης διαγραμματικά και σαν μέρος εγγράφου WSDL.




```
<binding type="StockPortofolioPortType" name="bind1">
   <soap:binding style="document"
transport=http://schemas.xmlsoap.org/soap/http/>
   <operation name="…">
      <soap:operation soapAction="…"/>
      <input>
         <soap:body use="…" namespace="…" encodingStyle="…" />
      </input>
      <output>
         <soap:body use="…" namespace="…" encodingStyle="…" />
      </output>
   </operation>
</binding>
```

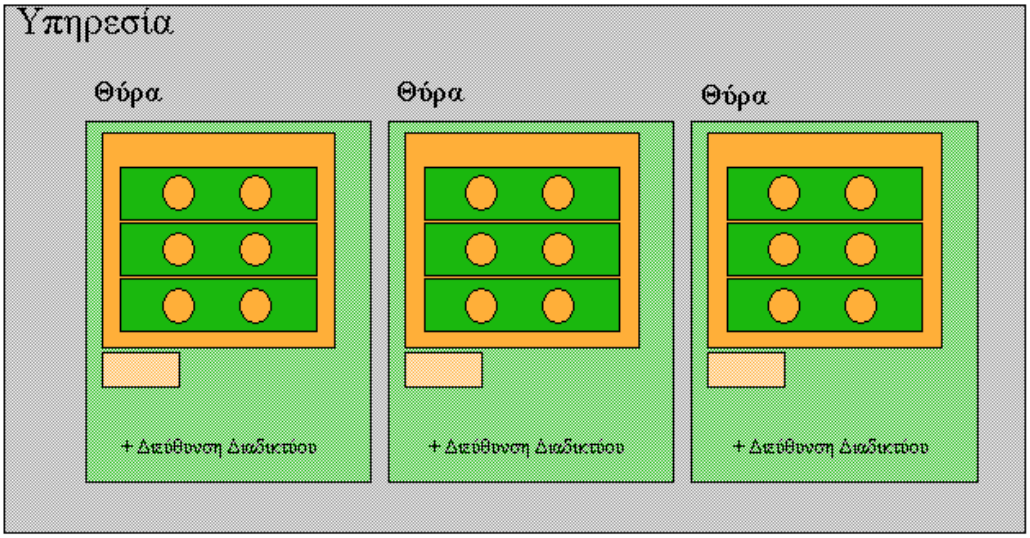
**Εικόνα 2.11: Ορισμός δεσμεύσεων στην WSDL**

- **Θύρα.** (Port) Ο συνδυασμός μίας *δέσμευσης* με μία διεύθυνση του Διαδικτύου( ip address) ορίζει μία *θύρα* (**Εικόνα 2.12**).




**Εικόνα 2.12: Ορισμός μίας θύρας στην WSDL**

- **Υπηρεσία.** Μια υπηρεσία είναι μία συλλογή από μία ή περισσότερες θύρες (**Εικόνα 2.13**).




**Εικόνα 2.13: Ορισμός μίας υπηρεσίας σε WSDL**

Υπάρχει και ένας εναλλακτικός τρόπος δόμησης του εγγράφου WSDL κατά τον οποίο το έγγραφο χωρίζεται ρητά σε δύο μέρη. Το ένα μέρος ονομάζεται «ορισμός διεπαφής υπηρεσίας» στον οποίο ορίζονται τα επαναχρησιμοποιήσιμα μέρη της υπηρεσίας, και το δεύτερο ονομάζεται «ορισμός υλοποίησης υπηρεσίας» στο οποίο ορίζονται τα πιό εξειδικευμένα στοιχεία υλοποίησης.

### 2.2.3.2 SOAP

Το πρωτόκολλο SOAP [WSDL] είναι το πρότυπο για την αποστολή μηνυμάτων και την απομακρυσμένη κλήση διαδικασιών (κλήση RPC) στο Διαδίκτυο. Παραδοσιακά η επικοινωνία μεταξύ των εφαρμογών γινόταν με κλήσεις RPC μεταξύ αντικειμένων όπως το DCOM ή η CORBA [CORBA]. Όμως το HTTP δεν έχει σχεδιαστεί για αυτό το σκοπό. Οι κλήσεις RPC παρουσιάζουν προβλήματα συμβατότητας και ασφάλειας με αποτέλεσμα να «μπλοκάρονται» από προγράμματα όπως τα firewalls ή οι proxy servers. Ο πλέον ενδεδειγμένος τρόπος επικοινωνίας είναι με ένα πρωτόκολλο «πάνω» από το HTTP μιάς και το HTTP υποστηρίζεται από όλους τους εξυπηρετητές και τούς φυλλομετρητές στο Διαδίκτυο. Για αυτό το σκοπό δημιουργήθηκε το SOAP.

Το SOAP είναι ανεξάρτητο από πλατφόρμες, λειτουργικά συστήματα και γλώσσες. Χρησιμοποιεί το HTTP σαν πρωτόκολλο μεταφοράς χωρίς αυτό να σημαίνει ότι και άλλα πρωτόκολλα όπως το FTP ή το SMTP δεν μπορούν να χρησιμοποιηθούν. Για την κωδικοποίηση των δεδομένων χρησιμοποιεί την XML.

Το SOAP ορίζει δύο τύπους μηνυμάτων, την *Αίτηση* (Request) και την *Απάντηση* (Response). Με αυτό τον τρόπο επιτρέπει στους *αιτούντες υπηρεσιών* να αιτούνται απομακρυσμένων διαδικασιών και στους *παροχείς υπηρεσιών* να απαντούν σε τέτοιες αιτήσεις. Ένα μήνυμα SOAP αποτελείται από δύο μέρη, την επικεφαλίδα (header) και το XML περιεχόμενο (XML payload). Η επικεφαλίδα εξαρτάται κυρίως από το πρωτόκολλο μεταφοράς που χρησιμοποιείται ενώ το περιεχόμενο του μηνύματος είναι ανεξάρτητο. Το XML μέρος μίας Αίτησης SOAP αποτελείται από τρία στοιχεία:

- Ο *Φάκελος* (Envelope) είναι το «ριζικό» στοιχείο (root element) ενός μηνύματος SOAP. Αυτό το στοιχείο καθορίζει ότι το XML έγγραφο αυτό, είναι ένα μήνυμα SOAP. Στον *Φάκελο* ορίζονται οι διάφοροι χώροι ονοματοδοσίας (namespaces) που χρησιμοποιούνται στο υπόλοιπο μήνυμα.
- Η *Επικεφαλίδα* (Header) είναι ένα προαιρετικό στοιχείο το οποίο περιλαμβάνει πληροφορία εξειδικευμένη για κάθε εφαρμογή (π.χ σχετική με πιστοποίηση, τρόπο πληρωμής κ.τ.λ). Για το λόγο αυτό συγκεκριμένες εφαρμογές μπορούν να την αγνοήσουν ή να επεξεργαστούν μόνο ένα μέρος της *Επικεφαλίδας*. Εάν υπάρχει, η *Επικεφαλίδα* πρέπει να είναι το πρώτο στοιχείο «παιδί» (child element) του *Φακέλου*.
- Ο *Κορμός* (Body) περιλαμβάνει το περιεχόμενο του μηνύματος. Εάν το SOAP χρησιμοποιείται για να πραγματοποιήσει μία κλήση RPC, ο *Κορμός* περιλαμβάνει ένα μόνο στοιχείο το οποίο περιέχει το όνομα και τα ορίσματα της διαδικασίας που θα κληθεί και την διεύθυνση της Ηλεκτρονικής Υπηρεσίας στην οποία βρίσκεται η διαδικασία. Αν υπάρχει *Επικεφαλίδα*, ο *Κορμός* πρέπει να είναι το ακριβώς επόμενο

στοιχείο της (sibling), ενώ αν δεν υπάρχει πρέπει να είναι το πρώτο στοιχείο «παιδί» του φακέλου.

Η *Απάντηση* SOAP επιστρέφεται σαν ένα XML έγγραφο μέσα σε μία HTTP απάντηση. Το XML κομμάτι είναι ακριβώς όπως της Αίτησης με την διαφορά ότι τώρα ο *Κορμός* περιλαμβάνει τα αποτελέσματα της κλήσης της διαδικασίας. Ένα παράδειγμα SOAP Αίτησης/ Απάντησης σχετικά με την τιμή ενός υπολογιστή φαίνεται παρακάτω.

Αίτηση SOAP:

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Απάντηση SOAP :

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

### 2.2.3.3 UDDI

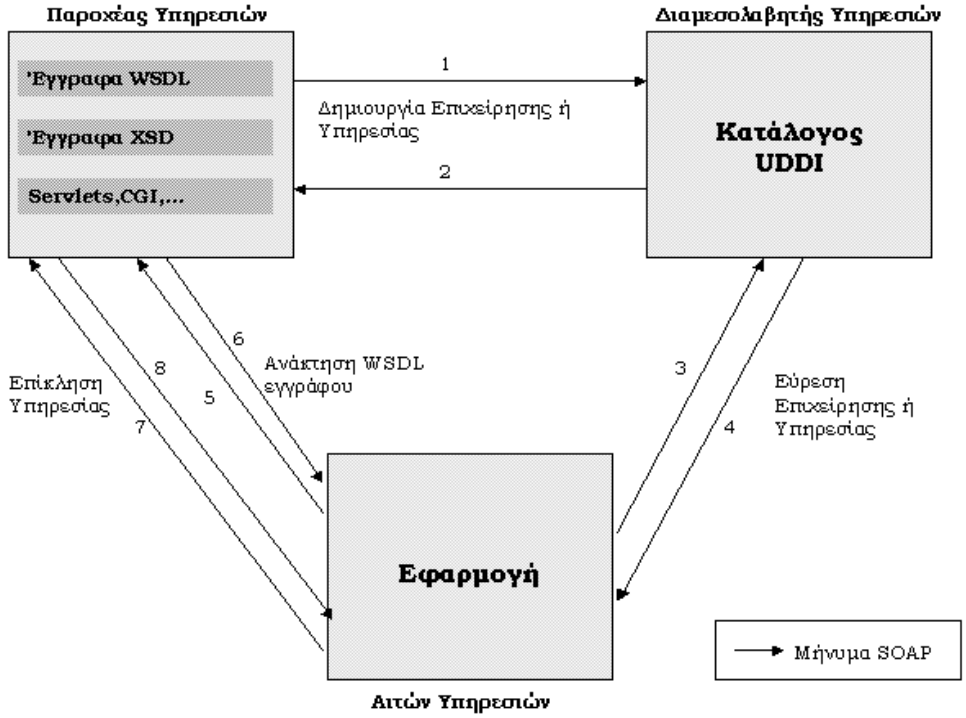
Το UDDI μπορεί να θεωρηθεί σαν μία μορφή «χρυσού οδηγού» που περιλαμβάνει όλες τις επιχειρήσεις που «εκθέτουν» Ηλεκτρονικές Υπηρεσίες. Το UDDI παρέχει μία Βάση Δεδομένων στην οποία γίνεται αναζήτηση βάσει του τύπου της επιχείρησης. Μπορεί να χρησιμοποιηθεί για να ελεγχθεί αν ένας επιχειρησιακός συνεργάτης προσφέρει μία συγκεκριμένη Ηλεκτρονική Υπηρεσία, να βρει τις επιχειρήσεις αυτές που παρέχουν ένα συγκεκριμένο τύπο υπηρεσίας και να ανακτήσει πληροφορία για τον τρόπο με τον οποίο μία επιχείρηση έχει «εκθέσει» μια Ηλεκτρονική Υπηρεσία, δηλαδή για τους τρόπους αλληλεπίδρασης με την Ηλεκτρονική Υπηρεσία. Η περιγραφή του UDDI περιλαμβάνει αφενός την προγραμματιστική διεπαφή (API) για την επικοινωνία μιας εφαρμογής (είτε πρόκειται για αιτών είτε για παροχέα μίας υπηρεσίας) με καταλόγους UDDI, αφετέρου ένα Σχήμα XML για μηνύματα SOAP που περιέχουν πληροφορία σχετική με τις επιχειρήσεις και υπηρεσίες που είναι καταχωρημένες στους UDDI καταλόγους.

Το Σχήμα XML του UDDI περιλαμβάνει τέσσερις βασικές δομές, τις *επιχειρησιακές οντότητες (*business entities), τις *επιχειρησιακές υπηρεσίες* (business services), τα *πρότυπα δέσμευσης* (binding templates) και τα tModels. Οι *επιχειρησιακές οντότητες*, περιλαμβάνουν πληροφορίες για επιχειρήσεις, όπως το όνομα τους, περιγραφή της επιχείρησης και των υπηρεσιών που προσφέρονται και πληροφορίες επικοινωνίας με την επιχείρηση. Οι *επιχειρησιακές υπηρεσίες* παρέχουν αναλυτικές πληροφορίες για τις υπηρεσίες που «εκθέτει» η κάθε επιχείρηση. Κάθε υπηρεσία μπορεί να προσφέρει παραπάνω από μία δυνατότητα δέσμευσης όπως είδαμε στην περιγραφή της WSDL. Πληροφορίες σχετικά με τους τρόπους δέσμευσης μίας Ηλεκτρονικής Υπηρεσίας δίνουν τα *πρότυπα δέσμευσης*. Τα tModels περιγράφουν ποια συγκεκριμένα πρότυπα χρησιμοποιεί μία υπηρεσία. Με αυτό τον τρόπο εφαρμογές μπορούν να ανακαλύψουν υπηρεσίες που είναι συμβατές με τα δικά τους εσωτερικά συστήματα.

Οι προγραμματιστική διεπαφή (API) του UDDI, περιλαμβάνει τον ορισμό των μηνυμάτων με τα οποία εφαρμογές μπορούν να αλληλεπιδράσουν με καταλόγους UDDI. Οι διεπαφές μπορούν να είναι είτε *αναζήτησης* είτε *έκδοσης*. Οι διεπαφές αναζήτησης χρησιμοποιούνται από αιτούντες για να εντοπίσουν πληροφορίες επιχειρήσεων, υπηρεσιών, προτύπων δέσμευσης ή tModels. Οι διεπαφές έκδοσης χρησιμοποιούνται από παροχείς για να δημιουργούν ή να αφαιρούν εγγραφές (υπηρεσιών) στους καταλόγους UDDI. Κατά αυτό τον τρόπο μπορούμε να πούμε ότι και οι κατάλογοι UDDI είναι Ηλεκτρονικές Υπηρεσίες με μία δεδομένη διεπαφή με

τους οποίους μπορεί οποιαδήποτε εφαρμογή να αλληλεπιδράσει χρησιμοποιώντας πάλι το πρωτόκολλο SOAP.

Έχοντας δει τα βασικά πρότυπα περιγραφής, διαφήμισης και επικοινωνίας μεταξύ Ηλεκτρονικών υπηρεσιών, μπορούμε να δούμε στην **Εικόνα 2.14** τις αλληλεπιδράσεις μεταξύ των διάφορων οντοτήτων σε ένα περιβάλλον Ηλεκτρονικών Υπηρεσιών που περιγράψαμε στην **Εικόνα 2.8**.




**Εικόνα 2.14: Σχέση των SOAP, WSDL και UDDI σε περιβάλλον Ηλεκτρονικών Υπηρεσιών.**

Αναφέραμε νωρίτερα ότι εκτός από τις βασικές λειτουργίες του μοντέλου Ηλεκτρονικών Υπηρεσιών, δηλαδή την δημιουργία, την περιγραφή, την έκδοση, την ανακάλυψη και την κλήση μίας Ηλεκτρονικής Υπηρεσίας υπάρχουν και κάποιες άλλες οι οποίες επιτρέπουν την σύνθεση πολύπλοκων Ηλεκτρονικών Υπηρεσιών χρησιμοποιώντας απλούστερες, την παρακολούθηση της εκτέλεσης Ηλεκτρονικών Υπηρεσιών κ.τ.λ. Στην επόμενη ενότητα θα περιγράψουμε δύο πρότυπες γλώσσες σύνθεσης Ηλεκτρονικών Υπηρεσιών την BPEL4WS και την BPML.

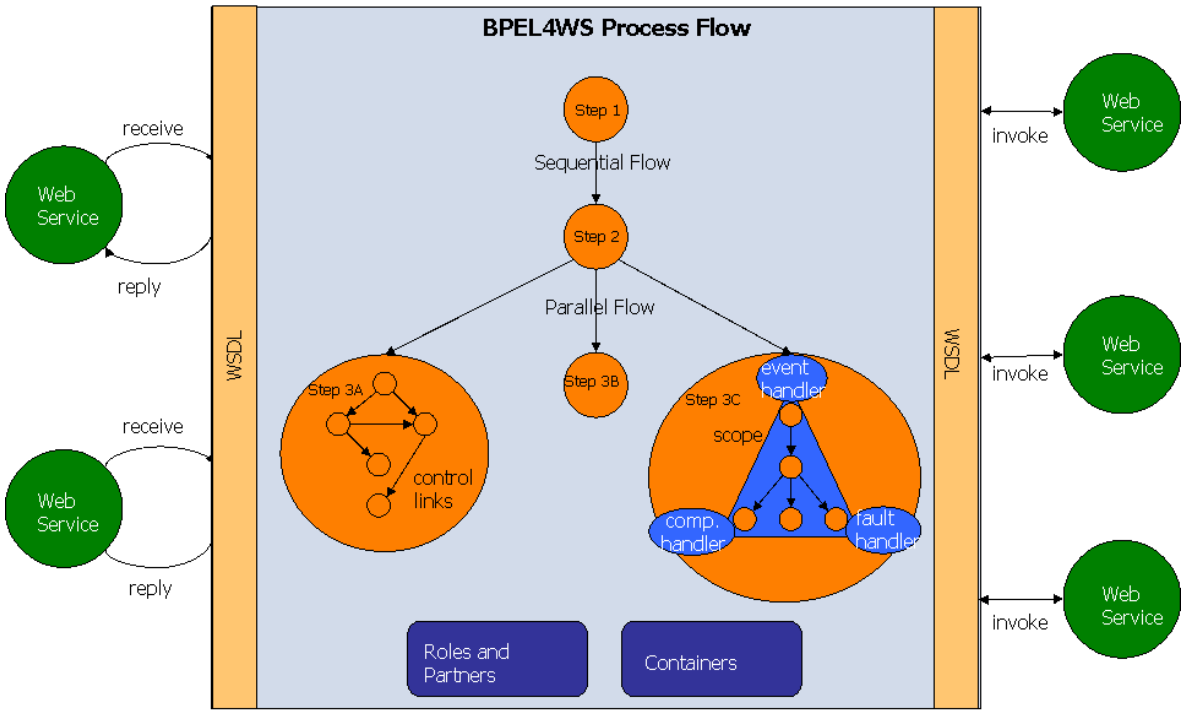
### 2.2.4 Σύντομη επισκόπηση των BPEL4WS και BPML

Υπάρχουν δύο βασικές προσεγγίσεις όσων αφορά την αναπαράσταση της ροής του ελέγχου στα πλαίσια μίας διαδικασίας, η αναπαράσταση δομημένη με «μπλόκ» (block structured) και η αναπαράσταση δομημένη με κατευθυνόμενους γράφους (graph structured). Η χρήση των «μπλόκ» είναι αποδοτική στις γλώσσες προγραμματισμού ενώ κατά την σχεδίαση επιχειρησιακών λειτουργιών (business operations) οι χρήστες προτιμούν διάφορα διαγράμματα ροής (flow diagrams) και γενικά διαγραμματικές τεχνικές. Η «μετάφραση» της ροής ελέγχου μίας διαδικασίας από «block-structured» σε κατευθυνόμενο γράφο δεν παρουσιάζει ιδιαίτερη δυσκολία. Δεν ισχύει όμως και το αντίστροφο, όπου απαιτούνται ένα σύνολο από περιορισμοί στην «δομή» του κατευθυνόμενου γράφου ούτως ώστε να μπορεί να «μεταφραστεί» σε block-structured αναπαράσταση.

#### 2.2.4.1 BPEL4WS

Η BPEL4WS χαρακτηρίζεται ως «block-structured» γλώσσα. Η δημιουργία των «μπλόκ» υποστηρίζεται με ένα σύνολο από δομημένα δομικά στοιχεία (structured constructs). Ο αναδρομικός ορισμός των blocks επιτρέπεται , αλλά όλοι οι ορισμοί και οι δηλώσεις περιορίζονται σε μία διαδικασία άνω επιπέδου (top-level process) δηλαδή δεν επιτρέπεται ο εμφωλευμένος ορισμός διαδικασιών (υποδιαδικασιών). Στα πλαίσια ενός «μπλόκ» υποστηρίζεται η δημιουργία ροών ενεργειών με την μορφή κατευθυνόμενων γράφων (graph-structured), εντούτοις σε περιορισμένο εύρος που απαγορεύει την δημιουργία κυκλικών γράφων και επιβάλει περιορισμούς στην διάσχιση των «ορίων» των «μπλόκ» από τους γράφους. Η **Εικόνα 2.15** παρέχει μία διαισθητική αναπαράσταση του μοντέλου διαδικασίας στην BPEL4WS.

Με τον συνδυασμό των παραπάνω χαρακτηριστικών η BPEL4WS επιχειρεί να προσφέρει ένα υβριδικό μοντέλο που θα συνδυάζει στοιχεία και από τις δύο τεχνικές (πρακτικά συνδυάζοντας στοιχεία από τους πρόγονους της, XLANG[T01] και WSFL[L01]). Έτσι προσφέρει το δομικό στοιχείο «flow» στα «πλαίσια» του οποίου μπορούν να οριστούν «σύνδεσμοι» (links) οι οποίοι περιγράφουν «αυθαίρετες» (arbitrary) εξαρτήσεις ροής ανάμεσα στις ενέργειες που ορίζονται στα «πλαίσια» του «flow». Παρόλαυτά υπάρχουν περιορισμοί που αποκλείουν την δημιουργία βρόγχων (loops) και την διάσχιση δομικών συνόρων (structural boundaries) με την χρήση «συνδέσμων» (links).




**Εικόνα 2.15 Το μοντέλο διαδικασίας της BPEL4WS**

Η BPEL4WS θεωρείται πως «χτίζει» πάνω από την WSDL για δύο λόγους. Πρώτον γιατί χρησιμοποιεί την WSDL για να καλεί άλλες Ηλεκτρονικές Υπηρεσίες αλλά και για να «προσφέρει» την διαδικασία της σαν Ηλεκτρονική Υπηρεσία. Δεύτερον, γιατί χρησιμοποιεί τους τύπους δεδομένων της WSDL για την αναπαράσταση της πληροφορίας που ανταλλάσσεται μεταξύ των ενεργειών στα πλαίσια μίας διαδικασίας. Η πληροφορία αυτή αποθηκεύεται σε στοιχεία που ονομάζονται «containers». Οι «containers» έχουν καθολική (global) εμβέλεια, και δεν επιτρέπεται ο ορισμός ενός «container» στα πλαίσια ενός μόνο «μπλόκ»

Στην BPEL4WS δεν υποστηρίζεται *ρητά* (explicitly) η δημιουργία ενός στιγμιοτύπου μίας διαδικασίας. Απεναντίας στιγμιότυπα διαδικασιών δημιουργούνται μόνο με την λήψη μηνυμάτων. Σε αυτή την περίπτωση το χαρακτηριστικό (attribute) «createInstance» του στοιχείου «receive» (που περιμένει έως την λήψη ενός μηνύματος) πρέπει να έχει την τιμή «yes». Η συσχέτιση εισερχόμενων μηνυμάτων (message correlation) με συγκεκριμένα στιγμιότυπα διαδικασιών δεν γίνεται με την επισύναψη στο μήνυμα κάποιου αναγνωριστικού στιγμιοτύπου (instance ID). Το «κατάλληλο» στιγμιότυπο διαδικασίας επιλέγεται βάσει ενός ή περισσοτέρων συλλογών

από πεδία δεδομένων «κλειδιά» (key data fields) μέσα στα ανταλλασσόμενα μηνύματα. Οι συλλογές αυτές από πεδία δεδομένων ονομάζονται «συλλογές συσχέτισης» (correlation sets).

Κατά την διάρκεια της εκτέλεσης μίας διαδικασίας είναι πιθανό να συμβούν «σφάλματα» (faults) τόσο στις καλούμενες Ηλεκτρονικές Υπηρεσίες όσο και στην ίδια την διαδικασία. Επιπρόσθετα, κάποιες «ενέργειες» (activities) που έχουν ήδη ολοκληρώσει την εκτέλεση τους πιθανώς να χρειαστεί να «αναιρεθούν» αν εκτελέστηκαν στα πλαίσια μίας γενικότερης συναλλαγής (transaction) η οποία για κάποιο λόγο χρειάζεται να «αναιρεθεί» (abort). Για τον χειρισμό τέτοιων περιπτώσεων η BPEL4WS υποστηρίζει τον ορισμό χειριστών σφαλμάτων (fault handlers) και χειριστών αντιστάθμισης (compensation handlers). Τέτοιου είδους χειρισμοί συνήθως αφορούν κάποια σύνολα από ενέργειες οι οποίες σχετίζονται μεταξύ τους. Η BPEL4WS επιτρέπει την ομαδοποίηση συνόλων ενεργειών για τις οποίες ορίζονται χειριστές σφαλμάτων και/ ή αντιστάθμισης με την χρήση του δομικού στοιχείου «scope». Έτσι το «scope» μπορεί να θεωρηθεί ότι ορίζει στο «πλαίσιο» του ένα «αντισταθμιζόμενο» (compensatable) ή «αναιρούμενο» (recoverable) κομμάτι δουλειάς.Να σημειωθεί εδώ ότι η «αντιστάθμιση» στην BPEL4WS εξαρτάται από την εφαρμογή. Αυτό σημαίνει ότι πρακτικά δεν «αναιρείται» μία ενέργεια που έχει ήδη γίνει αλλά εκτελείται μία ενέργεια την οποία ο «προγραμματιστής» έχει ορίσει ως ενέργεια αντιστάθμισης για το συγκεκριμένο «scope». Η συνολική διαδικασία ορίζει το καθολικό «scope». Περιέχει μία βασική ενέργεια (process) και στα πλαίσια της μπορούν να οριστούν «συνολικοί» χειριστές λαθών και αντιστάθμισης. Ο ορισμός των «containers» που είδαμε νωρίτερα δεν μπορεί να γίνει στα πλαίσια (και με εμβέλεια) ενός «scope». Αντίθετα όλοι οι «containers» έχουν καθολική εμβέλεια. Οι ενέργειες σε μία διαδικασία BPEL4WS μπορούν να είναι είτε «δομημένες» (structured) είτε «αρχέγονες» (primitive).

Το σύνολο των *αρχέγονων ενεργειών* περιλαμβάνει τις παρακάτω ενέργειες:

**invoke** Επικαλείται (invokes) μια λειτουργία σε μία Ηλεκτρονική Υπηρεσία

**receive** Περιμένει για ένα μήνυμα από μία εξωτερική πηγή

**reply** Απαντάει στην εξωτερική πηγή

**wait** Περιμένει για ένα χρονικό διάστημα

**assign** Αντιγράφει δεδομένα από ένα container σε ένα άλλο

**throw** Περιγράφει σφάλματα που μπορεί να συμβούν κατά την εκτέλεση

**terminate** Τερματίζει ολόκληρο το στιγμιότυπο της διαδικασίας

**empty** Είναι η κενή ενέργεια

Για την δημιουργία πολύπλοκων δομών υπάρχουν οι παρακάτω *δομημένες ενέργειες*:

**sequence** Ορίζει την σειρά εκτέλεσης

**switch** Για δρομολόγηση υπό συνθήκη

**while** Για την δημιουργία βρόγχων

**pick** Για συνθήκες συγχρονισμού που βασίζονται σε εξωτερικά γεγονότα ή διορίες

**flow** Για παράλληλη δρομολόγηση

**scope** Για την *ομαδοποίηση* ενεργειών που θα τις χειριστεί ο ίδιος χειριστής λαθών

Οι δομημένες ενέργειες μπορούν να εμφωλευτούν και να συνδυαστούν με αυθαίρετους τρόπους. Μεταξύ ενεργειών που εκτελούνται παράλληλα (με την χρήση του δομικού στοιχείου «flow») η σειρά εκτέλεσης μπορεί να καθοριστεί περαιτέρω με την χρήση συνδέσμων ελέγχου (control links) επιτρέποντας την δημιουργία των κατευθυνόμενων γράφων που περιγράψαμε νωρίτερα.

### 2.2.4.2 BPML

Η BPML περιγράφει διαδικασίες σαν ανταλλαγές μηνυμάτων XML ανάμεσα σε συμμετέχοντες (participants). Οι συμμετέχοντες μπορούν να αναπαριστούν κάποιους ρόλους και να εκπροσωπούν κάποιους οργανισμούς, εφαρμογές ή άλλες διαδικασίες. Οι συμμετέχοντες μπορούν να οριστούν είτε στατικά (κατά την σχεδίαση) είτε δυναμικά (κατά την διάρκεια της εκτέλεσης). Πρακτικά μία διαδικασία BPML περιγράφει την αλληλεπίδραση (interaction) και την συνεργασία (collaboration) ενός συνόλου συμμετεχόντων μερών..

Η BPML είναι μία «block-structured» γλώσσα. Η δυνατότητα αναδρομικού ορισμού των «blocks» παίζει σημαντικό ρόλο σε θέματα εμβέλειας (scoping issues) σχετικά με δηλώσεις, ορισμούς και εκτέλεση διαδικασιών. Η ροή του ελέγχου στην BPML περιγράφεται εξ'ολοκλήρου με δομικά στοιχεία (block strucuture concepts) ενώ η αναπαράσταση των «εσωτερικών δεδομένων» της διαδικασίας (workflow relevant data) γίνεται με την μορφή «ιδιοτήτων» (properties). Το μοντέλο διαδικασίας της BPML δεν παρουσιάζει καμία «διαισθητική» ιδιαιτερότητα σε σχέση με οποιαδήποτε άλλη «αυστηρά block structured» γλώσσα και για αυτό δεν θα παρουσιαστεί διαγραμματικά.

Μία διαδικασία στην BPML αποτελείται από απλές και πολύπλοκες ενέργειες. Οι απλές ενέργειες περιλαμβάνουν την αποστολή ή την λήψη ενός μηνύματος, την κλήση (invocation) μίας Ηλεκτρονικής Υπηρεσίας ή την δημιουργία μίας εξαίρεσης (exception raising). Οι πολύπλοκες ενέργειες περιλαμβάνουν «block-structured»

δομικά στοιχεία ροής ελέγχου για την ακολουθιακή, την παράλληλη ή την υπό συνθήκη εκτέλεση άλλων απλών ή πολύπλοκων ενεργειών. Πέραν αυτών των δομικών στοιχείων υποστηρίζεται η χρήση κάποιων «σινιάλων» για τον περαιτέρω συντονισμό της εκτέλεσης των ενεργειών. Οι ενέργειες μπορούν να προγραμματιστούν για εκτέλεση σε μία μελλοντική στιγμή και χρονικοί περιορισμοί μπορούν να προσαρτηθούν στην εκτέλεση μίας ενέργειας. Η ροή των δεδομένων επιτυγχάνεται με την ανάθεση δεδομένων από «εισερχόμενα» μηνύματα σε μεταβλητές περιβάλλοντος (state variables) που ονομάζονται «ιδιότητες» (properties) και αντίστροφα από «ιδιότητες» σε «εξερχόμενα» μηνύματα.

Η BPML χρησιμοποιεί ευρέως την δημιουργία «μπλόκ εμβέλειας» (block structure scooping) τα οποία επιτρέπουν την δημιουργία ορισμών και περιγραφών «ορατών» μόνο εντός του «μπλόκ» στο οποίο είναι ορισμένες. Κατά αυτό τον τρόπο ορίζονται πολύπλοκες ενέργειες (complex activities) οι οποίες αναφέρονται σε σύνολα ενεργειών (activity sets) ορισμένα εντός κάποιων συγκεκριμένων πλαισίων (contexts). Εντός ενός πλαισίου είναι δυνατός ο ορισμός ή επαναορισμός «ιδιοτήτων», ο ορισμός ή επαναορισμός διαδικασιών (εμφωλευμένες διαδικασίες ή υποδιαδικασίες) κ.λ.π. Απο την στιγμή που τα σύνολά ενεργειών (activity sets) μπορούν να περιέχουν πολύπλοκες ενέργειες, η εμφώλευση αυτή είναι *αναδρομική*.

Η BPML παρέχει υποστήριξη για «συναλλαγές» (transactions) τόσο τύπου ACID όσο και μεγάλης διάρκειας (long-running). Μία «συναλλαγή» μπορεί να συσχετιστεί με μία πολύπλοκη ενέργεια (εντός ενός πλαισίου (context)). Αυτό υπονοεί ότι και οι «συναλλαγές» μπορούν να είναι εμφωλευμένες. Ενέργειες αντιστάθμισης μπορούν να οριστούν και για τα δύο ήδη «συναλλαγών». Σε περίπτωση που μία «συναλλαγή» ακυρωθεί, η ενέργειες αντιστάθμισης εντός του ίδιου πλαισίου θα εκτελεστούν με αντίστροφη (reverse) σειρά.

Η BPML υποστηρίζει τον ορισμό χειριστών εξαιρέσεων (exception handlers) εντός ενός πλαισίου (context). Σε περίπτωση που η εξαίρεση δεν «πιαστεί» σε ένα πλαίσιο από ένα «χειριστή» (handler), διαδίδεται «προς τα πάνω» (propagate upwards) στο πλαίσιο που το περικλείει (enclosing context). Κατά αυτό τον τρόπο είτε θα «πιαστεί» η εξαίρεση σε κάποιο πλαίσιο είτε θα ακυρωθεί (abort) η συναλλαγή (transaction).

Έχοντας καλύψει τα βασικά χαρακτηριστικά του μοντέλου της BPML μπορούμε να συνοψίσουμε το σύνολο των (απλών και πολύπλοκων) ενεργειών της. Το σύνολο των *βασικών ενεργειών* της BPML περιλαμβάνει τις ενέργειες:

**action** Για την εκτέλεση ή επίκληση λειτουργιών που περιλαμβάνει ανταλλαγή μηνυμάτων. Οι ενέργειες action επιτρέπουν σε δύο υψηλότερου επιπέδου διαδικασίες να επικοινωνούν.

**assign** Αναθέτει μία τιμή σε μία «ιδιότητα»

**call** Αρχικοποιεί μία διαδικασία (είτε υψηλότερου επιπέδου είτε υποδιαδικασία) και την περιμένει να τελειώσει

**compensation** Προκαλεί αντιστάθμιση (compensation) για μία διαδικασία

**delay** Περιμένει για μία χρονική περίοδο

**empty** Είναι η κενή ενέργεια

**fault** Δημιουργεί ένα λάθος σε ένα πλαίσιο

**raise** Δημιουργεί ένα σινιάλο

**spawn** Αρχικοποιεί μία διαδικασία χωρίς να την περιμένει να τελειώσει

**synch** Περιμένει για ένα σινιάλο να δημιουργηθεί

Το σύνολο των *πολύπλοκων ενεργειών* της BPML είναι οι παρακάτω:

**all** Για την εκτέλεση ενεργειών παράλληλα

**choice** Για την επιλογή ανάμεσα σε εναλλακτικές επιλογές βάσει της άφιξης ενός γεγονότος

**foreach** Για την εκτέλεση ενεργειών μία φορά για κάθε στοιχείο μίας λίστας

**sequence** Για την εκτέλεση ενεργειών σειριακά

**switch** Για την εκτέλεση μίας από ένα σύνολο ενεργειών υπό συνθήκη

**until** Για την εκτέλεση ενεργειών μία ή περισσότερες φορές βάσει μίας συνθήκης εξόδου

**while** Για την εκτέλεση ενεργειών καμία ή περισσότερες φορές βάσει μίας συνθήκης εξόδου

### 2.2.5 Περιπτώσεις Ηλεκτρονικών Υπηρεσιών

Όπως αναφέραμε και νωρίτερα πέρα από τις βασικές ενέργειες που σχετίζονται με τις Ηλεκτρονικές Υπηρεσίες δηλαδή την δημιουργία, την διαφήμιση, την αναζήτηση και την επίκληση υπάρχουν ένα σύνολο ακόμα από ενέργειες οι οποίες σχετίζονται με την σύνθεση πολύπλοκων Ηλεκτρονικών Υπηρεσιών από άλλες απλούστερες, την παρακολούθηση της εκτέλεσης τέτοιων υπηρεσιών κ.τ.λ Μέχρι σήμερα, το μεγαλύτερο ποσοστό της έρευνας σχετικά με τις Ηλεκτρονικές Υπηρεσίες, είχε επικεντρωθεί στην σύνθεση *διακριτών* (discrete) υπηρεσιών με μικρή διάρκεια. Τέτοιου είδους υπηρεσίες είναι για παράδειγμα η προσθήκη ενός αντικειμένου σε ένα «καλάθι» αγορών, η χρέωση

μίας πιστωτικής κάρτας ή ο έλεγχος της διαθεσιμότητας εισιτηρίων για μία θεατρική παράσταση. Οι προγραμματιστικές διεπαφές (APIs) τέτοιων υπηρεσιών πιθανώς να περιλαμβάνουν πολλές μεθόδους οι οποίες όμως είναι *μη αναδραστικές,* δηλαδή δεν αποκρίνονται σε ασύγχρονα γεγονότα που προέρχονται από άλλες Ηλεκτρονικές Υπηρεσίες ή εφαρμογές. Από την άλλη υπάρχουν αρκετά είδη Ηλεκτρονικών Υπηρεσιών που υποστηρίζουν συνεδρίες (session-oriented) και χρειάζεται κατά την διάρκεια του κύκλου της ζωής τους να αποκρίνονται σε ασύγχρονα γεγονότα. Τέτοιες Ηλεκτρονικές Υπηρεσίες εμφανίζονται συχνά στις τηλεπικοινωνίες και στις δια-επιχειρησιακές ροές εργασίας.

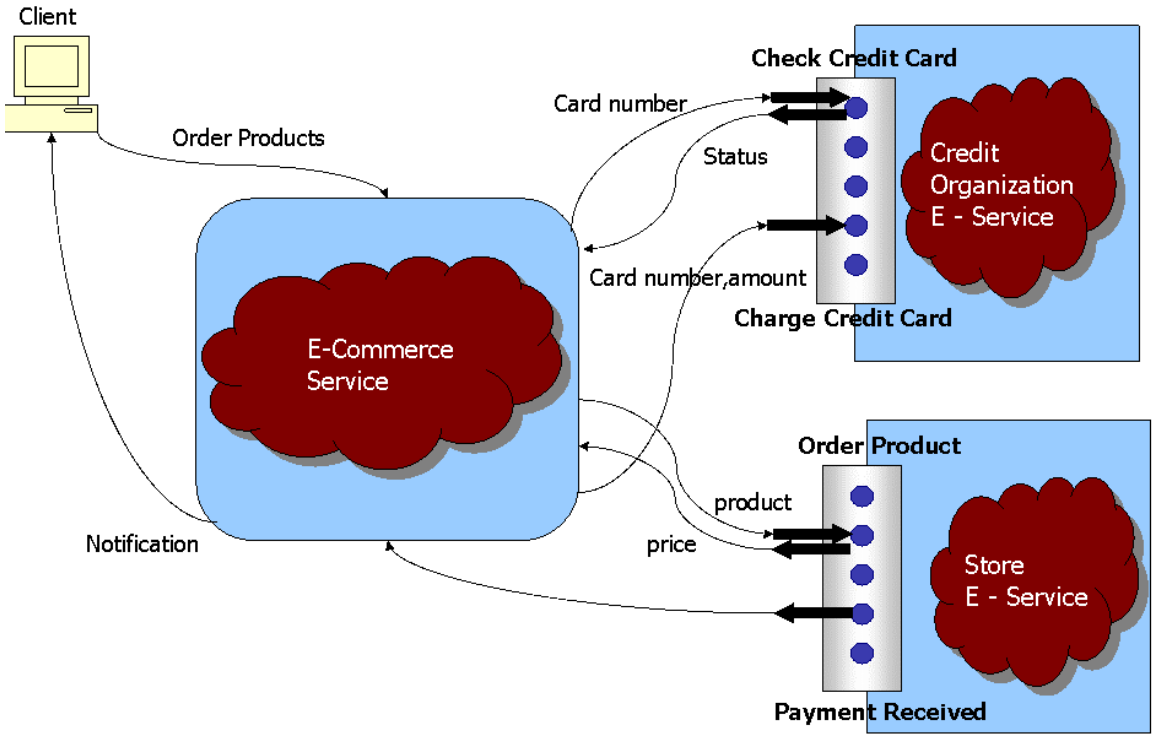
Τηλεπικοινωνιακές εφαρμογές όπως τηλεδιασκέψεις, συνεργατικές συνεδρίες με βίντεο ή αλληλεπιδραστικά παιχνίδια περιλαμβάνουν συνεδρίες και πρέπει να αποκρίνονται σε ασύγχρονα γεγονότα. Αν υποθέσουμε ότι όλα αυτά λειτουργούν χρεώνοντας ένα προπληρωμένο λογαριασμό τότε οι συνεδρίες πρέπει να τερματιστούν αν ο λογαριασμός αδειάσει. Αντίστοιχα αν ένας από τους συμμετέχοντες αποσυρθεί πιθανώς να είναι χρήσιμο οι υπόλοιποι να ενημερωθούν για αυτό το γεγονός. Μία άλλη περίπτωση είναι αυτή των υπηρεσιών «παρουσίας» οι οποίες δημιουργούν γεγονότα όταν κάποιος εμφανιστεί σε ένα δίκτυο. Σε αυτή την περίπτωση ίσως μία συνεδρία τηλεδιάσκεψης να πρέπει να αποκριθεί και να τον συμπεριλάβει ενημερώνοντας και τους υπόλοιπους συμμετέχοντες.

Γενικά μπορούμε να χρησιμοποιήσουμε τον όρο *αποκριτικές* (responsive) για τις Ηλεκτρονικές Υπηρεσίες που χρειάζεται να αποκρίνονται σε ασύγχρονα γεγονότα και *απομονωμένες* (insular) για αυτές που δεν συσχετίζονται με τέτοια γεγονότα.[CHK+01] Ένα χαρακτηριστικό παράδειγμα *απομονωμένης* υπηρεσίας είναι το κλασσικό σενάριο ηλεκτρονικού εμπορίου που χρησιμοποιούμε από την αρχή του κεφαλαίου. Αντίστοιχα ένα παράδειγμα αποκριτικής υπηρεσίας είναι αυτή που χειρίζεται μία ηλεκτρονική τηλεδιάσκεψη, το οποίο θα χρησιμοποιούμε στο εξής για να περιγράφουμε διαισθητικά τα χαρακτηριστικά των αποκριτικών Ηλεκτρονικών Υπηρεσιών.

### 2.2.5.1 Ηλεκτρονικό Εμπόριο

Ένα κλασσικό σενάριο εμπορίου περιγράφηκε αναλυτικά στην υποενότητα 2.1.3 σαν παράδειγμα Ροής Εργασίας. Το σημείο εισόδου της Ροής Εργασίας δηλαδή η πρώτη της ενέργεια είναι η υποδοχή μίας νέας παραγγελίας. Αν θεωρήσουμε ότι ένας παροχέας υπηρεσιών δέχεται από το Διαδίκτυο μηνύματα -σε μία συγκεκριμένη θύρα- που

αντιστοιχούν σε αιτήσεις νέων παραγγελιών τότε μπορούμε να μιλάμε πλέον για μία υπηρεσία Ηλεκτρονικού Εμπορίου. Μπορούμε να δούμε μία διαισθητική απεικόνιση αυτής της υπηρεσίας στην **Εικόνα 2.16**. Η Ηλεκτρονική Υπηρεσία έχει μία διεπαφή γνωστή σε όσους ενδιαφέρονται να κάνουν παραγγελίες, η οποία καθορίζει ότι όλες οι επικλήσεις προς αυτήν πρέπει να παρέχουν κάποια δεδομένα όπως το όνομα του προϊόντος που θέλουν να παραγγείλουν, τον τρόπο πληρωμής που επιθυμούν κ.λ.π Από την στιγμή που έχει αποσταλεί από μια εφαρμογή μία αίτηση για μία νέα παραγγελία, ο παροχέας υπηρεσίας εκτελεί αυτή την αίτηση χωρίς να επηρεάζεται από άλλα εξωτερικά γεγονότα. Όταν πλέον έχει ολοκληρώσει την διαδικασία εξυπηρέτησης της παραγγελίας και τα προϊόντα έχουν αποσταλεί, ενημερώνει τον πελάτη (τον αιτών της υπηρεσίας) με ένα e-mail ότι όλα πήγαν καλά. Η υπηρεσία Ηλεκτρονικού εμπορίου με την σειρά της χρησιμοποιεί κάποιες άλλες Ηλεκτρονικές Υπηρεσίες για να εκτελέσει κάποιες επιμέρους λειτουργίες. Χρησιμοποιεί μία υπηρεσία που ελέγχει την εγκυρότητα της πιστωτικής κάρτας, μία υπηρεσία που χρεώνει την πιστωτική κάρτα και μία υπηρεσία η οποία αναλαμβάνει την αποστολή του κάθε προϊόντος στον πελάτη. Όλες αυτές οι επιμέρους υπηρεσίες είναι «απομονωμένες», καλούνται με κάποια δεδομένα που χρειάζονται, επιτελούν ένα έργο και πιθανώς επιστρέφουν μετά από λίγο κάποια αποτελέσματα χωρίς να αποκρίνονται ή να επηρεάζονται από οποιαδήποτε εξωτερικά γεγονότα. Εν κατακλείδι μπορεί κάποιος να υπολογίσει προσεγγιστικά πόσο χρόνο θα διαρκέσει η εξυπηρέτηση μίας παραγγελίας αφού δεν την επηρεάζει κανένας εξωτερικός παράγοντας.

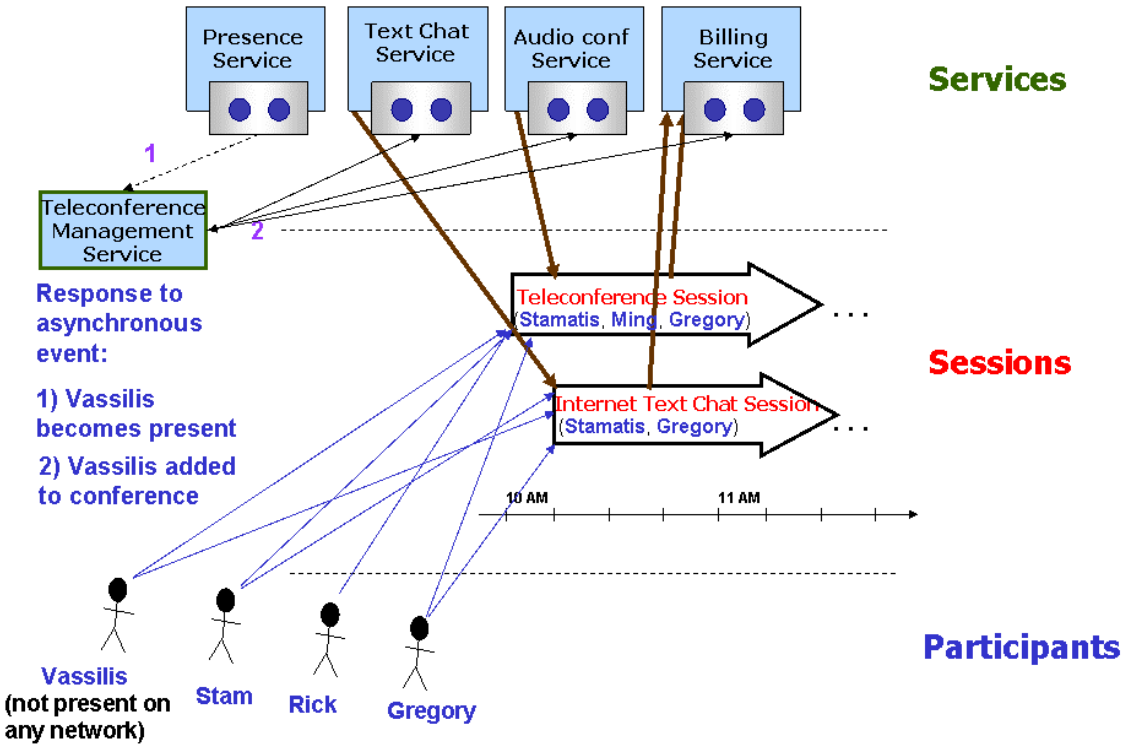



**Εικόνα 2.16 Παράδειγμα Ηλεκτρονικού Εμπορίου**

### 2.2.5.2 Ηλεκτρονική Τηλεδιάσκεψη

Οι τεχνολογικές καινοτομίες στα τηλεφωνικά δίκτυα επιτρέπουν όλο και περισσότερο την δημιουργία τηλεδιασκέψεων και άλλων συνεργατικών τηλεπικοινωνιακών μορφών δυναμικά προσαρμοζόμενων στις εκάστοτε συνθήκες. Ένα τυπικό σενάριο όπως αυτο της **Εικόνας 2.17** είναι ότι ένας χρήστης κάνει αίτηση για μία ηλεκτρονική τηλεδιάσκεψη από ένα δεδομένο αρχικό χρονικό σημείο έως ένα δεδομένο τελικό χρονικό σημείο με ένα δεδομένο αριθμό συμμετεχόντων και πιθανώς παραπάνω από μία μορφές επικοινωνίας (ηχητική τηλεδιάσκεψη, συνεδρία με γραπτά μηνύματα (chat), συνεργατική πλοήγηση στο Διαδίκτυο κ.λ.π). Παράλληλα πιθανή είναι η ύπαρξη υπηρεσιών «παρουσίας» που εντοπίζουν την παρουσία ή την αποσύνδεση κάποιου συμμετέχοντα από το δίκτυο, και υπηρεσιών χρέωσης που ασχολούνται με την χρέωση των συνεδριών.

Ας υποθέσουμε ότι ένας χρήστης αιτείται μια ηλεκτρονική τηλεδιάσκεψη που περιλαμβάνει ηχητική και γραπτή επικοινωνία με πέντε συμμετέχοντες από τις 10 π.μ έως τις 12 μ.μ. Η υπηρεσία «παρουσίας» χρησιμοποιείται τόσο για να αναγνωρίζει όταν ένας από τους συμμετέχοντες χάνεται από το δίκτυο, όσο και για να εντοπίζει ένα συμμετέχοντα που εμφανίζεται στο δίκτυο και να τον προσθέτει αυτόματα στην




**Εικόνα 2.17: Τυπικό σενάριο Ηλεκτρονικής Τηλεδιάσκεψης**

τηλεδιάσκεψη. Επιπρόσθετα, η τηλεδιάσκεψη χρεώνεται στον λογαριασμό του χρήστη που την ξεκίνησε, έχοντας ένα ποσό Α στον λογαριασμό του. Επιθυμία μας είναι να ελαχιστοποιήσουμε την χρέωση, χρεώνοντας τον χρήστη για τούς συμμετέχοντες που όντως συμμετέχουν στην τηλεδιάσκεψη και όχι για αυτούς που απλά έχουν δηλωθεί εξ'αρχής. Κατά την εκτέλεση αυτού του παραδείγματος τέσσερις συνεδρίες βρίσκονται σε εξέλιξη καθεμία από τις οποίες να μπορεί πιθανώς να επηρεάσει τις υπόλοιπες. Αν για παράδειγμα ένας χρήστης χάσει την ηχητική επικοινωνία ίσως οι υπόλοιποι να πρέπει να ενημερωθούν για αυτό το γεγονός με γραπτό μήνυμα. Ο εντοπισμός κάποιου από την υπηρεσία παρουσίας ίσως πρέπει να προκαλέσει την αυτόματη σύνδεση του σε κάποιες από τις συνεδρίες και παράλληλα να επηρεάσει την συνεδρία της χρέωσης. Η υπηρεσία της χρέωσης πρέπει να ενημερώνεται περιοδικά από τις διάφορες υπηρεσίες για την κατάσταση στην οποία βρίσκονται για να επηρεάζει την χρέωση αντίστοιχα, ενώ αν συμβεί το γεγονός του αδειάσματος του λογαριασμού πιθανώς να πρέπει να τερματίσει όλες τις συνεδρίες αφού πρώτα ενημερώσει τους συμμετέχοντες. Είναι προφανές ότι αντίθετα με το παράδειγμα του ηλεκτρονικού εμπορίου, το παράδειγμα της ηλεκτρονικής τηλεδιάσκεψης απαιτεί απόκριση σε πολλών ειδών γεγονότα και αλληλεπίδραση μεταξύ τρεχόντων συνεδριών γι' αυτό και αποτελεί μια αποκριτική Ηλεκτρονική Υπηρεσία.

# *Κεφάλαιο 3*

# *Το μοντέλο ενεργών διαγραμμάτων ροής και η γλώσσα XASC*

Η συνεισφορά της εργασίας μας περιλαμβάνει μεταξύ άλλων μια γλώσσα σύνθεσης Ενεργών Ηλεκτρονικών Υπηρεσιών με το όνομα *XASC* (XML-based Active Service Composition). Η γλώσσα αυτή που χρησιμοποιείται για τον ορισμό των ενεργών διαγραμμάτων ροής (active flowcharts) του μοντέλου μας είναι βασισμένη στην XML (XML-based) για δύο λόγους: η αναπαράσταση των ενεργών διαγραμμάτων ροής γίνεται σε XML και η επικοινωνία ανάμεσα στα διαγράμματα ροής και στις Ηλεκτρονικές Υπηρεσίες είναι βασισμένες σε μηνύματα SOAP XML.Πολλές γλώσσες έχουν προταθεί για την δημιουργία σύνθετων Ηλεκτρονικών Υπηρεσιών από απλούστερες. Αρκετοί συγγραφείς μάλιστα για να περιγράψουν την μεγάλη ποικιλία από γλώσσες με αμφιλεγόμενη συνεισφορά ονόμασαν αυτό το ρεύμα WSAH (Web Services Acronym Hell).

Κάποιες από αυτές τις γλώσσες έχουν καλά ορισμένο θεωρητικό υπόβαθρο βασισμένες σε τυπικούς φορμαλισμούς όπως τα Petri Nets και τα Statecharts ενώ αρκετές άλλες έχουν λιγότερο «σαφές» θεωρητικό υπόβαθρο, δημιουργημένες για να εξυπηρετήσουν τις ανάγκες κάποιου εμπορικού προϊόντος. Ακόμα και ανάμεσα σε αυτές που είναι θεωρητικά τεκμηριωμένες, υπάρχει συχνά διάσταση που οφείλεται στα ξεχωριστά χαρακτηριστικά της κάθε μιας και στις αρχές σχεδίασης της (design principles). Το πεδίο της σύνθεσης Ηλεκτρονικών Υπηρεσιών είναι πολύ μεγάλο και είναι πρακτικά αδύνατο για μία γλώσσα να αναπαραστήσει με βέλτιστο και συμπαγή (compact) τρόπο όλα τα πιθανά παραδείγματα αλληλεπίδρασης. Η συνήθης πρακτική είναι ότι κάθε γλώσσα εστιάζει στην βέλτιστη λύση μίας κατηγορίας προβλημάτων προσπαθώντας παράλληλα να παρέχει (έστω και με πολύπλοκες διαδικασίες) την εκφραστική δύναμη για να αντιμετωπιστούν τα περισσότερα προβλήματα. Για παράδειγμα κάποιες γλώσσες επιτρέπουν την δημιουργία πολλών στιγμιότυπων της ίδιας ενέργειας στα πλαίσια της ίδιας διαδικασίας (process) ενώ άλλες όχι. Κάποιες γλώσσες υποστηρίζουν μόνο δομημένες επαναλήψεις με ένα σημείο εισόδου και ένα εξόδου ενώ άλλες επιτρέπουν και πιο αυθαίρετες δομές. Κάποιες γλώσσες ορίζουν ένα σημείο το οποίο αποτελεί το σημείο τερματισμού της διαδικασίας ενώ άλλες επιτρέπουν και λιγότερο ρητούς ορισμούς του τερματισμού.

Αυτά και αρκετά ακόμα παρόμοια θέματα καθορίζουν την εκφραστική δύναμη της κάθε γλώσσας. Μελετώντας τις γλώσσες σύνθεσης Ηλεκτρονικών Υπηρεσιών σαν γλώσσες περιγραφής Ροών Εργασίας, δηλαδή από την σκοπιά της ροής του ελέγχου και των δομών (constructs) που την καθορίζουν θα ήταν πολύ χρήσιμο να προσδιοριστεί ένα σύνολο από «δομικές απαιτήσεις» τις οποίες πρέπει να πληροί μια γλώσσα ώστε να θεωρείται εκφραστικά πλήρης. Μια τέτοια σύνοψη από δομές ελέγχου, θα αποτελούσε τόσο σημείο αναφοράς για τις γλώσσες που είναι τώρα «υπό-κατασκευή» όσο και μέτρο σύγκρισης και εκτίμησης της εκφραστικής δύναμης υπαρχόντων γλωσσών. Μια τέτοια δουλειά παρουσιάζεται στο [ADH+02] και απεικονίζει τις απαιτήσεις που υπάρχουν ή μπορεί να υπάρξουν από μια γλώσσα ορισμού Ροών Εργασίας με τη μορφή *προτύπων* ροών εργασίας. Τα πρότυπα δεν είναι ορισμένα αυστηρά ή θεωρητικά αλλά επιτρέπουν μια ευέλικτη αναπαράσταση σε διάφορους φορμαλισμούς και τυπικές γλώσσες (Δίκτυα Petri, Χάρτες Καταστάσεων) των «απαιτήσεων» που αντιπροσωπεύουν. Η σημασιολογία και η ερμηνεία των *προτύπων* ροών εργασίας είναι ξεκάθαρη σε συγκεκριμένα πλαίσια Ροών Εργασίας με συγκεκριμένες παραδοχές για το περιβάλλον εκτέλεσης τους (παραδείγματος χάριν το πως γίνεται η ενεργοποίηση των μεταβάσεων κ.λ.π)

Η χρήση αυτών των προτύπων αυτά θα μας βοηθήσει να παρουσιάσουμε τα δομικά στοιχεία της γλώσσας μας μέσα από απαιτήσεις που καλείται να αντιμετωπίσει μια γλώσσα σύνθεσης Ηλεκτρονικών Υπηρεσιών (ή ορισμού ροών εργασίας). Προτού ξεκινήσουμε την παρουσίαση της XASC σε συνδυασμό με τα πρότυπα ροών εργασίας θα ήταν χρήσιμο να δούμε συνοπτικά τα χαρακτηριστικά του μοντέλου ενεργών διαγραμμάτων ροής του AZTEC και τις βασικές σχεδιαστικές αρχές του. Με αυτό τον τρόπο θα γίνει αφενός πιο κατανοητή η περιγραφή των διάφορων προτύπων με στοιχεία της XASC αφετέρου θα γίνει κατανοητό ότι ο λόγος που κάποια πρότυπα δεν υποστηρίζονται άμεσα είναι ότι είτε αυτό δεν μας είναι απαραίτητο, είτε ότι θα παρέβαινε τις βασικές αρχές του μοντέλου.

## *3.1 Βασικά Χαρακτηριστικά του Μοντέλου της XASC*

Σκοπός του AZTEC είναι να παρέχει ένα πλαίσιο (framework) για την σύνθεση (assembly), την εκτέλεση (execution και την παρακολούθηση (monitoring) τόσο απομονωμένων (insular) όσο και αποκριτικών (responsive) Ηλεκτρονικών Υπηρεσιών. Συνδυάζοντας στοιχεία τόσο από το παράδειγμα δομημένων ροών εργασίας (structured workflows) όσο και από το παράδειγμα γεγονότων (event paradigm) γίνεται εφικτή η

δημιουργία ενός χαλαρά συνδεδεμένου (loosely coupled) περιβάλλοντος ορισμού διαδικασιών. Πιο συγκεκριμένα, δομικά στοιχεία «τύπου ροής εργασίας» (workflow style constructs) μπορούν να χρησιμοποιηθούν για να καθοριστεί η «απόκριση» σε κάθε ένα τύπο γεγονότος, χωρίς αυτές οι «αποκρίσεις» να απαιτείται να είναι «ενσωματωμένες» στα πλαίσια μίας υψηλού επιπέδου διαδικασίας ροής εργασίας. Έτσι οι αποκρίσεις σε γεγονότα μπορούν να θεωρηθούν σαν ανεξάρτητα λογικά δομικά κομμάτια (logical building blocks) επιτρέποντας την αρθρωτή (modular) σύνθεση Ηλεκτρονικών Υπηρεσιών. Το μοντέλο ενεργών διαγραμμάτων ροής (*Active Flowchart Model*) της XASC επιτρέπει την δημιουργία σχημάτων διαδικασιών (process schemas) σαν την συλλογή από *ενεργά διαγράμματα ροής* (δηλαδή διαγράμματα ροής μαζί με τους τύπους γεγονότων (event types) που τα ενεργοποιούν)

Η δυνατότητα σύνθεσης τόσο διακριτών (discrete )Ηλεκτρονικών Υπηρεσιών όσο και Ηλεκτρονικών Υπηρεσιών που υποστηρίζουν συνεδρίες (session-oriented) είναι βασική απαίτηση στο AZTEC. Οι διάφορες εν εξελίξει συνεδρίες μοντελοποιούνται από το AZTEC με την μορφή αντικειμένων συνεδρίας (session objects) με καθορισμένη διεπαφή που περιλαμβάνει τόσο σύγχρονες κλήσεις συναρτήσεων (synchronous function calls) όσο και δημιουργία γεγονότων. Προγράμματα wrappers αναλαμβάνουν την μετατροπή της εσωτερικής αναπαράστασης των λειτουργιών και των γεγονότων σε μορφή κατανοητή από την διεπαφή SOAP που περιγράφει την Ηλεκτρονική Υπηρεσία. Στην **Ενότητα 2.2.5.2** είχαμε περιγράψει ένα σενάριο ηλεκτρονικής τηλεδιάσκεψης στο οποίο οι Ηλεκτρονικές Υπηρεσίας που καλούνται είναι «session-oriented» και στην **Εικόνα 2.16** είδαμε τα διάφορα αντικείμενα συνεδρίας που αναπτύσσονται και τις αλληλεπιδράσεις, τους τόσο με τις Ηλεκτρονικές Υπηρεσίες όσο και μεταξύ τους.

Ο πρωταρχικός σκοπός ενός σχήματος διαδικασίας (process schema) που συνθέτει πολλαπλά τέτοια αντικείμενα συνεδρίας είναι να καθορίσει πως πρέπει να αλληλεπιδρούν αυτά τα αντικείμενα. Η διαχείριση της αλληλεπίδρασης αυτών των αντικειμένων έχει να αντιμετωπίσει τρεις βασικές προκλήσεις:

**Γνώση του ποιος αλληλεπιδρά**. Δεν είναι πάντα εφικτό να γνωρίζουμε την ακριβή εσωτερική κατάσταση αντικειμένου συνεδρίας. Συχνά μάλιστα ίσως είναι απαραίτητο τα αντικείμενα αυτά να περιγράφονται σαν μαύρα κουτιά (black boxes) με καλά ορισμένη αλλά περιορισμένη διεπαφή αλληλεπίδρασης πού κάνει ακόμα πιο δύσκολη την παρακολούθηση της κατάστασης τους..

**Γνώση του τρόπου με τον οποίο τα αντικείμενα συνεδρίας θα πρέπει να μπορούν να επηρεάζονται εκατέρωθεν**. Η επίδραση ενός αντικειμένου συνεδρίας σε ένα άλλο μπορεί να είναι σχετικό με την εφαρμογή στα πλαίσια της οποίας

δημιουργούνται τα αντικείμενα. Επιπρόσθετα η απόκριση σε γεγονότα που δημιουργούνται από ένα αντικείμενο συνεδρίας πιθανώς να εξαρτάται από την κατάσταση στην οποία βρίσκονται τα υπόλοιπα αντικείμενα συνεδρίας και η συνολική διαδικασία.

**Γνώση των χρονικών στιγμών στις οποίες θα συμβούν οι διάφορες αλληλεπιδράσεις**. Οι αλληλεπιδράσεις με ανεξάρτητα αντικείμενα συνεδρίας δεν μπορούν να προβλεφθούν και να περιγραφούν στατικά, εκ των προτέρων μίας και κάθε αντικείμενο συνεδρίας μπορεί να αποκρίνεται σε πολλαπλά μη-ντετερμινιστικά γεγονότα τα οποία δεν είναι ορατά μέσω της «περιορισμένης» διεπαφής του.

Το AZTEC ανταποκρίνεται ικανοποιητικά στις πρώτες δύο «προκλήσεις» επιτρέποντας την δημιουργία διαγραμμάτων ροής που καθορίζουν τον τρόπο με τον οποίο αποκρίνεται σε ένα γεγονός που προέρχεται από ένα αντικείμενο συνεδρίας. Τα διαγράμματα ροής μπορούν να βολιδοσκοπήσουν (probe) τα αντικείμενα συνεδριών όσον αφορά την κατάσταση τους και να επηρεάσουν ένα ή περισσότερα τέτοια αντικείμενα. Τα διαγράμματα ροής παρέχουν πολλά από τα πλεονεκτήματα των μοντέλων ροών εργασίας (π.χ. τον διαχωρισμό της λογικής του ελέγχου (control logic) από τις ενέργειες (task) που εκτελούνται ενώ παράλληλα παρέχουν και χαμηλότερου επιπέδου δομικά στοιχεία για τον χειρισμό δεδομένων που προέρχονται από διαφορετικά γεγονότα. Επίσης το AZTEC παρέχει δομικά στοιχεία που επιτρέπουν σε ένα διάγραμμα ροής την ενεργοποίηση (enactment) νέων διαγραμμάτων ροής.

Το τρίτο στοιχείο είναι πιο περίπλοκο, γιατί δεν μπορεί να προβλεφθεί πότε ή πόσο συχνά θα προκύψει ένα συγκεκριμένο γεγονός από ένα αντικείμενο συνεδρίας. Αυτός είναι ο βασικός λόγος για τον οποίο ενσωματώσαμε το παράδειγμα γεγονότων (event paradigm) στο μοντέλο μας αντί απλά να επεκτείνουμε κάποιο υπάρχον μοντέλο ροών εργασίας όπου όλα τα διαγράμματα ροής θα συνδυάζονταν σε ένα σχήμα ροής εργασίας (workflow schema). Θα δούμε στην συνέχεια ότι αυτή είναι μία βασική διαφοροποίηση του μοντέλου μας από τα μοντέλα άλλων γλωσσών σύνθεσης Ηλεκτρονικών Υπηρεσιών. Την στιγμή που άλλες γλώσσες για να αποκριθούν σε εισερχόμενα μηνύματα ή γεγονότα, περιλαμβάνουν κάποιο δομημένο στοιχείο (structured construct) το οποίο μπλοκάρει και περιμένει το γεγονός να συμβεί, εμείς ορίζουμε ένα διάγραμμα ροής το οποίο εκτελείται όταν φτάσει το γεγονός χωρίς να μπλοκάρει την εκτέλεση του κανένα από τα ενεργά διαγράμματα ροής. Υπό αυτή την έννοια το μοντέλο μας είναι πλήρως ασύγχρονο σε αντίθεση με άλλα που είναι «μερικώς» ασύγχρονα.

### 3.1.1 Αλληλεπιδράσεις μεταξύ διαγραμμάτων ροής

Τα γεγονότα δεν έχουν κανένα περιορισμό στην συχνότητα με την οποία μπορούν να καταφτάνουν και οι ενεργοποιήσεις διαγραμμάτων ροής μπορεί να έχουν οσοδήποτε μεγάλη διάρκεια προκαλώντας έτσι «διαφυλλωμένες» (interleaved) εκτελέσεις διαγραμμάτων ροής. Σε τέτοιες περιπτώσεις μπορεί να απαιτηθεί η εισαγωγή προτεραιοτήτων στην εκτέλεση των διαγραμμάτων ροής ή και η αλληλεπίδραση μεταξύ τους για να συγχρονιστεί η εκτέλεση τους. Το μοντέλο μας υποστηρίζει την αλληλεπίδραση μεταξύ ενεργών διαγραμμάτων ροής όπου ένα διάγραμμα ροής μπορεί να αναστείλει (suspend), να εξετάσει (examine), να αλλάξει (modify) ή να ενεργοποιήσει ξανά (resume) ένα σταματημένο διάγραμμα ροής.

Μια βασική απαίτηση που τα διαγράμματα ροής ικανοποιούν σε αντίθεση με μία ολοκληρωμένη γλώσσα προγραμματισμού για παράδειγμα, είναι η δυνατότητα προσπέλασης και πιθανώς παραποίησης της κατάστασης ενεργών αντικειμένων συνεδρίας. Τα διαγράμματα ροής με την περιορισμένη αλλά ξεκάθαρη και «εύκολα προσβάσιμη» λογική ελέγχου της ροής (control flow logic) εξυπηρετούν αυτό το σκοπό πολύ αποδοτικότερα από αυθαίρετα κομμάτια κώδικα. Αυτός είναι και ο λόγος για τον οποίο περιοριζόμαστε σε ένα βασικό σύνολο από δομικά στοιχεία (constructs) τα οποία έχουν ξεκάθαρη σημασιολογία και δυνατότητα παρακολούθησης της κατάστασης εκτέλεσης τους.

Αναφέρθηκε νωρίτερα ότι δεν υπάρχει κανένας περιορισμός στον ρυθμό άφιξης γεγονότων από τα διάφορα αντικείμενα συνεδρίας με αποτέλεσμα να είναι συχνό φαινόμενο η ταυτόχρονη ύπαρξη πολλών ενεργών διαγραμμάτων ροής (interleaved flowchart enactment). Τα διαγράμματα αυτά συχνά πρέπει να αλληλεπιδρούν μεταξύ τους επερωτώντας και πιθανώς αλλάζοντας την κατάσταση εκτέλεσης τους. Το μοντέλο μας υποστηρίζει τέσσερις τρόπους αλληλεπίδρασης μεταξύ ενεργών διαγραμμάτων ροής.

**Με την δημιουργία γεγονότων που προκαλούν την εκκίνηση νέων διαγραμμάτων ροής.** Το AZTEC περιλαμβάνει ένα τύπο ενέργειας η οποία ονομάζεται «ενέργεια γεγονότος» (event task) και αποσκοπεί στην δημιουργία ενός γεγονότος που προκαλεί την ενεργοποίηση (enactment) ενός νέου διαγράμματος ροής. Τα γεγονότα στο μοντέλο του AZTEC μεταφέρουν δεδομένα (typed events) τα οποία χρησιμοποιούνται σαν δεδομένα εισόδου (input parameters) στα διαγράμματα ροής. Οι διαφορετικοί τύποι γεγονότων (που προέρχονται από διαφορετικές ενέργειες γεγονότων) καθορίζονται από το όνομα τους και από το όνομα και τον τύπο των παραμέτρων εισόδου τους (input

parameters). Οι παράμετροι αυτοί χρησιμοποιούνται για το «πέρασμα δεδομένων» (data passing) τόσο από ένα αντικείμενο συνεδρίας σε ένα ενεργό διάγραμμα ροής, όσο και μεταξύ διαγραμμάτων ροής. Οι παράμετροι τόσο των γεγονότων όσο και των διαγραμμάτων ροής

**f2(in:some_string,some_set out:some_int)**

```
<flowchart_root name="f2">
   <input_params>
      <param name="param1">
         <value_string/>
      </param>
      <param name="param2">
         <value_set/>
      </param>
   </input_params>
   <output_params>
      <param name="param3">
         <value_int/>
      </param>
   </output_params>
   …
</flowchart_root>
```

**f1(in:… out:…)**

```
<flowchart_root name="f1">
   …
   <task name="task1"
type="event" action="f2"
mode="request_response">
   <input_params>
      <in_param>
         <value_string>
            some
         </value_string>
      </in_param>
      <in_param>
         <value_variable>
            some_set
         </value_variable>
      </in_param>
   </input_params>
   <return_params>
      <return_param name="return">
         <value_int/>
      </return_param>
   </return_params>
</flowchart_root>
```

**Εικόνα 3.1 Μία ενέργεια γεγονότος και το διάγραμμα ροής που ενεργοποιεί σε XASC**

είναι δεδομένα XML βασισμένα σε ένα Σχήμα XML που εξαρτάται από την συγκεκριμένη διαδικασία. Στην **Εικόνα 3.1** βλέπουμε ένα παράδειγμα ορισμού παραμέτρων ενός γεγονότος και ενός διαγράμματος ροής που ενεργοποιείται όταν συμβεί αυτό το γεγονός. Το απλό αυτό σχήμα διαδικασίας ορίζει δύο διαγράμματα ροής, τα **f1** και **f2**. Η «ενέργεια γεγονότος» (event task) task1 στο διάγραμμα ροής f1 δημιουργεί το γεγονός f2 ( το οποίο είναι «typed event» γιατί μεταφέρει και τις παραμέτρους «some_string» και «some_set» μαζί του) Το γεγονός f2 προκαλεί την ενεργοποίηση του ομώνυμου διαγράμματος ροής f2 με τις συγκεκριμένες παραμέτρους που «μεταφέρει» . Βλέπουμε ότι μπορούμε να έχουμε είτε απλούς τύπους του Σχήματος XML (π.χ. value_string) είτε και σύνθετους (π.χ. value_set). Η ενέργεια γεγονότος μπορεί να καθορίσει εάν το νέο διάγραμμα ροής θα εκτελεστεί ανεξάρτητα, χωρίς ανάγκη συγχρονισμού (spawning) ή εάν θα μπλοκάρει η εκτέλεση του διαγράμματος ροής που περιλαμβάνει την ενέργεια περιμένοντας το νέο διάγραμμα ροής να ολοκληρώσει την εκτέλεση του, για να συνεχίσει (subprocess). Σε αυτή την περίπτωση το «νέο διάγραμμα ροής» μπορεί να έχει δεδομένα εξόδου (return

parameters) τα οποία επιστρέφονται στο διάγραμμα ροής που περιλάμβανε την ενέργεια γεγονότος. Οι ενέργειες γεγονότων αναπαρίστανται γραφικά με ένα εξάγωνο.

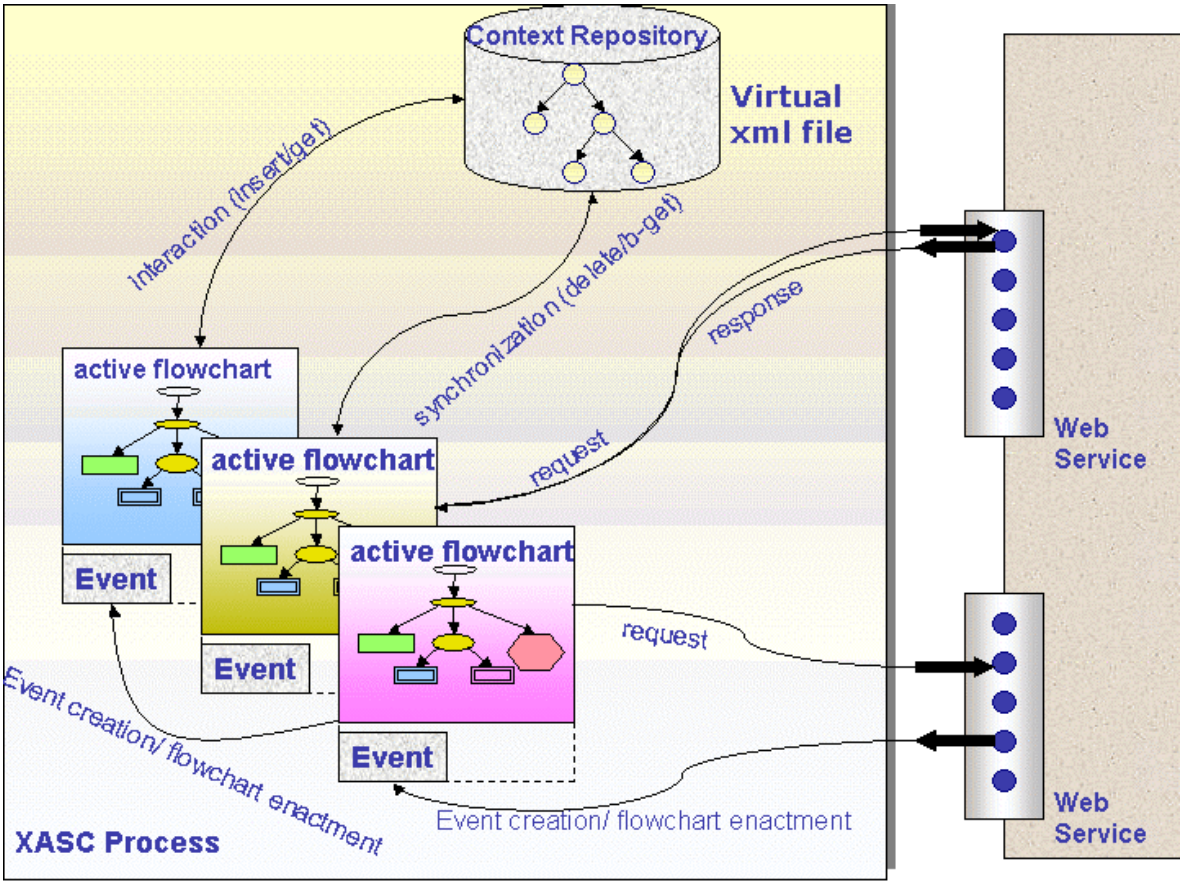
**Με τον διαμοιρασμό δεδομένων στο Context Repository (CR).** Το CR είναι μία «κοινή», διαμοιραζόμενη αποθήκη δεδομένων, μεταξύ των ενεργών διαγραμμάτων ροής της ίδιας διαδικασίας. Όταν δημιουργείται ένα σχήμα διαδικασίας (process schema) στην XASC, παράλληλα δημιουργείται και το CR σαν ένα εικονικό (virtual) XML έγγραφο, στιγμιότυπο ενός Σχήματος XML (XML Schema) καθορισμένου από τον προγραμματιστή της εφαρμογής. Υπάρχει ένα σύνολο από λειτουργίες του CR τις οποίες μπορούν να εκτελέσουν τα διαγράμματα ροής μίας διαδικασίας:

a. Εισαγωγή μίας μεταβλητής δεδομένου σε μία «θέση» (XPath) στο CR. Η λειτουργία αυτή εκτελείται με την ενέργεια CR Insert.
b. Ανάγνωση μίας μεταβλητής δεδομένου από μία «θέση» (XPath) του CR. Στην ανάγνωση υπάρχουν δύο περιπτώσεις ανάλογα με το αν η μεταβλητή που επιχειρούμε να διαβάσουμε στο δεδομένο XPath υπάρχει ή δεν έχει δημιουργηθεί ακόμα. Στην περίπτωση της «απλής» ανάγνωσης (CR Get) σε περίπτωση που η μεταβλητή δεν βρεθεί η ενέργεια δεν επιστρέφει τίποτα και το διάγραμμα ροής συνεχίζει την εκτέλεση του. Στην περίπτωση της «μπλοκαρισμένης» ανάγνωσης (CR BlockingGet) σε περίπτωση που δεν έχει δημιουργηθεί η καταχώρηση μεταβλητής στο συγκεκριμένο XPath το διάγραμμα ροής μπλοκάρει και περιμένει μέχρι να δημιουργηθεί η μεταβλητή. Όταν αυτό συμβεί, επιστρέφει την τιμή της και το διάγραμμα ροής συνεχίζει την εκτέλεση του. Η χρήση της «μπλοκαρισμένης» ανάγνωσης είναι ο βασικός τρόπος συγχρονισμού μεταξύ διαγραμμάτων ροής της ίδιας διαδικασίας.
c. Ανανέωση (update) της τιμής μίας μεταβλητής στο CR. Η λειτουργία αυτή εκτελείται με την ενέργεια CR Update.
d. Διαγραφή μίας μεταβλητής δεδομένου από το CR. Η λειτουργία αυτή εκτελείται με την ενέργεια CR Delete. Η διαγραφή μίας μεταβλητής από το CR από ένα διάγραμμα ροής σε μία διαδικασία όπου όλα τα διαγράμματα ροής ξεκινάνε την εκτέλεση τους με μία «μπλοκαρισμένη» ανάγνωση σε αυτή την μεταβλητή, θα υλοποιούσε μία κατάσταση όπου μόνο το πρώτο γεγονός που θα συμβεί πρέπει να εκτελεστεί.

**Επερωτώντας την κατάσταση εκτέλεσης άλλων ενεργών διαγραμμάτων ροής και ακόμα και αναστέλλοντας ή αλλάζοντας την κατάσταση εκτέλεσης τους**. Οι ενέργειες suspendAllFcs και ResumeAllFcs αναστέλλουν και αντίστοιχα εκκινούν ξανά όλα τα ενεργά διαγράμματα ροής ενός συγκεκριμένου τύπου στα πλαίσια μίας διαδικασίας.

**Τέλος επιβάλλοντας προτεραιότητες στην εκτέλεση των διαγραμμάτων ανάλογα με τον τύπο τους**.

Οι διάφορες αλληλεπιδράσεις μεταξύ των ενεργών διαγραμμάτων ροής, τόσο μεταξύ τους όσο και με εξωτερικές Ηλεκτρονικές Υπηρεσίες φαίνεται στην **Εικόνα 3.2**




**Εικόνα 3.2 Οι αλληλεπιδράσεις στα πλαίσια μίας διαδικασίας στην XASC**

## 3.1.2 Στοιχεία Ελέγχου και Είδη Ενεργειών στην XASC

Το κυρίως μέρος ( main body) ενός διαγράμματος ροής περιγράφεται με την χρήση μίας δομημένης γλώσσας περιγραφής ροών εργασίας που εισήχθη στο [KHB00] επεκταμένη κατάλληλα ώστε να υποστηρίζει παραμέτρους εισόδου και εξόδου και τοπικές μεταβλητές. Παράλληλα, στο σύνολο των βασικών δομικών στοιχείων της γλώσσας (core constructs) έχουμε προσθέσει κάποια δομικά στοιχεία τα οποία θεωρήσαμε απαραίτητα είτε γιατί απλοποιούν πολύ κάποια «κατηγορία» διαγραμμάτων ροής παρέχοντας πιο συμπαγείς (compact) ορισμούς είτε γιατί αντιμετωπίζουν προβλήματα των οποίων την σημασιολογία δεν «πιάνουν» επακριβώς τα βασικά στοιχεία όπως αυτά που περιγράφουν τα Πρότυπα 8 και 14. Αναφορικά, σε σχέση με τα πρότυπα ροών εργασίας που θα δούμε στην συνέχεια ότι αναπαριστούν, τα βασικά δομικά στοιχεία της XASC είναι τα Sequence

(Πρότυπο 1), Parallel (Πρότυπα 2,3,12 και 13), Choice (Πρότυπα 4 και 5) και While_do (Πρότυπα 10 και 12) ενώ τα «δευτερεύοντα» (redundant) είναι το Switch (Πρότυπα 4 και 5) και το Hparallel (Πρότυπα 8 και 14). Τα δομικά στοιχεία της γλώσσας XASC αναπαρίστανται γραφικά με ελλείψεις. Η χρήση αυτού του δομημένου μοντέλου ροών εργασίας έχει ως βασικό κίνητρο το γεγονός ότι διευκολύνει την επερώτηση της κατάστασης των ενεργών διαγραμμάτων ροής. Η επιλογή μας να περιλάβουμε ένα «σχετικά μικρό» σύνολο από δομικά στοιχεία με ξεκάθαρη όμως σημασιολογία μπορεί να έχει ως αποτέλεσμα ορισμένα από τα πρότυπα ροών εργασίας που θα δούμε στην συνέχεια να απαιτούν τον συνδυασμό δύο ή και τριών δομικών στοιχείων της XASC για να αναπαρασταθούν. Αυτό είναι ανεκτό σε σχέση με την δημιουργία διαγραμμάτων ροής με «κρυμμένη» ή «αδιαφανή» λειτουργικότητα και επομένως με αδυναμία παρακολούθησης της κατάστασης εκτέλεσης τους.

Η XASC υποστηρίζει τον ορισμό αρκετών διαφορετικών τύπων ενεργειών (tasks) οι οποίες συνδυάζονται με παραμέτρους εισόδου και παράγουν παραμέτρους εξόδου. Εκτός από τις ενέργειες γεγονότων (event tasks) που περιγράψαμε νωρίτερα, υποστηρίζονται οι παρακάτω τύποι ενεργειών

**Εξωτερικές ενέργειες** (exteral tasks). Οι εξωτερικές ενέργειες εκτελούν *σύγχρονες* ή *ασύγχρονες* κλήσεις συναρτήσεων (function calls) προς διακριτές (discrete) ή «βασισμένες σε συνεδρίες» (session-oriented) Ηλεκτρονικές Υπηρεσίες. Εάν η κλήση είναι *σύγχρονη*, το διάγραμμα ροής που περιλαμβάνει την ενέργεια «μπλοκάρει» την εκτέλεση του περιμένοντας την απάντηση από την Ηλεκτρονική Υπηρεσία να φτάσει. Αυτός ο τρόπος επικοινωνίας (communication mode), από την οπτική γωνία του διαγράμματος ροής αντιστοιχεί στην Αποστολή Αίτησης-Απάντηση (solicit response) που είδαμε κατά την περιγραφή της WSDL. Από την οπτική γωνία της καλούμενης Ηλεκτρονικής Υπηρεσίας η οποία δέχεται ένα μήνυμα και απαντά σε αυτό είναι η Αίτηση-Απάντηση (request-response) δηλαδή ο κλασσικότερος τρόπος αλληλεπίδρασης με μία Ηλεκτρονική Υπηρεσία. Εάν η κλήση είναι *ασύγχρονη*, το διάγραμμα ροής που περιλαμβάνει την εξωτερική ενέργεια, αφού εκτελέσει την ενέργεια συνεχίζει την εκτέλεση του, χωρίς να περιμένει απάντηση. Από την πλευρά του διαγράμματος ροής που περιλαμβάνει την εξωτερική ενέργεια αυτό αντιστοιχεί στον τρόπο επικοινωνίας Ειδοποίηση (notification) ενώ από την πλευρά της καλούμενης Ηλεκτρονικής Υπηρεσίας που δέχεται ένα μήνυμα αλλά δεν απαντάει σε αυτό, αντιστοιχεί στον Μονόδρομο (one-way) τρόπο επικοινωνίας. Εάν η Ηλεκτρονική Υπηρεσία επιθυμεί μετά από κάποιο διάστημα να επιστρέψει μία απάντηση μπορεί να το κάνει δημιουργώντας ένα από τα προκαθορισμένα γεγονότα του σχήματος της συγκεκριμένης διαδικασίας (process schema). Η δημιουργία ενός τέτοιου γεγονότος θα προκαλέσει την ενεργοποίηση

(enactment) ενός νέου διαγράμματος ροής που θα εκτελεστεί ανεξάρτητα από την Ηλεκτρονική Υπηρεσία που το ενεργοποίησε. Από την πλευρά του AZTEC αυτό αντιστοιχεί στον Μονόδρομο (one-way) τρόπο επικοινωνίας ενώ από την πλευρά της Ηλεκτρονικής Υπηρεσίας που δημιούργησε το γεγονός αντιστοιχεί στην Ειδοποίηση (notification). Η μοναδικός τρόπος επικοινωνίας ο οποίος δεν μπορεί να μοντελοποιηθεί από την XASC είναι η Αίτηση/Απάντηση (request-response). Προς το παρόν δεν μπορεί να μοντελοποιηθεί η κατάσταση στην οποία μία εξωτερική Ηλεκτρονική Υπηρεσία στέλνει ένα μήνυμα σε ένα διάγραμμα ροής και το διάγραμμα ροής στέλνει πίσω μία απάντηση. Το γεγονός ότι σε ένα ενεργό διάγραμμα ροής μπορούν να οριστούν παράμετροι εξόδου σημαίνει ότι η λήψη ενός μηνύματος (γεγονός που προκάλεσε την ενεργοποίηση του διαγράμματος ροής) ακολουθείται από την επιστροφή κάποιων αποτελεσμάτων. Υπό μία έννοια αυτός ο τρόπος επικοινωνίας είναι «request-response» παρόλαυτα δεν μπορούμε να ισχυριστούμε ότι μπορεί η XASC να μοντελοποιήσει όλες τις περιπτώσεις επικοινωνίας «request-response» . Οι εξωτερικές ενέργειες αναπαρίστανται γραφικά με ένα διπλό ορθογώνιο.

**Εσωτερικές ενέργειες** (Internal Tasks). Οι εσωτερικές ενέργειες αναπαρίστανται με απλό ορθογώνιο και χρησιμοποιούν εσωτερικές λειτουργίες/ μεθόδους του AZTEC. Ως εσωτερικές ενέργειες θεωρούνται οι ενέργειες του CR που είδαμε αναλυτικά νωρίτερα, όπως και ενέργειες χρονισμού για την αναμονή για ένα χρονικό διάστημα ή ως ένα χρονικό σημείο (wait for, wait until tasks). Επίσης εσωτερικές ενέργειες θεωρούνται οι ενέργειες που χειρίζονται την αναστολή, την επανεκκίνηση και τον τερματισμό των διαγραμμάτων ροής ενός τύπου ( suspendAllFcs, resumeAllFcs, CancelAllFcs), αλλά και τον τερματισμό μίας ολόκληρης διαδικασίας (terminateProcess). Ως εσωτερικές ενέργειες θεωρούνται αυτές που επερωτούν την κατάσταση εκτέλεσης ενεργών διαγραμμάτων ροής, βασισμένες σε τεχνικές από το [CHK01]. Τέλος εσωτερική ενέργεια είναι και η κενή ενέργεια (empty task).

Έχοντας δει τα βασικά στοιχεία του μοντέλου ενεργών διαγραμμάτων ροής του AZTEC, και της γλώσσας XASC, μπορούμε να προχωρήσουμε στην παρουσίαση των προτύπων ροών εργασίας και τους τρόπους αναπαράστασης τους από το μοντέλο μας. Για κάθε πρότυπο θα περιγράφουμε ένα χαρακτηριστικό παράδειγμα που να τονίζει την συχνότητα της εμφάνισης του (όσο είναι δυνατόν από το παράδειγμα ροής εργασίας της ενότητας 2.1.2) και έπειτα θα επιχειρούμε την αναπαράσταση του, τόσο με το μοντέλο των ενεργών διαγραμμάτων ροής όσο και με την γλώσσα *XASC*. Με αυτό τον τρόπο θα επιτύχουμε τόσο την διαγραμματική όσο και την συντακτική παρουσίαση της γλώσσας μας. Σε περίπτωση που κάποιο πρότυπο δεν εκφράζεται άμεσα από κάποιο στοιχείο της γλώσσας μας θα επιχειρούμε μία «σύνθεση» του με τον

συνδυασμό δύο ή περισσοτέρων στοιχείων, χρησιμοποιώντας τεχνικές που προτάθηκαν στα [AHK+02][ADH+02][WAD+01].

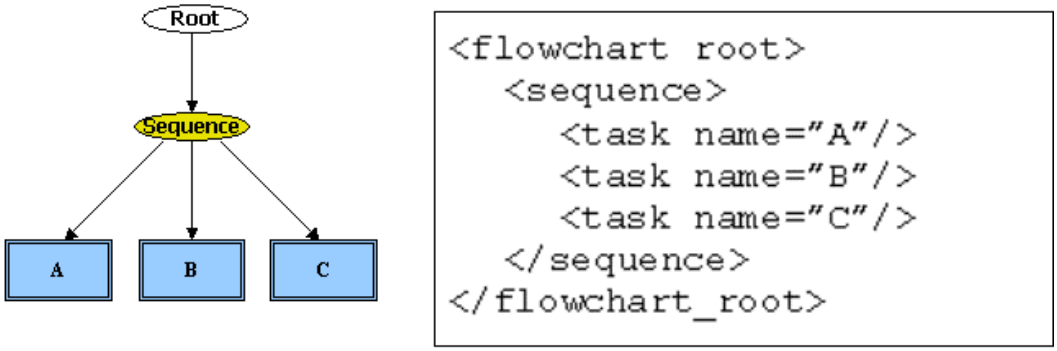
## *3.2 Δομικά Πρότυπα Ροών Εργασίας και η XASC*

**Πρότυπο1 Ακολουθία.**(Sequence)

Μία ενέργεια σε μια διαδικασία Ροής Εργασίας (workflow process) εκτελείται μετά την ολοκλήρωση της εκτέλεσης μίας άλλης ενέργειας στην ίδια διαδικασία. Συχνά μπορεί να συναντηθεί ως ακολουθιακή δρομολόγηση (sequential routing) ή σειριακή δρομολόγηση (serial routing). Για παράδειγμα η ενέργεια «*αποστολή_λογαριασμού*» εκτελείται μετά την ολοκλήρωση της εκτέλεσης της ενέργειας «*αποστολή_προϊόντων*».

**Υλοποίηση Προτύπου 1 στην XASC**

Στην *XASC* η Ακολουθία υποστηρίζεται άμεσα με το στοιχείο «*Sequence*» όπως φαίνεται στην **Εικόνα 3.3**.




**Εικόνα 3.3 Η ακολουθία στο AZTEC**

Κατά διάρκεια της παρουσίασης των προτύπων θα αποφύγουμε αρκετές συντακτικές λεπτομέρειες της γλώσσας εστιάζοντας στα βασικά της σημεία.

Στην *BPEL4WS* η Ακολουθία μπορεί να αναπαρασταθεί τόσο από το στοιχείο «sequence» όσο και χρησιμοποιώντας ένα σύνδεσμο ελέγχου (control link) από την μία ενέργεια στην άλλη.

Στην *BPML* η Ακολουθία υποστηρίζεται επίσης με το στοιχείο «sequence».

**Πρότυπο2 Παράλληλη Διακλάδωση** (Parallel Split)

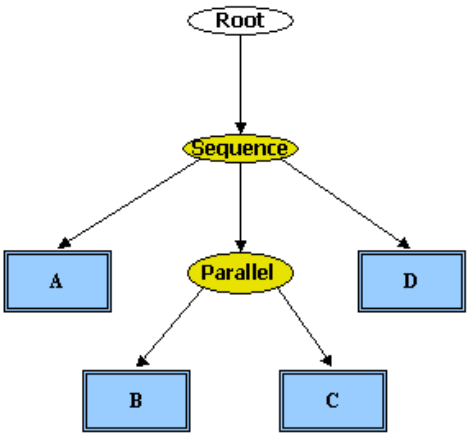
Είναι ένα σημείο στην διαδικασία (process) όπου μία ίνα ελέγχου (control thread) διακλαδώνεται σε πολλαπλές ίνες οι οποίες εκτελούνται παράλληλα, επιτρέποντας έτσι σε ενέργειες να εκτελεστούν ταυτόχρονα ή με τυχαία σειρά. Συχνά μπορεί να συναντηθεί ως AND-split, παράλληλη δρομολόγηση (parallel routing) ή διακλάδωση (fork). Για παράδειγμα η ενέργεια «πληρωμή» ακολουθείται από τις ενέργειες «αποστολή_προϊόντων» και «ειδοποίηση πελάτη».

**Πρότυπο3 Συγχρονισμός** (Synchronization)

Είναι ένα σημείο στην διαδικασία όπου πολλές παράλληλες διακλαδώσεις (branches) συγκλίνουν σε μία ίνα ελέγχου, συγχρονίζοντας έτσι τις παράλληλες διακλαδώσεις. Μια παραδοχή είναι ότι κάθε μία από τις παράλληλες διακλαδώσεις εκτελείται μία μόνο φορά. Συχνά μπορεί να συναντηθεί ως AND-join, ραντεβού (rendezvous) ή συγχρονιστής (synchronizer). Για παράδειγμα η ενέργεια «αρχειοθέτηση_συναλλαγής» εκτελείται μετά την ολοκλήρωση αμφότερων των ενεργειών «αποστολή_προϊόντων» και «παραλαβή_πληρωμής».

**Υλοποίηση Προτύπων 2 και 3 στην XASC.**

Στην *XASC* η Παράλληλη Διακλάδωση υποστηρίζεται άμεσα με το στοιχείο «*Parallel*» όπως φαίνεται στην **Εικόνα 3.4**. Οι ενέργειες που ορίζονται σαν «παιδιά» του κόμβου «*Parallel*» στο διάγραμμα ροής εκτελούνται παράλληλα. Ορίζοντας μία ενέργεια μετά το «*Parallel*», δηλαδή σαν αμφιθαλή του (sibling) στο διάγραμμα ροής, αναπαρίσταται ο Συγχρονισμός μίας και ο έλεγχος «φεύγει» από το «*Parallel*» όταν ολοκληρώσουν την εκτέλεση τους όλα τα παιδιά του.

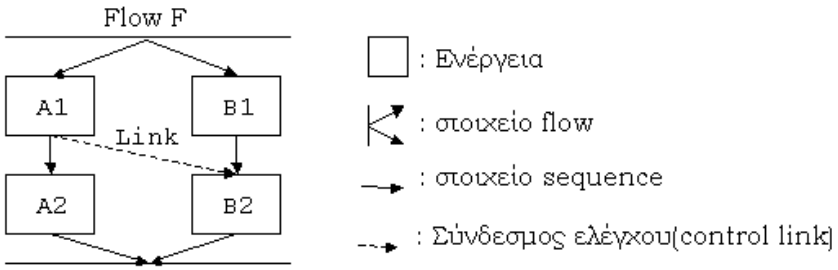



```
<flowchart_root>
   <sequence>
      <task name="A"/>
      <parallel>
         <task name="B"/>
         <task name="C"/>
      </parallel>
      <task name="D"/>
   </sequence>
</flowchart root>
```

**Εικόνα 3.4 Παράλληλη Διακλάδωση και Συγχρονισμός στο AZTEC**

Στην *BPEL4WS* η Παράλληλη Διακλάδωση υλοποιείται ορίζοντας τις ενέργειες σαν συνιστώσες του στοιχείου «flow». Τοποθετώντας μία ενέργεια μετά το «flow» υλοποιείται ο

συγχρονισμός (ακριβώς όπως με την χρήση του «*Parallel*» στο AZTEC). Ένας άλλος τρόπος υλοποίησης της παράλληλης διακλάδωσης και του συγχρονισμού είναι χρησιμοποιώντας συνδέσμους ελέγχου (control links) στα πλαίσια ενός «flow». Χρησιμοποιώντας αυτή τη λύση μπορούν να αναπαρασταθούν σημεία συγχρονισμού ανάμεσα σε παράλληλα εκτελούμενες ίνες ελέγχου. Η **Εικόνα 3.5** αναπαριστά δύο παράλληλους βρόγχους (branches) οι οποίοι όμως έχουν ένα σύνδεσμο ελέγχουν ανάμεσα στις ενέργειες A1 και B2. Η χρήση αυτού του συνδέσμου ορίζει ότι για να εκτελεστεί η ενέργεια B2 δεν πρέπει να έχει ολοκληρώσει την εκτέλεση της μόνο η B1 (στοιχείο sequence) αλλά πρέπει να έχει ολοκληρώσει την εκτέλεση της και η A1, παρόλο που ανήκει σε παράλληλο βρόγχο (branch). Στην *BPML* η παράλληλη διακλάδωση υλοποιείται ορίζοντας τις ενέργειες σαν συνιστώσες του στοιχείου «all».Τοποθετώντας μία ενέργεια μετά το «all» υλοποιείται ο συγχρονισμός. Ένας άλλος τρόπος υλοποίησης της παράλληλης διακλάδωσης και του συγχρονισμού είναι με χρήση των στοιχείων «spawn» και «synch». Πιο συγκεκριμένα με το «spawn» (στα πλαίσια ενός «all») δημιουργούνται




**Εικόνα 3.5 Χρήση συνδέσμου ελέγχου στα πλαίσια ενός στοιχείου «flow»**

δύο υποδιαδικασίες οι οποίες εκτελούνται παράλληλα Στην κύρια διαδικασία υπάρχουν στα πλαίσια ενός «all», δύο στοιχεία «synch» καθένα από τα οποία παραμένει μπλοκαρισμένο μέχρι να φτάσει ένα κατάλληλο «signal». Κάθε μία από τις παράλληλες υποδιαδικασίες όταν ολοκληρώσει την εκτέλεση της δημιουργεί το κατάλληλο «signal» με το στοιχείο «raise». Η λύση με τα «spawn» και «synch» θεωρείται γενικότερη από αυτή με την χρήση μόνο του «all» μίας και μπορεί να εκφράσει ακόμα και εξαρτήσεις ανάμεσα σε παράλληλα εκτελούμενους βρόγχους (όπως και η BPEL4WS με τους συνδέσμους ελέγχου).

**Πρότυπο4 Αποκλειστική Επιλογή** (Exclusive Choice)

Είναι ένα σημείο στην διαδικασία όπου επιλέγεται μία ανάμεσα σε πολλαπλές διακλαδώσεις βάσει κάποιων δεδομένων ελέγχου (control data). Συχνά μπορεί να συναντηθεί ως XOR-join, δρομολόγηση υπό συνθήκη (conditional routing), επιλογή

(switch) ή απόφαση (decision). Για παράδειγμα η ενέργεια «Πληρωμή» ακολουθείται είτε από την ενέργεια «Πληρωμή_με_Μετρητά» είτε από την ενέργεια «Χρέωση Κάρτας».

**Πρότυπο5 Απλή Συγχώνευση** (Simple Merge)

Είναι ένα σημείο στην διαδικασία όπου δύο ή περισσότερες εναλλακτικές διακλαδώσεις (branches) συγχωνεύονται σε μία χωρίς συγχρονισμό. Προφανώς μόνο μία από τις εναλλακτικές διακλαδώσεις εκτελείται και δεν έχουμε καμία έννοια παράλληλης εκτέλεσης. Συχνά μπορεί να συναντηθεί ως XOR-join, ασύγχρονη ένωση (asynchronous join) ή συγχώνευση (merge). Για παράδειγμα, η ενέργεια «Τέλος_Παραγγελίας» εκτελείται είτε μετά από την ενέργεια «Πληρωμή_με_Μετρητά» είτε μετά την ενέργεια «Χρέωση_Κάρτας».

**Υλοποίηση Προτύπων 4 και 5 στην XASC**

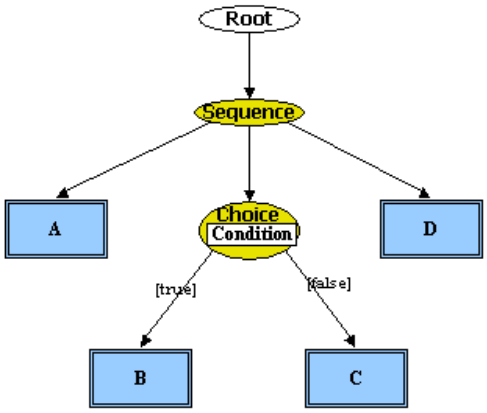
Στην *XASC* η αποκλειστική επιλογή υποστηρίζεται άμεσα είτε με το «*Choice*» για την περίπτωση που έχουμε διακλάδωση με δύο «παρακλάδια» είτε με το στοιχείο «*Switch*» για την περίπτωση της διακλάδωσης με περισσότερα από δύο παρακλάδια. Πρακτικά το «*Switch*» είναι ένα «δευτερεύων» (redundant) στοιχείο υπό την έννοια ότι ένα «Switch» μπορεί να υλοποιηθεί με συνδυασμό πολλών στοιχείων «*Choice*». Παρόλα αυτά συχνά η χρήση του «*Switch*» προσφέρει μεγάλη απλούστευση στην μορφή των διαγραμμάτων ροής. Θα δούμε στην συνέχεια ότι μέρος των πιθανών επεκτάσεων της γλώσσας είναι και η επέκταση του στοιχείου «*Switch*» για την επιλογή ενός βρόγχου ανάλογα με κάποιο γεγονός. Τοποθετώντας μία ενέργεια σαν αμφιθαλή του «Choice»/«Switch» αυτή θα εκτελεστεί αμέσως μόλις το «ενεργό» παρακλάδι ολοκληρώσει την εκτέλεση τού (**Εικόνα 3.6**).

Στην *BPEL4WS* η αποκλειστική επιλογή και η απλή συγχώνευση υλοποιούνται με χρήση του στοιχείου «switch». Ένας άλλος τρόπος υλοποίησης τους είναι με συνδέσμους ελέγχου όπου κάθε σύνδεσμος έχει προσαρτημένη (attached) μία συνθήκη μετάβασης (transition condition). Για να επιβεβαιωθεί ότι μόνο ένα από τα παρακλάδια μπορεί να επιλεχθεί, οι συνθήκες στις διάφορες συνδέσεις πρέπει να είναι αμοιβαία αποκλειόμενες.

Στην *BPML* η αποκλειστική επιλογή και η απλή συγχώνευση υλοποιούνται επίσης με χρήση του στοιχείου «switch».

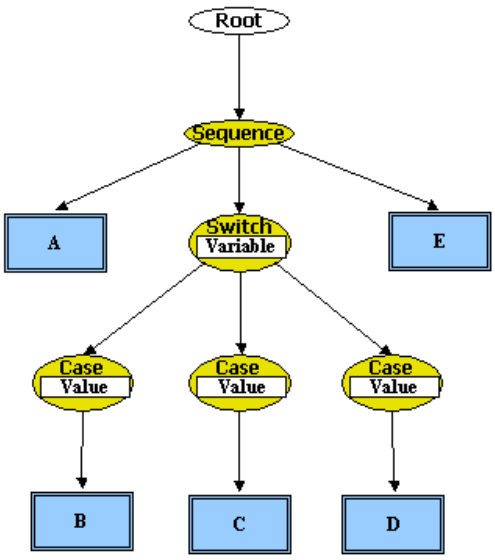
**Πρότυπο6 Πολλαπλή Επιλογή** (Multiple Choice)

Είναι ένα σημείο στην διαδικασία όπου επιλέγονται κάποιες (μία ή περισσότερες) ανάμεσα σε πολλαπλές διακλαδώσεις βάσει κάποιας απόφασης ή δεδομένων ελέγχου (control data) και εκτελούνται παράλληλα. Συχνά μπορεί να συναντηθεί ως δρομολόγηση υπό συνθήκη (conditional routing), επιλογή (selection) ή OR-split. Για παράδειγμα μετά την ενέργεια «εκτίμηση_ζημειάς» τόσο η ενέργεια «κλήση_πυροσβεστικής» όσο και η ενέργεια «κλήση_ασφαλιστικής_εταιρίας» μπορεί να εκτελεστεί. Ωστόσο είναι πιθανό να πρέπει να εκτελεστούν και οι δύο αυτές ενέργειες.




```
<flowchart_root>
  <sequence>
    <task name="A"/>
    <choice condition="…">
      <true>
        <task name="B"/>
      </true>
      <false>
        <task name="C"/>
      </false>
    </choice>
    <task name="D"/>
  </sequence>
</flowchart_root>
```

**α)Με χρήση του στοιχείου Choice**




```
flowchart_root>
 <sequence>
   <task name-"A"/>
   <switch condition="…">
     <case value="…">
       <task name="B"/>
     </case>
     <case value="…">
       <task name="C"/>
     </case>
     <case value="…">
       <task name="D"/>
     </case>
   </switch>
   <task name="E"/>
 </sequence>
```

**β)Με χρήση του στοιχείου Switch**

**Εικόνα 3.6 Αποκλειστική Επιλογή και Απλή Συγχώνευση**

**Πρότυπο7 Συγχώνευση με Συγχρονισμό** (Synchronizing Merge)

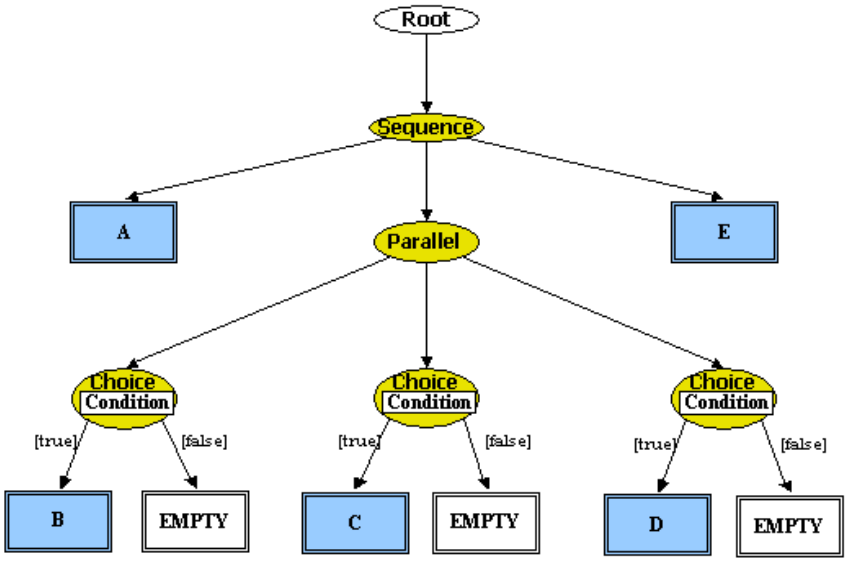
Είναι ένα σημείο στη διαδικασία όπου πολλές διακλαδώσεις (branches) συγκλίνουν σε μία ίνα εκτέλεσης (control thread). Μερικές από αυτές τις διακλαδώσεις είναι ενεργές (δηλαδή εκτελούνται) και μερικές όχι. Αν μόνο μία διακλάδωση είναι ενεργή, η ενέργεια μετά την συγχώνευση εκτελείται μόλις η διακλάδωση ολοκληρώσει την εκτέλεση της. Αν όμως υπάρχουν παραπάνω από μία ενεργές διακλαδώσεις τότε πρέπει να γίνει συγχρονισμός όλων των ενεργών διακλαδώσεων πριν εκτελεστεί η επόμενη ενέργεια. Μία υπόθεση εδώ είναι ότι κάθε μία από τις ενεργές διακλαδώσεις μπορεί να εκτελεστεί μόνο μία φορά. Συχνά μπορεί να συναντηθεί ως ένωση με συγχρονισμό (synchronizing join). Για παράδειγμα είτε μετά από την μία, είτε μετά και από τις δύο ενέργειες «κλήση_πυροσβεστικής» και «κλήση_ασφαλιστικής_εταιρίας», πρέπει να εκτελεστεί η ενέργεια «διεξαγωγή_αποτελέσματος» ακριβώς μία φορά.

**Υλοποίηση Προτύπων 6 και 7 στην XASC**

Στην XASC η σημασιολογία της Πολλαπλής Επιλογής/ Συγχώνευσης με Συγχρονισμό δεν υποστηρίζεται άμεσα από κάποιο δομικό στοιχείο (construct). Μία συνθετική λύση μπορεί να δοθεί με συνδυασμό των στοιχείων «*Parallel*» και «*Choice*» με τεχνική που προτάθηκε στο [AHK+02]. Τοποθετούνται δύο (ή περισσότερα) στοιχεία «*Choice*» μέσα σε ένα στοιχείο «*Parallel*». Για κάθε «*Choice*» αν έχει αληθή συνθήκη εκτελείται μια ενέργεια ενώ αν έχει ψευδή συνθήκη εκτελείται η κενή ενέργεια. Η ενέργεια μετά το «*Parallel*» (αμφιθαλής) θα εκτελεστεί αφού έχουν εκτελεστεί όλα οι παράλληλες διακλαδώσεις είτε εκτελώντας μια ενέργεια είτε την κενή ενέργεια. Η λύση αυτή παρουσιάζεται στην **Εικόνα 3.7**. Η συνθετική αυτή υλοποίηση των προτύπων 6 και 7 δεν υποδηλώνει άμεση υποστήριξη τους XASC. Ο λόγος για την μη ύπαρξη ενός σύνθετου στοιχείου που θα αναπαριστούσε άμεσα το πρότυπο είναι η βασική απαίτηση του μοντέλου μας για στοιχεία δομημένα με *ξεκάθαρη* σημασιολογία που θα μας επιτρέπουν την παρακολούθηση της κατάστασης εκτέλεσης του διαγράμματος ροής οποιαδήποτε στιγμή.

Στην *BPEL4WS* η Πολλαπλή Επιλογή/ Συγχώνευση με Συγχρονισμό υλοποιείται ομοίως με πριν με την χρήση συνδέσμων ελέγχου με προσαρτημένες συνθήκες ελέγχου. Ακόμα και όταν μία συνθήκη ελέγχου παίρνει τιμή «ψευδής», αυτή η «άρνηση εκτέλεσης» διαδίδεται κατά μήκος των συνδέσμων που ακολουθούν με αποτέλεσμα να επιτυγχάνεται ο συγχρονισμός στην ενέργεια που έχει την συνθήκη ένωσης (join condition).

Στην *BPML* η Πολλαπλή Επιλογή/ Συγχώνευση με συγχρονισμό αναπαρίσταται με τεχνική παρόμοια με πριν, είτε με συνδυασμό των στοιχείων «switch» και «all» είτε με συνδυασμό των στοιχείων «raise» και «synch».




```
<flowchart_root>
  <sequence>
  <task name="A"/>
  <parallel>
    <choice condition="…">
      <true>
        <task name="B"/>
      </true>
      <false>
      <task type="empty"/>
      </false>
    </choice>
    …
  </parallel>
  <task name="E"/>
  </sequence>
</flowchart_root>
```

**Εικόνα 3.7 Πολλαπλή Επιλογή και Συγχώνευση με Συγχρονισμό στο AZTEC**

**Πρότυπο8 Πολλαπλή Συγχώνευση** (Multi-merge)

Είναι ένα σημείο στην διαδικασία όπου πολλές διακλαδώσεις συγκλίνουν σε μία ίνα εκτέλεσης (control thread) *χωρίς συγχρονισμό*. Αυτό σημαίνει ότι η ενέργεια που

ακολουθεί την συγχώνευση θα εκτελεστεί τόσες φορές όσες και οι διακλαδώσεις που ολοκληρώνουν την εκτέλεση τους. Αυτό το πρότυπο συναντάται συνήθως σε διαδικασίες όπου παράλληλες διακλαδώσεις έχουν το ίδιο τελείωμα. Αντί η κοινή αυτή ενέργεια (τελειώματος) να επαναληφθεί (replicate) σε κάθε διακλάδωση, χρησιμοποιείται η Πολλαπλή Συγχώνευση.

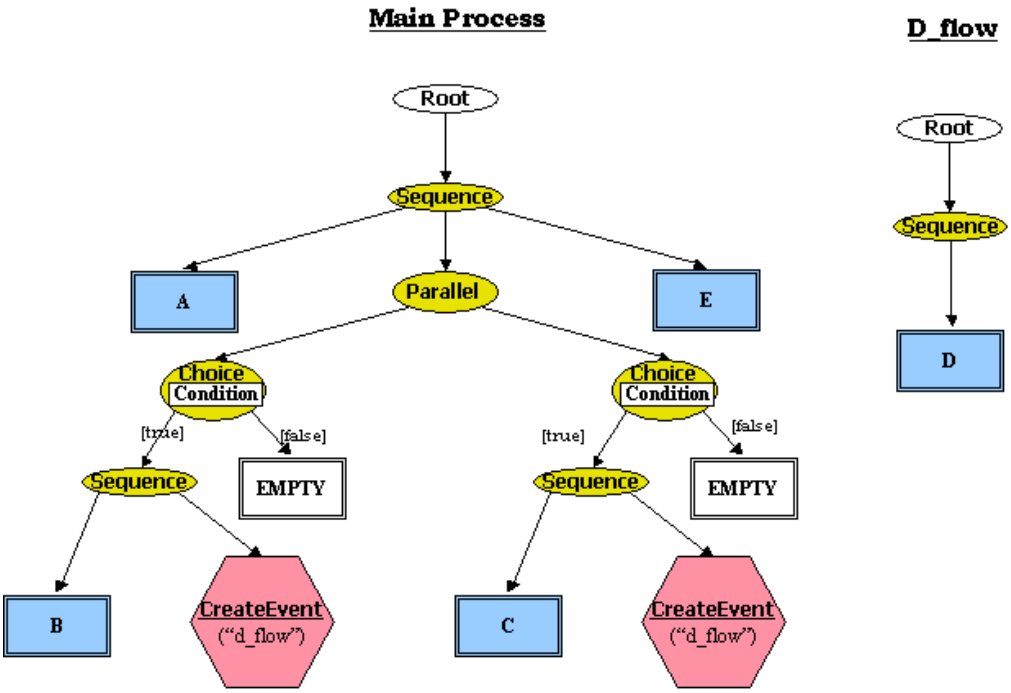
**Υλοποίηση Προτύπου 8 την XASC**

Στην XASC η Πολλαπλή Συγχώνευση μπορεί να υλοποιηθεί με δύο τρόπους. Ο πρώτος τρόπος χρησιμοποιεί την τεχνική που προτείνεται στο [ADH+02]. Καταρχήν χρειαζόμαστε δύο διαγράμματα ροής για αυτή την περίπτωση. Ένα για την «κύρια διαδικασία» (main process), και ένα που θα περιλαμβάνει μόνο την ενέργεια αυτή που δεν θέλουμε να επαναλάβουμε (replicate). Το κύριο διάγραμμα ροής θα περιλαμβάνει μία Parallel και δύο Choice μέσα σε αυτή. Η Choice θα περιλαμβάνει ακολουθιακά, μία διακλάδωση που μπορεί να είναι μία ενέργεια είτε ένα πολύπλοκο «υποδιάγραμμα ροής», και μία ενέργεια τύπου «*γεγονός*» (event task) η οποία πρακτικά δημιουργεί ένα στιγμιότυπο του διαγράμματος ροής που περιλαμβάνει την «κοινή» ενέργεια (το «κοινό τελείωμα»). Με αυτό τον τρόπο δημιουργούνται δύο ή περισσότερα στιγμιότυπα της ίδιας ενέργειας χωρίς πρακτικά να επαναληφθεί ο ορισμός της. Ο δεύτερος τρόπος υλοποίησης της Πολλαπλής Συγχώνευσης είναι με χρήση του δομικού στοιχείου Homogeneous Parallel (*Hparallel*) της XASC. Η σημασιολογία αυτού του στοιχείου είναι ότι το κομμάτι του διαγράμματος ροής εργασίας που «κρέμεται» από το «*Hparallel*», εκτελείται παράλληλα πολλές φορές, «παραμετροποιημένο» από μια ή περισσότερες λίστες με τιμές που παίρνει σαν παραμέτρους το «*Hparallel*». Σε αυτή την περίπτωση δημιουργούνται τόσα παράλληλα «παρακλάδια» όσα και το μέγεθος τής λίστας-παραμέτρου του «*Hparallel*». Κατά αυτό τον τρόπο το «*Hparallel*» θα περιλαμβάνει ακολουθιακά, ένα «*Switch*» που ανάλογα με την τιμή θα ενεργοποιεί το κατάλληλο παρακλάδι, και την ενέργεια που αποτελεί το «κοινό» τελικό σημείο όλων των παράλληλων διακλαδώσεων. Η λύση με την χρήση του «*Hparallel*» μπορεί να θεωρηθεί σχεδόν άμεση υποστήριξη του προτύπου από την XASC, μιας και ο «ρόλος» του στοιχείου «Hparallel» στο AZTEC είναι ακριβώς αυτός: Η «δήλωση» μία μόνο φορά, μίας ενέργειας που πρέπει να εκτελεστεί πολλές φορές παράλληλα. Οι δύο αυτοί δυνατοί τρόποι υλοποίησης φαίνονται στις **Εικόνες 3.8 α) και β)**

Στην BPEL4WS δεν μπορεί να αναπαρασταθεί η Πολλαπλή Συγχώνευση.

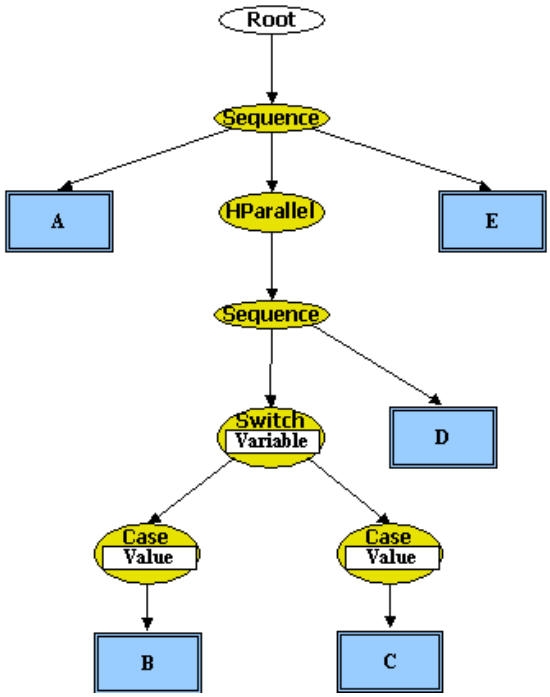
Στην BPML η Πολλαπλή Συγχώνευση μπορεί να αναπαρασταθεί με μια «all» και δύο «switch», και την ενέργεια που αποτελεί το «κοινό τελείωμα» να δημιουργείται στα πλαίσια μίας υποδιαδικασίας που καλείται και στις δύο «switch» (παρόμοια με την πρώτη

υλοποίηση που προτείνεται για το AZTEC δηλαδή). Το [ADH+02] την αποκαλεί αυτή «μερική λύση» γιατί η υπόλοιπη διαδικασία πρέπει να υλοποιηθεί σε ξεχωριστή διαδικασία περιορίζοντας έτσι την εφαρμογή αυτής της τεχνικής.




```
<flowchart_root name="main_process>
  <sequence>
    <task name="A"/>
    <parallel>
      <choice condition="…">
        <true>
          <sequence>
            <task name="B"/>
            <task name="d_flow" type="event"
mode="request-response"/>
          </sequence>
        </true>
      </choice>
      …
    </parallel>
    <task name="E"/>
  </sequence>
</flowchart_root>

<flowchart_root name="d_flow">
  <sequence>
    <task name="D"/>
  </sequence>
</flowchart_root>
```

**α)**




```
<flowchart_root>
  <sequence>
    <task name="A"/>
    <hparallel>
      <sequence>
        <switch condition="…">
        <case value="…">
          <task name="B"/>
        </case>
        …
        </switch>
        <task name="D"/>
      </sequence>
    </hparallel>
  </sequence>
</flowchart_root>
```

**β)**

**Εικόνα 3.8 Πολλαπλή Συγχώνευση στο AZTEC α) με χρήση Event Tasks και β)με χρήση του στοιχείου Hparallel**

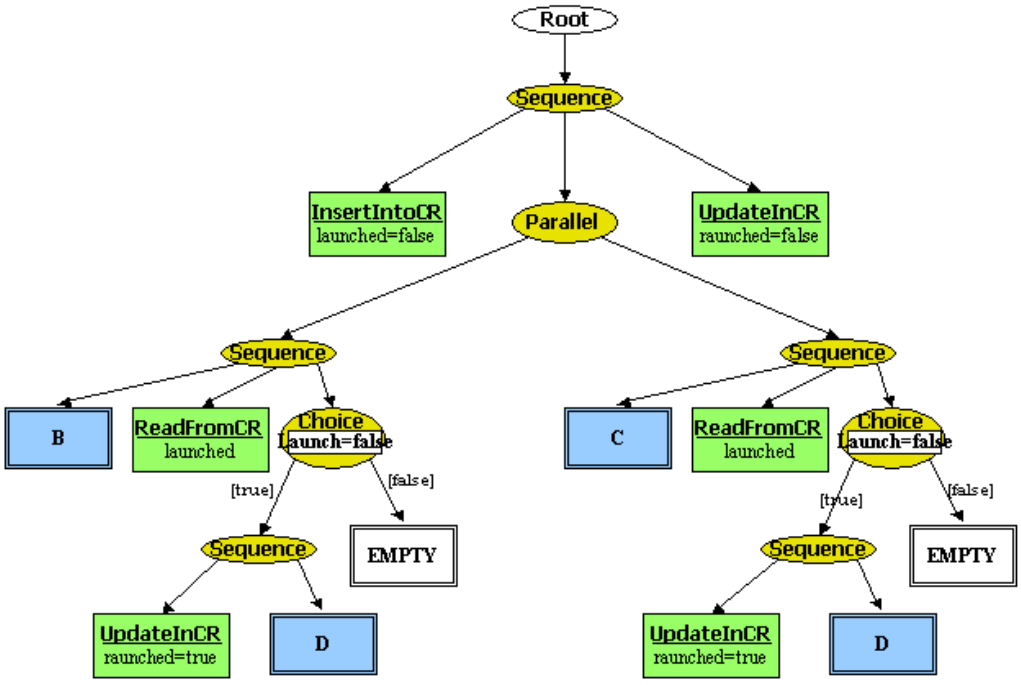
**Πρότυπο9 Διευκρινιτής** (Discriminator)

Είναι ένα σημείο στη διαδικασία Ροής Εργασίας που περιμένει το πρώτο από ένα πλήθος από εισερχόμενες διακλαδώσεις (branches) πριν εκτελέσει την επόμενη ενέργεια. Από αυτή την στιγμή και μετά, περιμένει όλες τις υπόλοιπες διακλαδώσεις να τελειώσουν προτού να μπορεί να ξαναχρησιμοποιηθεί (reset). Αυτή η λειτουργικότητα είναι απαραίτητη για να μπορεί το πρότυπο να χρησιμοποιηθεί στα πλαίσια ανακύκλωσης. Σαν παράδειγμα μπορούμε να σκεφτούμε την αποστολή της ίδιας επερώτησης σε πολλές βάσεις δεδομένων στο διαδίκτυο. Όταν φτάσει η πρώτη απάντηση από μία από όλες τις βάσεις, η εκτέλεση μπορεί να συνεχίσει αλλά για να μπορέσουμε να ξανακάνουμε επερώτηση πρέπει να έχουν απαντήσει στην προηγούμενη επερώτηση όλες οι βάσεις.

**Υλοποίηση Προτύπου 9 στην XASC.**

Η υλοποίηση του Διευκρινιτή στο AZTEC δεν είναι τετριμμένη. Τέτοιου είδος συγχρονισμός ανάμεσα σε παράλληλες διακλαδώσεις μπορεί να επιτευχθεί μόνο χρησιμοποιώντας λειτουργίες του Context Repository (CR). Το CR είναι μια εικονική αποθήκη δεδομένων με την μορφή εικονικού XML εγγράφου όπου μπορεί να γίνει διαμοιρασμός δεδομένων μεταξύ στοιχείων του ίδιου διαγράμματος ροής, ή και διαφορετικών διαγραμμάτων ροής στα πλαίσια της ίδιας διαδικασίας. Στην περίπτωση του

Διευκρινητή χρειαζόμαστε μία μεταβλητή (συγχρονισμού) δεδομένων στο CR. Η μεταβλητή αυτή θα ονομάζεται launched, λογικού (Boolean) τύπου η οποία πρακτικά όταν καμία διακλάδωση δεν έχει ολοκληρώσει την εκτέλεση της θα έχει τιμή «ψευδής». Κάθε διακλάδωση που θα ολοκληρώνει την εκτέλεση της θα κοιτάει την τιμή της launched στο CR. Η πρώτη που θα τελειώσει θα την βρει «ψευδή», θα την αλλάξει σε «αληθή» και θα εκτελέσει την επόμενη ενέργεια (Ενέργεια D στην **Εικόνα 3.9**). Οποιαδήποτε άλλη διακλάδωση τελειώσει θα ελέγξει την τιμή της launched, θα την βρει «αληθή» οπότε ξέροντας ότι κάποια άλλη διακλάδωση τελείωσε πρώτη, δεν θα εκτελέσει την επόμενη ενέργεια (την ενέργεια D). Από την στιγμή που έχουμε δομημένα διαγράμματα ροής (structured flowcharts), όταν ο έλεγχος φύγει από την Parallel αυτό σημαίνει ότι όλες οι παράλληλες διακλαδώσεις έχουν ολοκληρώσει την εκτέλεση τους οπότε ο διευκρινιτής μπορεί να «επανεκκινηθεί» (reset). Πρακτικά αυτό γίνεται θέτοντας την τιμή της launched στο CR «ψευδής». Ένα ακόμα μειονέκτημα αυτής της τεχνικής (πλην της πολυπλοκότητας της ) είναι ότι απαιτεί επανάληψη (replication) της ενέργειας που ακολουθεί σε όλα τα παράλληλα παρακλάδια, εκτός και αν αυτή τοποθετηθεί στα πλαίσια διαφορετικού διαγράμματος ροής όπως έγινε στην περίπτωση της Πολλαπλής Συγχώνευσης. Αυτή η λύση δεν μπορεί να υπονοήσει άμεση υποστήριξη του προτύπου «Διευκρινιτής» από το AZTEC. Το διάγραμμα ροής που δείχνει την αναπαράσταση του Διευκρινητή στο AZTEC φαίνεται στην **Εικόνα 3.9**.



```
<flowchart_root>
   <sequence>
      <task name="ctx1" type="repository" action="insert">
        <task_input_params>
          <in_param>
            <value_string>some_xpath</value_string>
          </in_param>
          <in_param>
            <value_string>launched</value_string>
          </in_param>
          <in_param><value_boolean>false</value_Boolean></in_param>
        </task_input_params>
      </task>
      <parallel>
      …
      <task name="ctx2" type="repository" action="get">
        <task_input_params>
          <in_param>
              <value_string>some_xpath</value_string>
          </in_param>
        <task_input_params>
        <task_return_params>
          <return_param>
            <value_bool>launched</value_bool>
          </return_param>
        </task_return_params>
      </task>
      </parallel>
      …
   </sequence>
</flowchart_root>
```

**Εικόνα 3.9 Ο Διευκρινητής στο AZTEC**

Το XML κομμάτι της **Εικόνας 3.9** ήταν κυρίως για να απεικονίσει την σύνταξη μίας ενέργειας του CR, παρά για να εκφράσει επακριβώς το διάγραμμα ροής.

Στην *BPEL4WS* δεν μπορεί να αναπαρασταθεί ο διευκρινιτής.

Στην *BPML* ο διευκρινιτής μπορεί, να αναπαρασταθεί με χρήση των στοιχείων «spawn», «synch» και «raise» χωρίς όμως να θεωρείται ότι υποστηρίζει άμεσα το πρότυπο.

**Πρότυπο10 Βρόγχοι** (arbitrary cycles)

Ένα τμήμα της διαδικασίας (περιλαμβάνοντας μία απλή ή περισσότερες δομημένες ενέργειες) πρέπει να εκτελεστεί επαναλαμβανόμενα χωρίς να τεθούν περιορισμοί ούτε στο πλήθος, ούτε στην «τοποθεσία» ούτε στην εμφώλευση αυτών των ενεργειών.

**Υλοποίηση Προτύπου 10 στην XASC**

Το *AZTEC* παρέχει ένα μοντέλο δομημένων διαγραμμάτων ροής (structured flowcharts). Εγγενώς λοιπόν μη δομημένοι κύκλοι δεν υποστηρίζονται. Για την δημιουργία δομημένων κύκλων η *XASC* περιλαμβάνει το στοιχείο «*While_do*». Το [AHK+02] προτείνει κάποιες τεχνικές για την μετατροπή μη δομημένων κύκλων σε δομημένους. Μόνο έτσι θα μπορούσε να αναπαρασταθεί ένας μη δομημένος κύκλος στο AZTEC.

Η *BPEL4WS* δεν μπορεί να αναπαραστήσει μη δομημένους κύκλους. Οι «σύνδεσμοι ελέγχου» της δεν μπορούν να «διασχίσουν» (cross) τα όρια ενός δομημένου κύκλου ορισμένου με ένα στοιχείο while.

Παρομοίως και η *BPML* δεν υποστηρίζει, ούτε μπορεί να αναπαραστήσει μη δομημένους κύκλους.

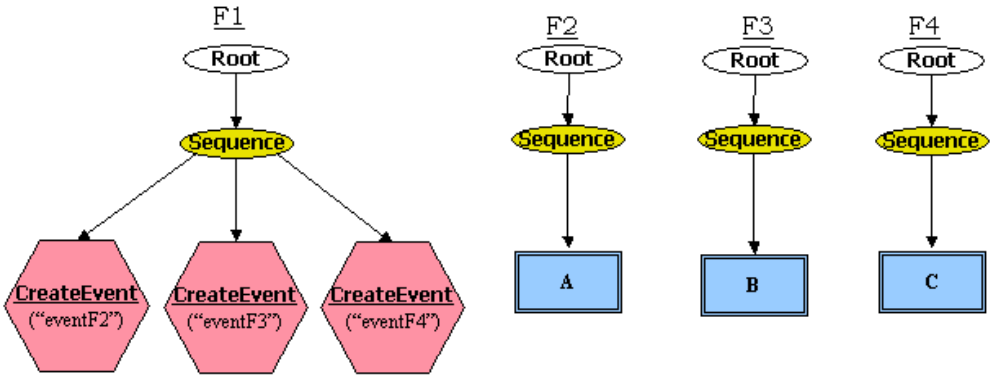
**Πρότυπο11 Έμμεσος Τερματισμός** (Implicit Termination)

Μια υποδιαδικασία μπορεί να τερματιστεί όταν δεν υπάρχει καμία άλλη ενέργεια να εκτελεστεί (δηλαδή δεν απαιτείται η εκτέλεση κάποιας συγκεκριμένης ενέργειας τέλους για να τερματιστεί η υποδιαδικασία).

**Υλοποίηση Προτύπου 11 στην XASC**

Οποιαδήποτε δομημένη ενέργεια ολοκληρώνεται όταν εκτελεστεί η «υψηλότερου επιπέδου» ενέργεια της [WAD+01]. Για παράδειγμα η εκτέλεση ενός διαγράμματος ροής τερματίζει όταν ολοκληρωθεί η εκτέλεση του στοιχείου flowchart_root υπονοώντας *ρητό* (explicit) τερματισμό κατά αυτό τον τρόπο. Χρησιμοποιώντας ιδέες από το [ADH+02] ο μόνος τρόπος για να επιτευχθεί έμμεσος τερματισμός στο AZTEC είναι με την εκτέλεση ενεργειών γεγονότων (Event Tasks). Όταν εκτελείται μία ενέργεια γεγονός, δημιουργείται ένα γεγονός το οποίο προκαλεί την ενεργοποίηση ενός νέου διαγράμματος ροής το οποίο δεν «συγχρονίζεται» (τουλάχιστον όχι απαραίτητα) με το «αρχικό» διάγραμμα ροής, δηλαδή το «αρχικό» διάγραμμα ροής δεν μπλοκάρει περιμένοντας το νέο διάγραμμα ροής να ολοκληρώσει την εκτέλεση του για να συνεχίσει.. Έτσι από ένα ενεργό διάγραμμα ροής μπορούν να ενεργοποιηθούν πολλά διαγράμματα ροής τα οποία να εκτελούνται ανεξάρτητα χωρίς αλληλεπιδράσεις. Με αυτό τον τρόπο ένα διάγραμμα ροής μπορεί να έχει πολλά σημεία τερματισμού, τα σημεία τερματισμού των διάφορων διαγραμμάτων ροής. Τα παραπάνω μπορούν να γίνουν πιο κατανοητά βλέποντας το διάγραμμα ροής της **Εικόνας 3.10**. Από το διάγραμμα ροής «F1» εκτελούνται οι ενέργειες γεγονότων «eventF2», «eventF3» και «eventF4» οι οποίες προκαλούν την ενεργοποίηση των διαγραμμάτων ροής F2, F3 και F4 αντίστοιχα. Τα σημεία τερματισμού των F2,F3 και F4 μπορούν πρακτικά να θεωρηθούν ως τα διάφορα σημεία τερματισμού του διαγράμματος ροής F1.

Η BPEL4WS υλοποιεί τον έμμεσο τερματισμό με χρήση των συνδέσμων ελέγχου της. Παρόλο που τα δομημένα στοιχεία («sequence», «all» κ.λ.π) υπονοούν ρητό τερματισμό, στα πλαίσια ενός στοιχείου flow μπορούν να δημιουργηθούν πολλαπλές ενέργειες η οποίες να είναι τελικές, δηλαδή να μην ξεκινάει κανένας σύνδεσμος




**Εικόνα 3.10 Παράδειγμα διαγράμματος ροής με πολλαπλά σημεία τερματισμού**

ελέγχου από αυτές. Έχοντας πολλαπλές τελικές ενέργειες υπονοείται έμμεσος τερματισμός.

Η BPML υλοποιεί τον έμμεσο τερματισμό με την δημιουργία υποδιαδικασιών χωρίς συγχρονισμό χρησιμοποιώντας το στοιχείο «spawn» (όπως δηλαδή και το AZTEC).

**Πρότυπο12 Πολλαπλά Στιγμιότυπα χωρίς Συγχρονισμό** (MI without Synchronization)

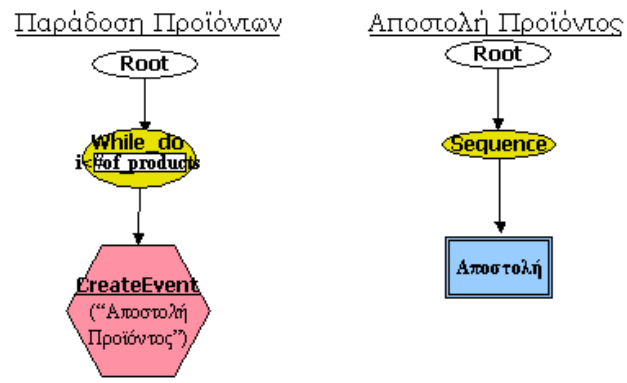
Στα πλαίσια ενός στιγμιοτύπου μίας Ροής Εργασίας μπορούν να δημιουργηθούν πολλά στιγμιότυπα της ίδιας ενέργειας (δηλαδή πρακτικά υπάρχει η δυνατότητα για δημιουργία πολλαπλών ανεξάρτητων «ινών εκτέλεσης» (control threads). Αυτές οι ίνες εκτέλεσης δεν χρειάζεται να συγχρονιστούν αφού τελειώσουν την εκτέλεση τους. Για παράδειγμα μία διαδικασία ηλεκτρονικού εμπορίου που εξυπηρετεί μία παραγγελία με πολλά προϊόντα εκτελεί την ενέργεια «αποστολή_προϊόντος» μία φορά για κάθε προϊόν.

**Υλοποίηση Προτύπου 12 στην XASC**

Στην XASC το πρότυπο αυτό υλοποιείται δημιουργώντας πολλαπλές «ενέργειες γεγονότων» (Event Tasks) στα πλαίσια ενός «*Parallel*» ή ενός «*While_do*». Τα διαγράμματα ροής που θα εκτελεστούν για τις διάφορες «ενέργειες γεγονότων» θα εκτελεστούν ανεξάρτητα μεταξύ τους όπως βλέπομε στην **Εικόνα 3.11**.

Στην BPEL4WS το πρότυπο αυτό υλοποιείται με το στοιχείο «invoke» στα πλαίσια ενός στοιχείου while.

Στην BPML το πρότυπο αυτό υλοποιείται με το στοιχείο «spawn» στα πλαίσια ενός στοιχείου «while», «until» ή «foreach».




```
<flowchart_root name="product_delivery">
   …
 <while_do condition="i &lt num_of_products">
   <task name="…" type="event" action="product_dispatch"
mode="notification"/>
   <task name="incrI" type="internal" action="i++"/>
 </while_do>
 …
</flowchart_root>

<flowchart_root name="product_dispatch">
     <sequence>
          <task name="dispatching"…/>
     </sequence>
</flowchart_root>
```

**Εικόνα 3.11 Δημιουργία πολλών στιγμιότυπων της ενέργειας «αποστολή προϊόντος» χωρίς συγχρονισμό.**

**Πρότυπα13 – 15 Πολλαπλά Στιγμιότυπα με Συγχρονισμό** (MI with Synchronization)

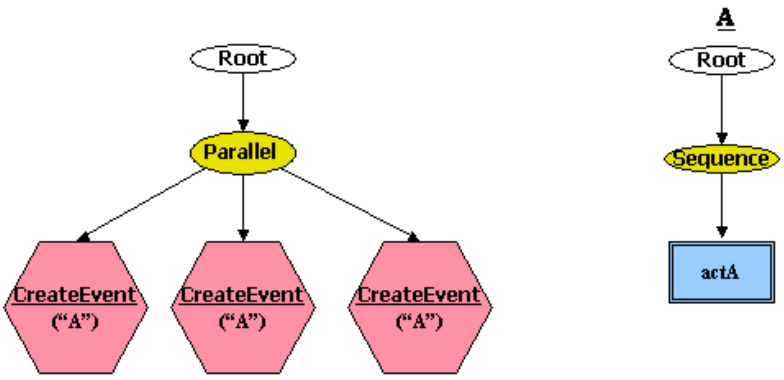
Είναι ένα σημείο στην Ροή Εργασίας όπου πολλά στιγμιότυπα μίας ενέργειας δημιουργούνται, και αφού ολοκληρωθεί η (παράλληλη) εκτέλεση τους, συγχρονίζονται για να συνεχιστεί η εκτέλεση της ροής εργασίας. Υπάρχουν τρεις περιπτώσεις προτύπων που απαιτούν συγχρονισμό. Στο πρότυπο 13 ο αριθμός των στιγμιότυπων που πρέπει να δημιουργηθούν είναι γνωστός από την φάση της σχεδίασης της Ροής Εργασίας. Στο πρότυπο 14 ο αριθμός των στιγμιότυπων που πρέπει να δημιουργηθούν γίνεται γνωστός σε κάποια φάση της εκτέλεσης αλλά πριν την δημιουργία των στιγμιότυπων. Στο πρότυπο 15 ο αριθμός των στιγμιότυπων δεν είναι γνωστός εξαρχής αλλά μετά την δημιουργία κάθε στιγμιότυπου, ο χρήστης με κάποιο μήνυμα καθορίζει αν επιθυμεί την δημιουργία ενός νέου στιγμιότυπου. Ένα παράδειγμα είναι όταν ένας πελάτης κανονίζει ένα ταξίδι, όπου κάθε φορά που δημιουργείται ένα στιγμιότυπο της ενέργειας «κράτηση_πτήσης» ο πελάτης αποφασίζει αν θέλει να κλείσει και κάποια άλλη πτήση ή αν έχει ολοκληρώσει τις κρατήσεις του, οπότε και του στέλνεται το συνολικό τιμολόγιο.

**Υλοποίηση Προτύπων 13, 14 και 15 στην XASC**

Στο *AZTEC* για το πρότυπο 13, μία απλή λύση είναι η επανάληψη (replication) της ενέργειας της οποίας πολλά στιγμιότυπα είναι επιθυμητά, στα πλαίσια ενός στοιχείου «*Parallel*». Όταν όλα τα στιγμιότυπα ολοκληρώσουν την εκτέλεση τους θα συγχρονιστούν για να συνεχίσει η διαδικασία την εκτέλεση της (**Εικόνα 3.12**). Για το πρότυπο 14, λύση δίνει το στοιχείο «*Hparallel*». Πράγματι με χρήση του «*Hparallel*» μπορεί να δημιουργηθεί ένα σύνολο από στιγμιότυπα (με μεταβλητό πλήθος) που θα εκτελεστούν παράλληλα, και θα συγχρονιστούν όταν όλα θα έχουν τελειώσει την εκτέλεση τούς (**Εικόνα 3.13**). Το πλήθος των στιγμιότυπων που θα δημιουργηθεί μπορεί να καθοριστεί σε οποιαδήποτε στιγμή της εκτέλεσης της διαδικασίας, πριν ξεκινήσει η δημιουργία των στιγμιότυπων. Το πρότυπο 15 δεν μπορεί να αναπαρασταθεί από το *AZTEC*, γιατί απαιτεί την ύπαρξη κάποιου δομημένου στοιχείου (structured construct) που να «μπλοκάρει» και να περιμένει την άφιξη ενός μηνύματος ή γεγονότος. Η τοποθέτηση ενός τέτοιου στοιχείου στα πλαίσια ενός στοιχείου «*While_do*», θα παρείχε μία αντιμετώπιση του προβλήματος. Η δημιουργία ενός τέτοιου στοιχείου είναι μία από τις μελλοντικές επεκτάσεις της γλώσσας XASC. Για την ακρίβεια δεν θα δημιουργηθεί ένα νέο στοιχείο αλλά η επιθυμητή λειτουργικότητα θα προστεθεί στην σημασιολογία του υπάρχοντος στοιχείου «*Switch*» της XASC. Η σημασιολογία του στοιχείου «*Switch*» είναι η επιλογή μίας διακλάδωσης ανάμεσα σε πολλαπλές βάσει κάποιων δεδομένων ελέγχου (control data). Η σημασιολογία αυτή θα εμπλουτιστεί ούτως ώστε να περιλαμβάνει την επιλογή μίας διακλάδωσης ανάμεσα σε πολλαπλές βάσει είτε κάποιων δεδομένων ελέγχου, είτε κάποιου γεγονότος.

Στην *BPEL4WS* το πρότυπο 13 αναπαρίσταται με επανάληψη (replication) της ενέργειας τόσες φορές όσες χρειάζεται να εκτελεστεί στα πλαίσια του στοιχείου «flow». Για τα πρότυπα 14 και 15 προτείνεται στο [WAD+01] μία λύση με ένα στοιχείο «pick» (ανάλογα με το εισερχόμενο μήνυμα εκτελεί κάποιες ενέργειες) μέσα σε μία «while» καταμετρώντας πόσα στιγμιότυπα έχουν ξεκινήσει, και πόσα έχουν τελειώσει την εκτέλεση τους.

Στην *BPML* το πρότυπο 13 αναπαρίσταται ομοίως με επανάληψη της ενέργειας τόσες φορές όσες χρειάζεται να εκτελεστεί στα πλαίσια του στοιχείου «all». Τα πρότυπα 14 και 15 αναπαρίστανται με παρόμοια τεχνική όπως αυτή της BPEL4WS με χρήση των στοιχείων «raise» και «synch» στα πλαίσια μίας «while», και πάλι καταμετρώντας τον αριθμό των στιγμιότυπων που έχουν ξεκινήσει και που έχουν τελειώσει.




```
<flowchart_root name="">
   …
 <parallel>
   <task name="…" type="event" action="A"
mode="request_response"/>
 </parallel>
 …
</flowchart_root>

<flowchart_root name="A">
 <sequence>
   <task name="actA"…/>
 </sequence>

</flowchart_root>
```

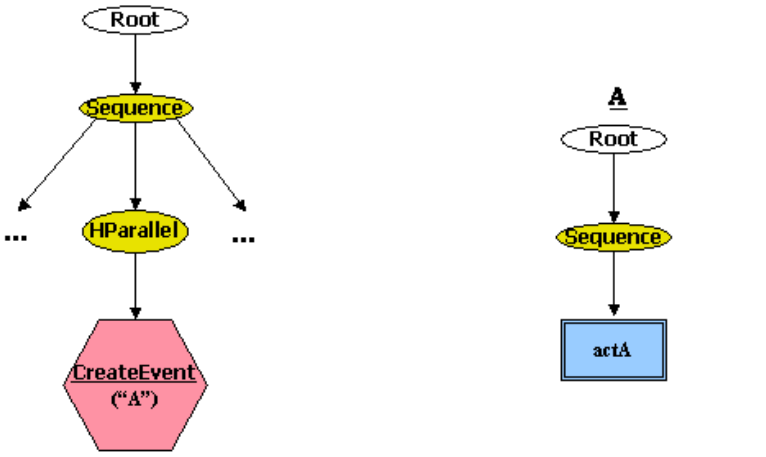
**Εικόνα 3.12 Δημιουργία πολλών στιγμιοτύπων(γνωστού πλήθους εξαρχής) της ενέργειας Α, με συγχρονισμό**

**Πρότυπο16 Επιλογή με Καθυστέρηση** (Deferred Choice)

Είναι ένα σημείο στην διαδικασία όπου μία διακλάδωση πρέπει να επιλεγεί, ανάμεσα σε πολλαπλές πιθανές διακλαδώσεις. Η διαφορά αυτού του προτύπου με την Αποκλειστική Επιλογή (Exclusive Choice) είναι ότι η επιλογή δεν γίνεται βάσει κάποιων δεδομένων ελέγχου (control data). Αντίθετα η επιλογή της διακλάδωσης καθυστερείται έως την λήψη ενός μηνύματος ή γεγονότος που θα καθορίσει την επιλογή της διακλάδωσης. Μερικές φορές συναντάται και ως ενδεχόμενη επιλογή (implicit choice). Για παράδειγμα η ενέργεια «αποστολή_ερωτηματολογίου» μπορεί να ακολουθείται είτε από την ενέργεια «λήψη_ερωτηματολογίου» αν ο ενδιαφερόμενος το συμπλήρωσε και το έστειλε έγκαιρα, ή «τέλος διορίας» αν έχει λήξει η διορία και τυχόν αποστολή ερωτηματολογίου δεν θα γίνει πλέον δεκτή.

**Υλοποίηση Προτύπου 16 στην XASC**

Στο AZTEC καταστάσεις σαν αυτή δεν αντιμετωπίζονται με κάποιο συγκεκριμένο δομικό στοιχείο (construct). Η «φύση» του AZTEC, δηλαδή η απόκριση σε συγκεκριμένα



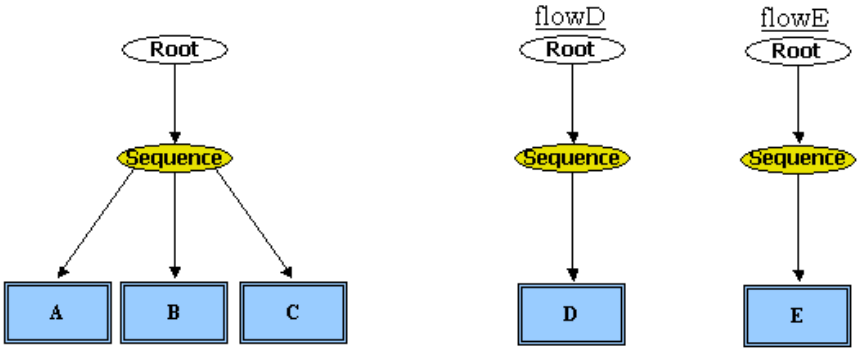

```
<flowchart_root name="">
   …
 <hparallel>
   <task_input_params>
     <in_param name="set1">
        <value_set>set1</value_set>
     </in_param>
   </task_input_params>
   <task name="…" type="event" action="A"
mode="request_response"/>
 </hparallel>
 …
</flowchart_root>

<flowchart_root name="A">
 <sequence>
   <task name="actA"…/>
 </sequence>
</flowchart_root>
```

**Εικόνα 3.13 Δημιουργία πολλών στιγμιοτύπων (πλήθους γνωστού κατά την διάρκεια της εκτέλεσης) της ενέργειας A, με συγχρονισμό**

γεγονότα προκαλώντας την εκτέλεση συγκεκριμένων διαγραμμάτων ροής αντιμετωπίζει απαιτήσεις σαν αυτή της επιλογής με καθυστέρηση. Σαν απλοϊστικό παράδειγμα, αν θέλαμε να εκτελεστούν τρεις ενέργειες σειριακά, οι A,B,C και έπειτα μία εκ των D,E ανάλογα με την λήψη του σχετικού γεγονότος, το σχήμα της διαδικασίας (process schema) θα περιείχε τα διαγράμματα ροής της **Εικόνας 3.14**. Ένα πρόβλημα που μπορεί να προκύψει με αυτή την αναπαράσταση είναι ότι αν συμβεί ένα από τα γεγονότα που προκαλεί την ενεργοποίηση των flowD ή flowE πριν εκτελεστούν οι πρώτες τρεις ενέργειες (A,B,C), η ενέργεια D ή E θα εκτελεστεί παρόλαυτα. Επομένως για να υλοποιηθεί σωστά το πρότυπο πιθανώς να χρειάζεται κάποιο συγχρονισμό μεταξύ των διαγραμμάτων ροής της ίδιας διαδικασίας με χρήση ενεργειών του CR. Η επέκταση που

προτάθηκε νωρίτερα για το στοιχείο «*Switch*» θα προσφέρει μια εναλλακτική υλοποίηση και για αυτό το πρότυπο.

Τόσο στην BPEL4WS όσο και στην BPML υπάρχει ένα δομημένο στοιχείο το οποίο «μπλοκάρει» και περιμένει συγκεκριμένα μηνύματα αντιστοιχίζοντας τα σε συγκεκριμένες ενέργειες. Στην BPEL4WS το στοιχείο ονομάζεται «pick» ενώ στην BPML «choice».




**Εικόνας 3.14 Επιλογή με Καθυστέρηση στο AZTEC**

**Πρότυπο17 Μη Διατεταγμένη Ακολουθία** (Unordered Sequence)

Είναι ένα σύνολο από ενέργειες οι οποίες εκτελούνται με οποιαδήποτε σειρά. Κάθε ενέργεια εκτελείται μόνο μία φορά. Η σειρά εκτέλεσης των ενεργειών μπορεί να είναι είτε τυχαία είτε να αποφασίζεται την ώρα της εκτέλεσης. Όταν ολοκληρώνεται η εκτέλεση μίας ενέργειας γίνεται γνωστό ποια θα είναι η επόμενη που θα εκτελεστεί. Κάθε στιγμή μόνο μία ενέργεια μπορεί να είναι ενεργή δηλαδή δεν υπάρχει παράλληλη εκτέλεση ενεργειών. Για παράδειγμα κάθε τράπεζα στο τέλος κάθε χρονιάς για κάθε λογαριασμό εκτελεί δύο ενέργειες, την «προσθήκη_τόκου» και την «χρέωση_πιστωτικής_κάρτας». Οι ενέργειες αυτές μπορούν να εκτελεστούν με οποιαδήποτε σειρά, απλά δεν πρέπει να είναι ενεργές ταυτόχρονα.

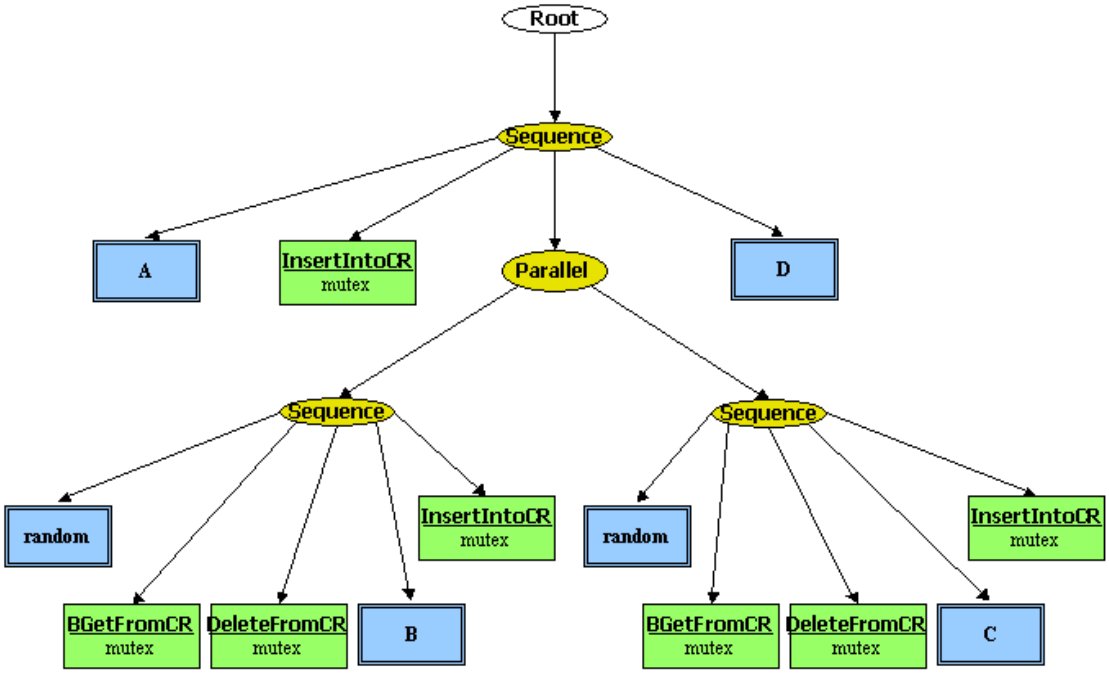
**Υλοποίηση Προτύπου 17 στην XASC**

Στο AZTEC ο μόνος τρόπος να αναπαρασταθεί η μη διατεταγμένη ακολουθία είναι με συνδυασμό του στοιχείου Parallel, ενεργειών του CR, και κάποιες «τυχαίας (ή εξαρτούμενης από το περιβάλλον) διάρκειας» ενέργειας. Οι ενέργειες που θέλουμε να εκτελεστούν μη διατεταγμένα, τοποθετούνται σαν διακλαδώσεις ενός Parallel μετά από μία ενέργεια που διαρκεί «τυχαίο» χρονικό διάστημα, και μετά από μία blocking ανάγνωση (CR blocking get) στο CR μίας μεταβλητής αμοιβαίου αποκλεισμού (mutex). Αυτή η ενέργεια έχει το εξής νόημα. Η πρώτη ενέργεια που θα καταφέρει να εκτελεστεί (βάσει της τυχαίας καθυστέρησης) θα σβήσει (CR delete) από το CR την μεταβλητή

mutex. Οι υπόλοιπες προσπαθώντας να διαβάσουν από το CR την mutex με blocking τρόπο (που σημαίνει ότι αν δεν υπάρχει η mutex, μπλοκάρουν μέχρι να δημιουργηθεί και μετά ξεμπλοκάρουν) μπλοκάρουν περιμένοντας την «πρώτη» ενέργεια να εκτελεστεί. Αφού η πρώτη ενέργεια ολοκληρώσει την εκτέλεση της, δημιουργεί στο CR (CR insert) την μεταβλητή mutex, οπότε η δεύτερη «γρηγορότερη» ενέργεια εκτελείται κ.λ.π Φυσικά αυτή η λύση που βλέπουμε στην **Εικόνας 3.15** έχει πολλά μειονεκτήματα. Πρώτον, η σειρά εκτέλεσης είναι μάλλον πιο πολύ τυχαία παρά εξαρτούμενη από το περιβάλλον. Για αυτό το πρόβλημα το [AHK+02] προτείνει μία τεχνική παρόμοια με αυτή της επιλογής με καθυστέρηση. Στην περίπτωση μας θα χρειαστούν και λειτουργίες του CR για να διασφαλίσουν ότι κάθε ενέργεια θα εκτελεστεί μία μόνο φορά αλλά και ότι δεν θα έχουμε παράλληλη εκτέλεση. Γενικά αυτή η τεχνική μπορεί να οδηγήσει σε πολύ περίπλοκες λύσεις. Δεύτερον πιθανότατα θα δημιουργηθεί πρόβλημα συγχρονισμού, όταν υπάρχουν δύο ή περισσότερες blocking get που περιμένουν για την ίδια μεταβλητή. Όταν αυτή η μεταβλητή δημιουργηθεί, δεν μπορούμε να ξέρουμε ποια από τις δύο blocking get θα την διαβάσει πρώτη, ή αν το καταφέρουν και οι δύο (κάτι που είναι προφανώς ανεπιθύμητο). Γενικά δεν μπορεί να θεωρηθεί ότι το AZTEC υποστηρίζει άμεσα αυτό το πρότυπο.

Η *BPEL4WS* χρησιμοποιεί την έννοια των «serializable scopes» για να αναπαραστήσει την μη διατεταγμένη ακολουθία. Ένα «serializable scope» είναι μία ενέργεια τύπου «scope» με το χαρακτηριστικό (attribute) «containerAccessSerializable» να έχει την τιμή «yes» εγγυούμενο έτσι τον έλεγχο ταυτόχρονης πρόσβασης (concurrency control) σε διαμοιραζόμενους (shared) «containers». Πρακτικά αυτό που γίνεται είναι ότι υπάρχει ένας «container» στον οποίο κάθε «serializable scope» «γράφει» πριν εκτελεστεί η ενέργεια του. Από την στιγμή που μόνο μία εγγραφή επιτρέπεται κάθε στιγμή σε ένα «container», υλοποιείται η «μη διάταξη» με τυχαίο τρόπο. Η αντιμετώπιση αυτή χρησιμοποιεί την ίδια τεχνική με αυτή που χρησιμοποιήσαμε νωρίτερα στο *AZTEC*. Για να υλοποιήσει την περίπτωση η σειρά να μήν επιλέγεται τυχαία, αλλά κατόπιν επιλογής (on demand) η *BPEL4WS* μπορεί να χρησιμοποιήσει το στοιχείο pick (ala deferred choice) οδηγώντας έτσι όμως σε πολύ περίπλοκες λύσεις.

Η *BPML* μπορεί με παρόμοιες τακτικές στα δικά της πλαίσια να παρέχει παρόμοιες «μερικές» (partial) λύσεις για την μη διατεταγμένη ακολουθία.




**Εικόνας 3.15 Μη Διατεταγμένη Ακολουθία στο AZTEC**

**Πρότυπο18 Ορόσημο** (milestone)

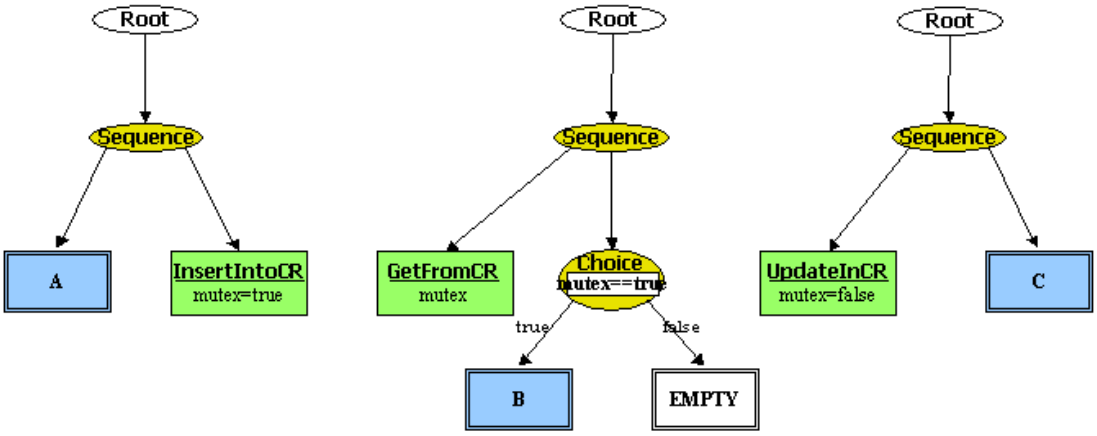
Μία ενέργεια μπορεί να εκτελεστεί αν η εκτέλεση έχει φτάσει σε ένα σημείο ορόσημο. Ένα ορόσημο είναι ένα σημείο στην διαδικασία όπου μία ενέργεια έχει ολοκληρώσει την εκτέλεση της και μία άλλη ενέργεια δεν έχει ξεκινήσει ακόμα την εκτέλεση της. Για παράδειγμα από την στιγμή που κάποιος κάνει μία κράτηση θέσης σε μία πτήση, μπορεί να ακυρώσει την κράτηση έως ότου μία διορία φτάσει (και φυσικά μετά που έχει ολοκληρώσει την κράτηση).

**Υλοποίηση Προτύπου 18 στην XASC**

Τέτοιου είδους συγχρονισμός επιτυγχάνεται στο AZTEC με χρήση του CR σαν μέρος αμοιβαίου αποκλεισμού (mutual exclusion place). Αν οι ενέργειες A και C «ορίζουν» το ορόσημο, και η ενέργεια B είναι αυτή που μπορεί να εκτελεστεί μόνο τότε, τότε με το τέλος της A η λογική μεταβλητή mutex παίρνει την τιμή «αληθής» και ακριβώς πριν την εκτέλεση της C παίρνει την τιμή «ψευδής». Η μεταβλητή mutex ελέγχεται κάθε φορά που είναι επιθυμητή η εκτέλεση της ενέργειας B, και μόνο αν είναι «αληθής» εκτελείται όντως η B. Το σχήμα αυτής της διαδικασίας (process schema) φαίνεται στην **Εικόνας 3.16**

Στην *BPEL4WS* μία συνθετική λύση (work-around solution) μπορεί να δοθεί με την χρήση του στοιχείου «pick» μέσα σε ένα «while». Όταν φτάσει το μήνυμα που ορίζει το τέλος του ορόσημου, η συνθήκη της «while» γίνεται ψευδής.

Στην *BPML* μια παρόμοια λύση μπορεί να δοθεί, με μία υποδιαδικασία που δημιουργεί μηνύματα «αρχή_ορόσημου» και «τέλος_ορόσημου» και μία παράλληλα εκτελούμενη διαδικασία η οποία εκτελεί την εν λόγω ενέργεια στα πλαίσια ενός στοιχείου «choice».




**Εικόνα 3.16 Το Σχήμα μίας διαδικασίας που υλοποιεί το πρότυπο ορόσημο**

**Πρότυπο19 Ακύρωση Ενέργειας** (Cancel Task)

Είναι η ακύρωση ενός ενεργού στιγμιότυπου μίας ενέργειας.

**Υλοποίηση Προτύπου 19 στην XASC**

Η ακύρωση της εκτέλεσης μίας συγκεκριμένης ενέργειας δεν υποστηρίζεται από το *AZTEC*. Αυτό που υποστηρίζεται είναι η ακύρωση της εκτέλεσης όλων των ενεργών διαγραμμάτων ροής ενός συγκεκριμένου τύπου.

Στην *BPEL4WS* η ακύρωση ενεργειών υλοποιείται με την χρήση χειριστών λαθών και αντισταθμίσεων (fault και compensation handlers).

Στην *BPML* η ακύρωση ενεργειών υλοποιείται με την μορφή εξαιρέσεων (exceptions).

**Πρότυπο20 Ακύρωση Διαδικασίας** (Cancel Case)

Είναι η ακύρωση ενός στιγμιότυπου μίας διαδικασίας (process).

**Υλοποίηση Προτύπου 20 στην XASC**

Η ακύρωση ενός στιγμιότυπου μίας διαδικασίας στην XASC υλοποιείται με το στοιχείο «TerminateProcess».

Η *BPEL4WS* υλοποιεί την ακύρωση της εκτέλεσης μίας διαδικασίας με χρήση του στοιχείου «terminate».

Η *BPML* υλοποιεί την ακύρωση της εκτέλεσης μίας διαδικασίας με την μορφή μίας εξαίρεσης (exception).

## *3.3 Πρότυπα Επικοινωνίας Ροών Εργασίας και η XASC*

Σε αυτή την ενότητα θα αξιολογήσουμε το μοντέλο ενεργών διαγραμμάτων ροής του AZTEC και την γλώσσα XASC σε σχέση με τα πρότυπα επικοινωνίας που παρουσιάστηκαν στο [RMB01]. Εδώ γίνεται η παραδοχή ότι η επικοινωνία πραγματοποιείται με την ανταλλαγή μηνυμάτων μεταξύ διαφορετικών διαδικασιών, οπότε μοντελοποιείται αποκλειστικά με τις έννοιες της αποστολής και λήψης μηνυμάτων. Δύο τύποι επικοινωνίας μπορούν να αναγνωρισθούν, η σύγχρονη (synchronous) και η ασύγχρονη (asynchronous) επικοινωνία. Να σημειωθεί ότι η σημασιολογία των προτύπων που θα περιγραφεί παρακάτω δεν είναι απαραίτητα η ίδια με τα είδη αλληλεπίδρασης (modes of interaction) που ορίζει η WSDL παρόλο που οι ονομασίες πολλές φορές θα είναι παρεμφερείς.

### 3.3.1 Σύγχρονη Επικοινωνία

Η σύγχρονη επικοινωνία περιγράφει την κατάσταση στην οποία ο «αποστολέας» και ο «παραλήπτης» *συντονίζουν* την εκτέλεση της διαδικασίας τους ανάλογα με την επικοινωνία τους. Αυτού του είδους η επικοινωνία προτιμάται είτε όταν ο «αποστολέας» «χρειάζεται» δεδομένα από την εκτέλεση της διαδικασίας του «παραλήπτη» είτε όταν «χρειάζεται» επιβεβαίωση της λήψης του μηνύματος από τον παραλήπτη για να συνεχίσει την εκτέλεση του.

**Πρότυπο 1 Αίτηση/Απάντηση** (Request/Reply)

Η επικοινωνία με Αίτηση/Απάντηση είναι μία μορφή σύγχρονης επικοινωνίας στην οποία ένας «αποστολέας» στέλνει μία αίτηση (request) σε ένα «παραλήπτη» και περιμένει για μία απάντηση προτού συνεχίσει την εκτέλεση της διαδικασίας του. Η λήψη της απάντησης πιθανώς να επηρεάσει την εξέλιξη της εκτέλεσης στην πλευρά του αποστολέα.

**Πρότυπο 2 Μονόδρομος** (One-Way)

Είναι μία μορφή σύγχρονης επικοινωνίας όπου ένας «αποστολέας» στέλνει μία αίτηση (request) σε ένα «παραλήπτη» και έπειτα περιμένει μία επιβεβαίωση (notification) της λήψης του μηνύματος του από τον παραλήπτη. Από την στιγμή που ο «παραλήπτης» απλά επιβεβαιώνει την λήψη, η απάντηση που στέλνει είναι «άδεια» και απλά καθυστερεί την περαιτέρω εκτέλεση στην πλευρά του αποστολέα.

**Υλοποίηση των Προτύπων 1 και 2 στην XASC.**

Στην XASC δεν υπάρχει η έννοια του ρητού *συντονισμού* της εκτέλεσης δύο διαγραμμάτων ροής με ανταλλαγή μηνυμάτων. Αντίθετα ο συντονισμός της εκτέλεσης διαγραμμάτων ροής επιτυγχάνεται με τον διαμοιρασμό (sharing) μεταβλητών στο «Context Repository» και την άσκηση «blocking» ενεργειών πάνω στις μεταβλητές αυτές. Και τα δύο πρότυπα μπορούν να υλοποιηθούν στην περίπτωση που αναφερόμαστε στην αλληλεπίδραση ενός διαγράμματος ροής και μίας εξωτερικής Ηλεκτρονικής Υπηρεσίας. Σε αυτή την περίπτωση η χρήση ενός εξωτερικού γεγονότος (external task) προσφέρει την λύση στο πρόβλημα μας. Ορίζοντας το είδος (mode) της επικοινωνίας σε “request-response” ο «αποστολέας» (διάγραμμα ροής) περιμένει τον «παραλήπτη» (Ηλεκτρονική Υπηρεσία) να απαντήσει, για να συνεχίσει την εκτέλεση του. Στην περίπτωση που αυτή η απάντηση είναι κενή τότε υλοποιείται το πρότυπο «Μονόδρομος» όπου αυτή η κενή απάντηση μπορεί να θεωρηθεί σαν την επιβεβαίωση της λήψης του μηνύματος από τον «παραλήπτη». Στην XASC το είδος της επικοινωνίας μεταξύ του διαγράμματος ροής και της καλούμενης (invoked) Ηλεκτρονικής Υπηρεσίας μπορεί να οριστεί και σαν “notification” όπου το διάγραμμα ροής συνεχίζει την εκτέλεση μίας ενέργειας χωρίς περιμένει για απάντηση. Αυτός ο τρόπος επικοινωνίας υλοποιεί τον «Μονόδρομο» (one-way) όπως τον ορίζει η WSDL, αλλά όχι όπως τον ορίζει το πρότυπο 2 μίας και δεν υπάρχει η επιβεβαίωση λήψης του μηνύματος από τον παραλήπτη. Θα δούμε αργότερα ότι με την χρήση του τύπου επικοινωνίας “notification” σε μία εξωτερική ενέργεια υλοποιείται ένα από τα πρότυπα ασύγχρονης επικοινωνίας (το πέρασμα παραμέτρων). Οι δύο αυτοί τρόποι επικοινωνίας μπορούν να χρησιμοποιηθούν και στην περίπτωση που το διάγραμμα ροής δεν καλεί (invokes) μία εξωτερική υπηρεσία άλλα δημιουργεί ένα γεγονός για την ενεργοποίηση ενός νέου διαγράμματος ροής. Σε αυτή την περίπτωση, αντί για εξωτερική ενέργεια (external task) έχουμε ενέργεια γεγονότος (event task), το οποίο προκαλεί την δημιουργία ενός γεγονότος που ενεργοποιεί ένα νέο διάγραμμα ροής. Στην περίπτωση που το είδος της επικοινωνίας οριστεί ως “request-response”, το «καλών» διάγραμμα ροής μπλοκάρει την εκτέλεση του και περιμένει το «νέο» διάγραμμα ροής να ολοκληρώσει την εκτέλεση του και πιθανώς να επιστρέψει κάποια αποτελέσματα (όχι με

την μορφή μηνύματος βέβαια) που μπορεί να επηρεάσουν την εκτέλεση του «καλούντος» διαγράμματος ροής. Σε περίπτωση που το είδος της επικοινωνίας οριστεί ως “notification” τότε ενεργοποιείται ένα νέο διάγραμμα ροής το οποίο εκτελείται ανεξάρτητα από αυτό που δημιούργησε το γεγονός που προκάλεσε την εκτέλεση του.

Στην BPEL4WS η σύγχρονη επικοινωνία υλοποιείται με την ύπαρξη μίας ενέργειας «invoke» στην πλευρά της διαδικασίας «αποστολέα» και ενός ζεύγους εντολών «receive», «reply» στην πλευρά της διαδικασίας «παραλήπτη». Η ενέργεια «invoke» περιλαμβάνει δύο «containers». Έναν εισόδου ο οποίος περιλαμβάνει το μήνυμα που θα αποσταλεί και ένας εξόδου που θα αποθηκεύσει το μήνυμα που θα καταφτάσει. Αντίστοιχα, η «receive» περιλαμβάνει έναν «container» εξόδου (μίας και απλά δέχεται ένα μήνυμα) ενώ η «reply» περιλαμβάνει ένα «container» εξόδου που περιλαμβάνει το μήνυμα (απάντηση) που θα αποσταλεί. Ο «Μονόδρομος» υλοποιείται απλά με ένα κενό «container» στην ενέργεια «reply» που λειτουργεί σαν ειδοποίηση (notification).

Στην BPML η σύγχρονη επικοινωνία υλοποιείται χρησιμοποιώντας την ενέργεια «action» και στην διαδικασία «αποστολέα» και στην διαδικασία «παρλήπτη». Η «action» του αποστολέα πρέπει να είναι τύπου “solicit-response” (ala WSDL) και για αυτό το μήνυμα εξόδου της «action» ορίζεται πρώτα από το μήνυμα εισόδου. Αντίστοιχα η «action» του παραλήπτη πρέπει να είναι τύπου “request-response” και για αυτό το μήνυμα εισόδου της «action» ορίζεται πρώτο από το μήνυμα εξόδου. Ο «Μονόδρομος» ορίζεται με την αποστολή ενός κενού μηνύματος από τον «παραλήπτη» προς τον «αποστολέα». Στην περίπτωση που η επικοινωνία μεταξύ των διαδικασιών αποσκοπεί στην δημιουργία ενός νέου στιγμιότυπου μίας διαδικασίας, αυτό μπορεί να γίνει με την ενέργεια «call» στην περίπτωση του προτύπου Αίτηση/Απάντηση (request/response) και με την ενέργεια «spawn» στην περίπτωση του προτύπου Μονόδρομος. Σε αυτή την περίπτωση έχουμε πρακτικά την εκτέλεση μίας υποδιαδικασίας με συγχρονισμό (call) ή και χωρίς (spawn) Το [ADH+02] αμφισβητεί το κατά πόσο αυτό αποτελεί ορθή υλοποίηση του πρότυπου Μονόδρομος μιας και δεν μοντελοποιεί την αποστολή επιβεβαίωσης.

**Πρότυπο 3 Σύγχρονη Ψηφοφορία** (Synchronous Polling)

Η σύγχρονη ψηφοφορία είναι μία μορφή σύγχρονης επικοινωνίας όπου ένας «αποστολέας» στέλνει μία αίτηση σε ένα «παραλήπτη» και συνεχίζει κανονικά την εκτέλεση του χωρίς να μπλοκάρει. Ο «αποστολέας» κοιτάει ανά τακτά χρονικά διαστήματα εάν η απάντηση έχει φτάσει. Μόλις παρατηρήσει ότι η απάντηση έχει φτάσει σταματάει την ψηφοφορία (polling).

**Υλοποίηση Προτύπου 3 στην XASC**

Στην XASC η έννοια της αναμονής για ένα μήνυμα, ή του ελέγχου για την λήψη ενός μηνύματος δεν γίνεται ρητά, δηλαδή με την χρήση κάποιου δομικού στοιχείου (construct). Απεναντίας, όταν ο «παραλήπτης» στείλει την απάντηση του, αυτό θα προκαλέσει «αυτόματα» την ενεργοποίηση του κατάλληλου διαγράμματος ροής που αντιστοιχεί σε αυτό το γεγονός. Η επέκταση της XASC που προτάθηκε σε προηγούμενο σημείο παρέχει ένα ακόμα τρόπο υλοποίησης της «σύγχρονης ψηφοφορίας».

Τόσο στην BPEL4WS όσο και στην BPML αυτό το πρότυπο υλοποιείται με την χρήση μίας παράλληλης διακλάδωσης. Στο ένα «παρακλάδι» είναι το κομμάτι της διαδικασίας που θέλουμε να εκτελεστεί και στο άλλο «παρακλάδι» είναι το δομικό στοιχείο που περιμένει την άφιξη του μηνύματος (είτε receive για την BPEL4WS είτε action με τύπο one-way για την BPML).

### 3.1.2 Ασύγχρονη Επικοινωνία

Σε αντίθεση με την σύγχρονη επικοινωνία, η ασύγχρονη δεν απαιτεί από τον αποστολέα να συντονίσει την εκτέλεση της διαδικασίας του με την ανταλλαγή των μηνυμάτων. Ο αποστολέας στέλνει ένα μήνυμα και συνεχίζει την εκτέλεση του αμέσως χωρίς να περιμένει για τυχόν απάντηση από τον παραλήπτη. Αυτό το είδος της επικοινωνίας χρειάζεται όταν ο σκοπός είναι η μεταφορά πληροφοριών ή ελέγχου.

**Πρότυπο 4 Πέρασμα Μηνυμάτων** (Message Passing)

Το πέρασμα παραμέτρων είναι ένας τύπος ασύγχρονης επικοινωνίας όπου ένας «αποστολέας» στέλνει σε ένα «παραλήπτη» μία αίτηση (request) και μετά πρακτικά «ξεχνάει» ότι την έστειλε συνεχίζοντας κανονικά την εκτέλεση της διαδικασίας του.

**Υλοποίηση Προτύπου 4 στην XASC.**

Στην ουσία η υλοποίηση αυτού του προτύπου έχει περιγραφεί και νωρίτερα. Στην XASC μία εξωτερική ενέργεια (external task) μπορεί να οριστεί τύπου “notification” όποτε και μετά την εκτέλεση της το διάγραμμα ροής συνεχίζει να εκτελείται κανονικά. Επίσης και μία ενέργεια γεγονότος (event task) μπορεί να οριστεί ως τύπου “notification” όποτε και αφού την εκτελέσει, το διάγραμμα ροής συνεχίζει την εκτέλεση του χωρίς να περιμένει το διάγραμμα ροής που ενεργοποιήθηκε να τελειώσει.

Στις BPEL4WS και BPML επίσης έχει περιγραφεί νωρίτερα η υλοποίηση αυτού του προτύπου. Στην μία υλοποιείται με μία ενέργεια «invoke» χωρίς container εξόδου και στην άλλη με μία ενέργεια «action» χωρίς μήνυμα εξόδου.

**Πρότυπο 5 Δημοσίευση/Εγγραφή** (Publish/Subscribe)

Είναι ένας τύπος ασύγχρονης επικοινωνίας όπου μία αίτηση αποστέλλεται από ένα «αποστολέα» και ο «παραλήπτης» επιλέγεται από μία δήλωση ενδιαφέροντος για μηνύματα αυτού του τύπου. Για παράδειγμα, έστω ένας οργανισμός που προσφέρει πληροφορίες για προϊόντα στους πελάτες του. Αν οι πελάτες ενδιαφέρονται για τέτοιου είδους πληροφορίες, «ειδοποιούν» το σύστημα, το οποίο κρατάει σε μία λίστα τους ενδιαφερόμενους πελάτες. Όταν πρόκειται να αποσταλεί ένα μήνυμα με πληροφορίες προϊόντων, ο οργανισμός ζητάει την λίστα αυτή με τους πελάτες και τους στέλνει τις πληροφορίες.

**Πρότυπο 6 Εκπομπή** (Broadcast)

Είναι ένας τύπος ασύγχρονης επικοινωνίας όπου μία αίτηση (μήνυμα) αποστέλλεται σε όλους τους συμμετέχοντες (παραλήπτες) ενός δικτύου. Κάθε συμμετέχον αποφασίζει αν το μήνυμα τον ενδιαφέρει κοιτάζοντας το περιεχόμενο του. Για παράδειγμα πριν κλείσει ένα σύστημα για εργασίες συντήρησης, όλοι οι χρήστες που είναι συνδεμένοι ενημερώνονται με ένα προειδοποιητικό μήνυμα.

**Υλοποίηση Προτύπων 5 και 6 στην XASC.**

Τα πρότυπα 5 και 6 δεν υποστηρίζονται άμεσα στην XASC όπως άλλωστε δεν υποστηρίζονται και στις BPEL4WS και BPML.

## *3.4 Σύγκριση της Εκφραστικότητας των BPEL4WS, BPML και XASC.*

Ο **Πίνακας 3.1** συνοψίζει τα πρότυπα που είδαμε στις προηγούμενες υποενότητες και την δυνατότητα ή μη απεικόνισης τους από τις γλώσσες που είναι υπό εξέταση. Με (+) περιγράφεται η άμεση υποστήριξη του προτύπου από μία γλώσσα, με (+/-) ή δυνατότητα αναπαράστασης με κάποια συνθετική λύση (work-around solution) ενώ με (-) περιγράφεται η αδυναμία αναπαράστασης ενός προτύπου από τη γλώσσα.


Στράτης Καρβουναράκης

Το συμπέρασμα που βγαίνει από τη σύγκριση είναι ότι η γλώσσα δημιουργίας των διαγραμμάτων ροής XASC έχει την ίδια σχεδόν εκφραστικότητα με αυτή των κυριότερων γλωσσών σύνθεσης Ηλεκτρονικών Υπηρεσιών. Απεναντίας η χρήση του δομικού στοιχείου Hparallel που εισάγαμε δίνει λύση σε προβλήματα που οι άλλες γλώσσες δυσκολεύονται να αντιμετωπίσουν όπως η δημιουργία πολλών στιγμιότυπων της ίδιας ενέργειας χωρίς επανάληψη (replication) του ορισμού της και χωρίς να είναι απαραίτητα εξαρχής γνωστό το πλήθος των στιγμιότυπων που πρέπει να δημιουργηθούν. Τα πρότυπα 6, 7 και 10 δεν υποστηρίζονται άμεσα από την XASC (δηλαδή με την ύπαρξη κάποιου εξειδικευμένου δομικού στοιχείου) γιατί αυτό θα αντίβαινε την έννοια των δομημένων διαγραμμάτων ροής (structured flowcharts) και θα παρέβαινε την βασική σχεδιαστική επιλογή για την ύπαρξη δομικών στοιχείων με καθαρή σημασιολογία και χωρίς «κρυμμένη» λειτουργικότητα. Τα πρότυπα 9 και 17 δεν εμφανίζονται συχνά στα είδη των εφαρμογών στις οποίες «εστιάζει» το AZTEC και για αυτό η αναπαράσταση τους περιορίζει σημαντικά την λειτουργικότητα της XASC. Το πρότυπο 15, όπως και μία υλοποίηση του προτύπου 8 αποτελούν μελλοντική εργασία με τον εμπλουτισμό της σημασιολογίας του δομικού στοιχείου «Switch». Τέλος το πρότυπο 19 απαιτεί την μελέτη ιδιοτήτων συναλλαγής (transactional properties) στη γλώσσα μας.

Όσων αφορά τα πρότυπα επικοινωνίας, η XASC υλοποιεί αυτά που υποστηρίζονται και από τις άλλες δύο γλώσσες με μία όμως διαφορετική φιλοσοφία επικοινωνίας. Στην XASC δεν υπάρχει η έννοια του συντονισμού (coordination) της εκτέλεσης δύο διαδικασιών ή δύο διαγραμμάτων ροής με την ανταλλαγή μηνυμάτων μεταξύ τους, αλλά με τον διαμοιρασμό δεδομένων στο CR που μπορεί να λειτουργήσει και σαν μέρος αμοιβαίου αποκλεισμού (mutex). Αυτό που απαιτείται και που υποστηρίζεται είναι τόσο η σύγχρονη όσο και η ασύγχρονη επικοινωνία με εξωτερικές Ηλεκτρονικές Υπηρεσίες, όσο και η αλληλεπίδραση μεταξύ διαγραμμάτων ροής της ίδιας διαδικασίας με την ενεργοποίηση από ένα διάγραμμα ροής νέων διαγραμμάτων ροής, αλλά και με την ενεργοποίηση διαγραμμάτων ροής εξαιτίας γεγονότων από εξωτερικές Ηλεκτρονικές Υπηρεσίες.

Η τελευταία έκδοση της BPEL4WS (BPEL4WS v1.1) επεκτάθηκε ούτως ώστε να επιτρέπει τον ορισμό -στα πλαίσια ενός «scope»- και «χειριστών γεγονότων» (event handlers) εκτός από «χειριστές λαθών» και «αντισταθμίσεων». Η χρήση ενός «χειριστή γεγονότων» αντιστοιχίζει την εκτέλεση μίας απλής ή σύνθετης ενέργειας σαν αποτέλεσμα της δημιουργίας ενός γεγονότος στα πλαίσια της εκτέλεσης του συγκεκριμένου «scope». Όπως ισχύει και για τους «χειριστές λαθών» και «αντισταθμίσεων», «χειριστές γεγονότων» μπορούν να οριστούν και στα πλαίσια της συνολικής διαδικασίας (καθολικό «scope»).

Σε αυτή την περίπτωση η εκτέλεση μίας ενέργειας αντιστοιχίζεται στην δημιουργία ενός γεγονότος καθ'όλη την διάρκεια εκτέλεσης της διαδικασίας.

Η σημασιολογία των χειριστών γεγονότων παρέχει μία «ένα προς ένα» αντιστοίχηση τους με τα ενεργά διαγράμματα ροής του μοντέλου μας προσδίδοντας καταυτό τον τρόπο στην BPEL4WS χαρακτηριστικά πλήρως ασύγχρονης επικοινωνίας τα οποία έλειπαν από την αρχική της έκδοση. Πράγματι στο μοντέλο ενεργών διαγραμμάτων ροής η δημιουργία ενός γεγονότος αντιστοιχίζεται με την ενεργοποίηση ενός διαγράμματος ροής που μπορεί να παραλληλιστεί με μία «σύνθετη» ενέργεια στην BPEL4WS. Παρόλαυτά δεν γίνεται σαφές στον ορισμό της BPEL4WS [T03] αν επιτρέπεται η ταυτόχρονη εκτέλεση παραπάνω του ενός «χειριστών γεγονότων» στα πλαίσια του ίδιου ή και διαφορετικών «scopes», και αν ναι ποια είναι η δυνατότητα αλληλεπιδράσεων και συγχρονισμού των ενεργειών που ορίζονται από αυτούς τους «χειριστές ενεργειών». Θα ήταν επίσης ενδιαφέρον να μελετηθεί κατά πόσο ο ορισμός «χειριστών γεγονότων» στα πλαίσια κάποιων «scopes» στην υψηλού επιπέδου διαδικασία (top-level process) της BPEL4WS μπορεί να μοντελοποιήσει τις «συνεδρίες» (sessions) που αναπτύσσονται κατά την εκτέλεση Ηλεκτρονικών Υπηρεσιών όπως αυτές που μελετάμε στο AZTEC.

## *3.5 Υλοποίηση σεναρίων Ηλεκτρονικού Εμπορίου και Ηλεκτρονικής Τηλεδιάσκεψης σε XASC.*

Έχοντας περιγράψει το μοντέλο ενεργών διαγραμμάτων ροής της γλώσσας XASC μπορούμε πλέον να μοντελοποιήσουμε και τα δύο σενάρια σύνθεσης Ηλεκτρονικών υπηρεσιών της **Ενότητας 2.2.4.** Το σενάριο ηλεκτρονικού εμπορίου συνθέτει μόνο διακριτές Ηλεκτρονικές Υπηρεσίες και βασίζεται στην *σύγχρονη* επικοινωνία ανάμεσα στην διαδικασία (XASC process) και τις εξωτερικές Ηλεκτρονικές Υπηρεσίες. Σε αυτό το παράδειγμα δεν δημιουργούνται συνεδρίες, επομένως και η εκτέλεση του «κύριου» διαγράμματος ροής, αυτού δηλαδή που ενεργοποιείται όταν φτάσει μία νέα παραγγελία, δεν επηρεάζεται από την λήψη γεγονότων. Στο σενάριο Ηλεκτρονικής Τηλεδιάσκεψης από την άλλη συνθέτονται «session-oriented» Ηλεκτρονικές Υπηρεσίες. Από τα αντικείμενα συνεδρίας δημιουργούνται τόσο σύγχρονες κλήσεις προς εξωτερικές Ηλεκτρονικές Υπηρεσίες, όσο και γεγονότα σε αυθαίρετες (arbitrary) χρονικές στιγμές που επηρεάζουν την εκτέλεση των ενεργών διαγραμμάτων ροής και την συνολική κατάσταση ολόκληρης της διαδικασίας.

| Δς | *Πρότυπα* | *γλώσσα* | | |
|---|---|---|---|---|
| | | BPEL | BPML | XASC |
| **Δομικά Πρότυπα** | 1 Ακολουθία | + | + | + |
| | 2 Παράλληλη Διακλάδωση | + | + | + |
| | 3 Συγχρονισμός | + | + | + |
| | 4 Αποκλειστική Επιλογή | + | + | + |
| | 5 Απλή Συγχώνευση | + | + | + |
| | 6 Πολλαπλή Επιλογή | + | - | - |
| | 7 Συγχώνευση με Συγχρονισμό | + | - | - |
| | 8 Πολλαπλή Συγχώνευση | - | +/- | + |
| | 9 Διευκρινητής | - | - | - |
| | 10 Βρόγχοι | - | - | - |
| | 11 Έμμεσος Τερματισμός | + | + | + |
| | 12 Πολλαπλά Στιγμιότυπα χωρίς Συγχρονισμό | + | + | + |
| | 13 Π.Σ. με Συγχρονισμό και γνώση του πλήθους κατά την Σχεδίαση | + | + | + |
| | 14 Π.Σ. με Συγχρονισμό και γνώση του πλήθους κατά την Εκτέλεση | - | - | + |
| | 15 Π.Σ. με Συγχρονισμό χωρίς πρότερη γνώση του πλήθους τους. | - | - | - |
| | 16 Επιλογή με Καθυστέρηση | + | + | + |
| | 17 Μη Διατεταγμένη Ακολουθία | +/- | - | - |
| | 18 Ορόσημο | - | - | +/- |
| | 19 Ακύρωση Ενέργειας | + | + | - |
| | 20 Ακύρωση Διαδικασίας | + | + | + |
| **Πρότυπα Επικοινωνίας** | 1 Αίτηση/Απάντηση | + | + | + |
| | 2 Μονόδρομος | + | + | + |
| | 3 Σύγχρονη Ψηφοφορία | + | + | + |
| | 4 Πέρασμα Μηνυμάτων | + | + | + |
| | 5 Δημοσίευση/Εγγραφή | - | - | - |
| | 6 Εκπομπή | - | - | - |

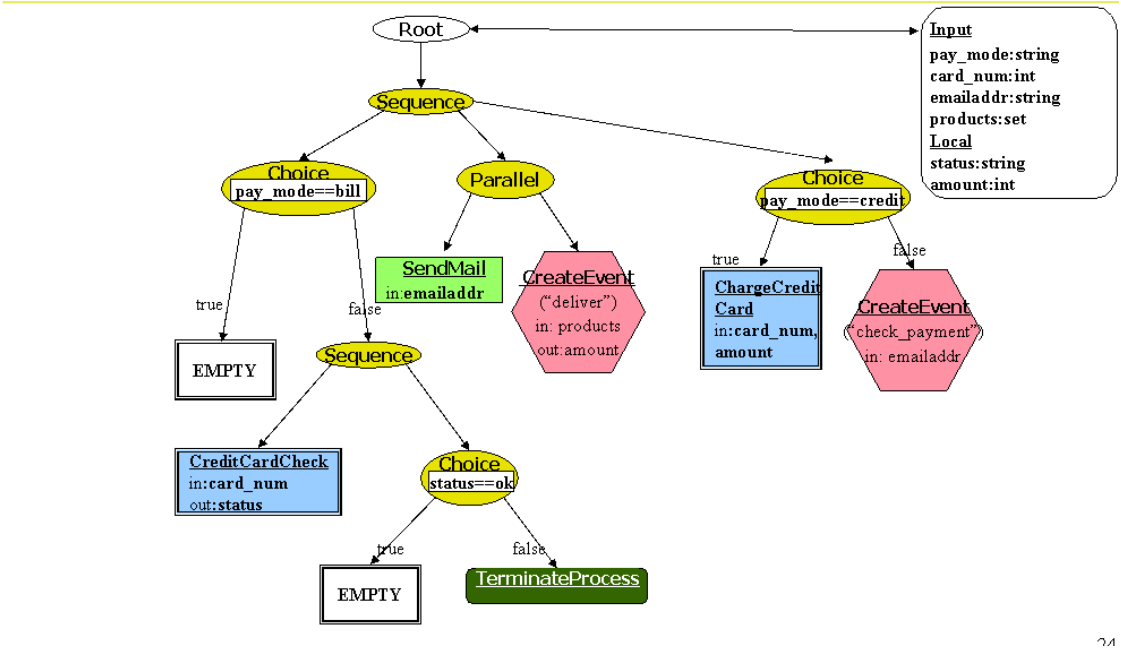
**Πίνακας 3.1 Συγκριτικός πίνακας των BPEL4WS, BPML και XASC χρησιμοποιώντας τα Δομικά Πρότυπα και τα Πρότυπα Επικοινωνίας.**

### 3.5.1 Ηλεκτρονικό Εμπόριο

Ένα τυπικό σενάριο Ηλεκτρονικού Εμπορίου περιγράφηκε με λεπτομέρεια στις **Ενότητες 2.1.2 και 2.4.2.1** και στην συνέχεια αναπαραστάθηκε με διάφορους φορμαλισμούς όπως δίκτυα Petri, χάρτες Κατάστασης (state charts) αλλά και με μία αφαιρετική (abstract) Ροή Εργασίας (workflow) χωρίς αυστηρή σημασιολογία. Συνοπτικά όταν μία νέα παραγγελία καταφτάνει επιλέγεται ο τρόπος πληρωμής. Αν ο τρόπος πληρωμής είναι με πιστωτική κάρτα, γίνεται έλεγχος εγκυρότητας της κάρτας. Στην συνέχεια γίνονται δύο παράλληλες ενέργειες. Η μία είναι «απλή» και στέλνει ένα ενημερωτικό μήνυμα ηλεκτρονικού ταχυδρομείου στον πελάτη και η άλλη «που είναι μία «σύνθετη» ενέργεια (δηλαδή ενέργεια γεγονότος), παραγγέλνει» κάθε προϊόν που είναι στην λίστα από ένα «μαγαζί». Αυτό που γίνεται στην συνέχεια, είναι ότι στην περίπτωση πληρωμής με πιστωτική κάρτα, χρεώνεται η κάρτα, ενώ στην περίπτωση πληρωμής με λογαριασμό, σε περίπτωση που ο πελάτης δεν έχει τακτοποιήσει την πληρωμή σε διάρκεια 6 εβδομάδων στις οποίες έχουν παρεμβληθεί 3 υπενθυμίσεις πληρωμής (με μήνυμα ηλεκτρονικού ταχυδρομείου), το θέμα προωθείται στην Νομική Υπηρεσία και η παραγγελία ακυρώνεται.

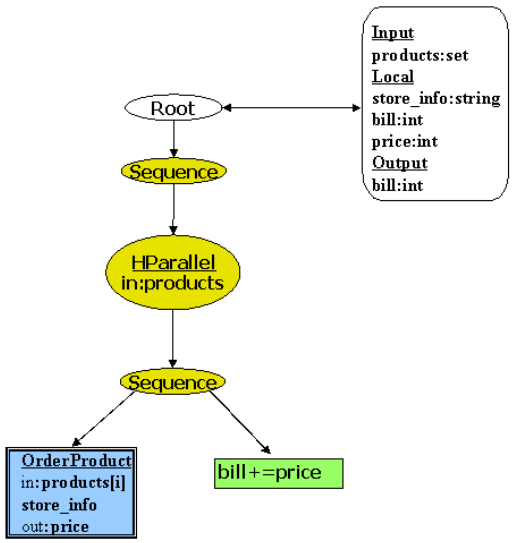
Το σχήμα της διαδικασίας (process schema) του παραδείγματος Ηλεκτρονικού Εμπορίου αποτελείται από πέντε ενεργά διαγράμματα ροής, ονομαστικά τα *new_order*, *deliver*, *payment*, *check_payment* και *due_day*. Ας δούμε αναλυτικά τι λειτουργικότητα έχει καθένα από αυτά τα διαγράμματα ροής:

**New order** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.17** ενεργοποιείται όταν ένας πελάτης αιτηθεί μία παραγγελία. Καταρχήν ελέγχεται το είδος της πληρωμής που έχει επιλέξει ο πελάτης. Αν είναι «πληρωμή με κάρτα» τότε καλείται μια εξωτερική Ηλεκτρονική Υπηρεσία (πιθανώς κάποια Credit Service) η οποία ελέγχει την εγκυρότητα της πιστωτικής κάρτας. Αν η κάρτα δεν είναι έγκυρη, η παραγγελία ακυρώνεται. Στην συνέχεια γίνονται παράλληλα δύο ενέργειες. Η μία αποστέλλει ένα ενημερωτικό μήνυμα ηλεκτρονικού ταχυδρομείου στον πελάτη και η άλλη δημιουργεί ένα γεγονός που ενεργοποιεί το διάγραμμα ροής «deliver» το οποίο χειρίζεται την ανεύρεση και την αποστολή των προϊόντων στον πελάτη. Στην συνέχεια υπάρχουν δύο περιπτώσεις. Σε περίπτωση που ο πελάτης έχει επιλέξει «πληρωμή με κάρτα», απλά χρεώνεται η κάρτα του, ενώ αν έχει επιλέξει «πληρωμή με λογαριασμό» ενεργοποιείται το διάγραμμα ροής «check_payment» το οποίο ελέγχει «περιοδικά» αν έχει ολοκληρωθεί η πληρωμή και αν όχι λαμβάνει τα κατάλληλα μέτρα.



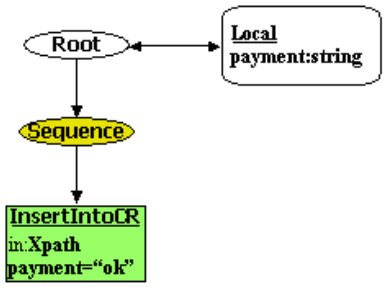

**Εικόνα 3.17 Το διάγραμμα ροής «new_order» της διαδικασίας Ηλεκτρονικού Εμπορίου**

**Deliver** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.18** χειρίζεται -για μία λίστα- με προϊόντα την ανεύρεση του καταστήματος που έχει κάθε προϊόν διαθέσιμο και έπειτα την παραγγελία του προϊόντος. Παρόλο πού οι ενέργειες «FindStore» και «OrderProduct» θα μπορούσαν να γίνουν επαναληπτικά για όλα τα προϊόντα στην λίστα (με χρήση ενός στοιχείου «*While_do*») ακόμα πιο κατάλληλη λύση είναι η χρήση του στοιχείου «*Hparallel*» το οποίο θα εκτελέσει τις ενέργειες αυτές για όλα τα προϊόντα στην λίστα παράλληλα.



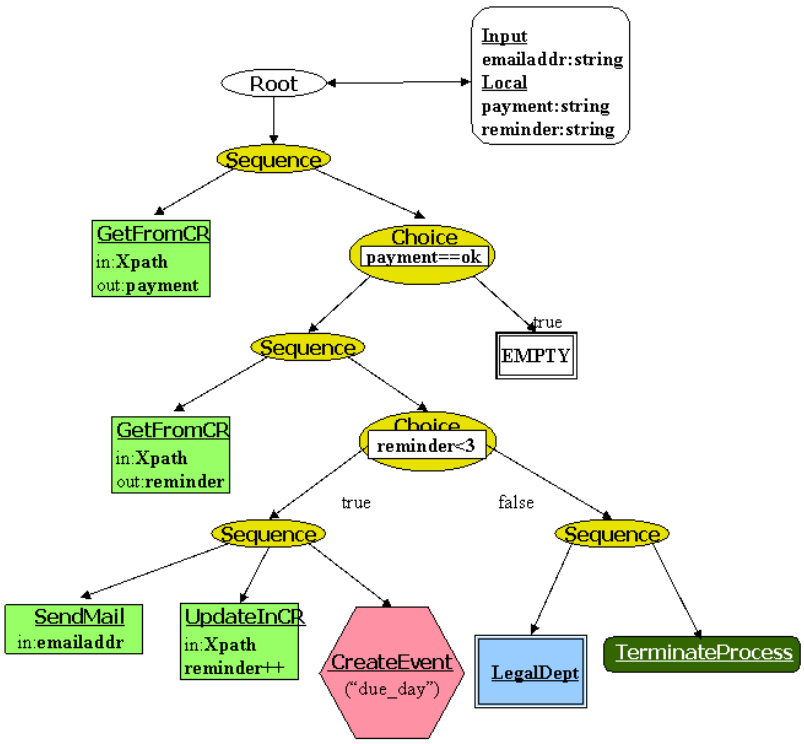

**Εικόνα 3.18 Το διάγραμμα ροής «deliver» της διαδικασίας Ηλεκτρονικού Εμπορίου**

**Payment** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.19** ενεργοποιείται όταν δημιουργηθεί ένα γεγονός από την υπηρεσία που χειρίζεται τις πληρωμές, ότι ο πελάτης ολοκλήρωσε την πληρωμή. Το διάγραμμα ροής αυτό περιλαμβάνει μία ενέργεια η οποία καταχωρεί στο «Context Repository» την μεταβλητή «payment» με τιμή «ok».



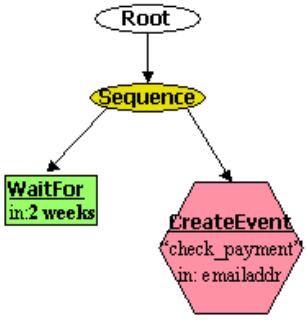

**Εικόνα 3.19 Το διάγραμμα ροής «payment» της διαδικασίας Ηλεκτρονικού Εμπορίου**

**Check_payment** Αυτό το διάγραμμα ροής που φαίνεται στην **Εικόνα 3.20** ελέγχει περιοδικά αν έχει ολοκληρωθεί η πληρωμή από τον πελάτη. Καταρχήν προσπαθεί να ανακτήσει την μεταβλητή «payment» από το «Context Repository». Αν τα καταφέρει, και έχει τιμή «ok» σημαίνει ότι η πληρωμή έχει γίνει και το διάγραμμα ροής ολοκληρώνει την εκτέλεση του. Αν δεν βρει την μεταβλητή, σημαίνει ότι η πληρωμή δεν έχει γίνει ακόμα (βλ. διάγραμμα ροής «payment») οπότε πρέπει να λάβει κάποια μέτρα. Καταρχήν ανακτά από το «Context Repository» την μεταβλητή «reminder» που μετράει πόσες υπενθυμίσεις έχουν αποσταλεί στον πελάτη. Αν έχουν αποσταλεί ήδη τρεις υπενθυμίσεις, καλείται η ενέργεια που ειδοποιεί την Νομική Υπηρεσία και η διαδικασία τερματίζεται. Αν έχουν αποσταλεί λιγότερες από τρεις υπενθυμίσεις τότε αποστέλλεται ένα μήνυμα ηλεκτρονικού ταχυδρομείου (υπενθύμιση) στον πελάτη, αυξάνεται η τιμή της μεταβλητής «reminder» κατά 1 και δημιουργείται ένα γεγονός που προκαλεί την ενεργοποίηση του διαγράμματος ροής «due_day».




**Εικόνα 3.20 Το διάγραμμα ροής «check_payment» της διαδικασίας Ηλεκτρονικού Εμπορίου**

**Due_day** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.21** περιμένει (wait-for task) για 2 εβδομάδες και έπειτα ενεργοποιεί ξανά το διάγραμμα ροής «check_payment» με την λειτουργικότητα που περιγράψαμε παραπάνω.




**Εικόνα 3.21 Το διάγραμμα ροής «due_day» της διαδικασίας Ηλεκτρονικού Εμπορίου**

Όπως είδαμε, η εκφραστικότητα της XASC αρκεί για να εκφραστεί με επιτυχία το παράδειγμα Ηλεκτρονικού Εμπορίου. Καταρχήν η χρήση του στοιχείου «HParallel» για την ταυτόχρονη ανεύρεση και παραγγελία όλων τον προϊόντων αποτελεί καινοτομία με άλλες αναπαραστάσεις που θα χρησιμοποιούσαν επανάληψη. Η ασύγχρονη φύση της λήψης της πληρωμής αναπαρίσταται επίσης με τρόπο βέλτιστο από την XASC αφού δεν είναι αναγκαίο να «μπλοκάρει» κάποια διαδικασία περιμένοντας την λήψη του γεγονότος

πληρωμής. Μοναδική εξαίρεση στην «κομψότητα» αναπαράστασης του παραδείγματος αποτελεί το διάγραμμα ροής «due_day» το οποίο πρακτικά εκτελεί μία αναμονή για 2 εβδομάδες και έπειτα δημιουργεί το γεγονός που θα προκαλέσει τον έλεγχο για την λήψη της πληρωμής. Φυσικά αυτό δεν αποτελεί μειονέκτημα του μοντέλου μας, αλλά της φύσης του παραδείγματος. Αυτό το πρόβλημα μπορεί να επιλυθεί υποθέτοντας ότι κάθε δύο εβδομάδες κάποια εξωτερική Ηλεκτρονική Υπηρεσία (για παράδειγμα η ίδια που μας ειδοποιεί για την άφιξη της αμοιβής) μας ειδοποιεί για αυτό με ένα γεγονός (deadline).

Στο παράδειγμα αυτό δεν τονίσαμε την δημιουργία συνεδριών όπως θα κάνουμε στο παράδειγμα ηλεκτρονικής τηλεδιάσκεψης μιας και οι Ηλεκτρονικές Υπηρεσίες που καλούνται δεν αναπτύσσουν συνεδρίες ρητά όπως οι «τηλεπικοινωνιακές» υπηρεσίες. Παρόλαυτά θα μπορούσε και εδώ να θεωρηθεί ως συνεδρία η διαδικασία παράδοσης ενός προϊόντος από την στιγμή που έχει διάρκεια ζωής από την στιγμή που γίνεται η παραγγελία σε κάποιο μαγαζί μέχρι την στιγμή που δημιουργείται το «ασύγχρονο γεγονός» της λήψης της πληρωμής οπότε και λήγει η συνεδρία. Τα γεγονότα λήξης των εκάστοτε «διοριών» (κάθε δύο εβδομάδες) θα μπορούσαν να δημιουργούνται από αυτά τα αντικείμενα συνεδρίας πωλήσεων. Υπό αυτή την έννοια για κάθε συνολική παραγγελία θα αναπτύσσονταν τόσες συνεδρίες πωλήσεων όσα και τα ξεχωριστά προϊόντα στην παραγγελία. Ακολουθώντας βέβαια αυτή τη λογική θα έπρεπε να περιμένουμε, όχι για μία πληρωμή αλλά για τόσες όσες και τα διαφορετικά αντικείμενα συνεδρίας. Επίσης σε περίπτωση «μη πληρωμής» και άφιξης του «τελικού» αδιεξόδου θα μπορούσαμε να ξέρουμε πρακτικά πιο αντικείμενο συνεδρίας πωλήσεων δεν ολοκλήρωσε την εκτέλεση του. Το παράδειγμα Ηλεκτρονικού Εμπορίου είναι ένα τυπικό παράδειγμα και μπορεί να εκφραστεί με επιτυχία από τις περισσότερες γλώσσες σύνθεσης Ηλεκτρονικών Υπηρεσιών. Παρόλαυτά εκμεταλλευόμενοι όλο τα «εύρος εκφραστικότητας» του μοντέλου ενεργών διαγραμμάτων ροής μπορούμε να παρέχουμε μία «πλούσια» σημασιολογικά αναπαράσταση.

### 3.5.2 Ηλεκτρονική Τηλεδιάσκεψη

Η περιγραφή του σεναρίου Ηλεκτρονικής Τηλεδιάσκεψης έγινε αναλυτικά στην **Ενότητα 2.4.2.2**. Για να υπενθυμίσουμε μερικά από τα βασικά του στοιχεία, ένας χρήστης μπορεί να αιτηθεί μία Ηλεκτρονική Τηλεδιάσκεψη από ένα δεδομένο αρχικό χρονικό σημείο έως ένα δεδομένο τελικό χρονικό σημείο με ένα δεδομένο αριθμό συμμετεχόντων και πιθανώς παραπάνω από μία μορφές επικοινωνίας (ηχητική τηλεδιάσκεψη, συνεδρία με γραπτά μηνύματα (chat), συνεργατική πλοήγηση στο Διαδίκτυο κ.λ.π). Παράλληλα πιθανή είναι

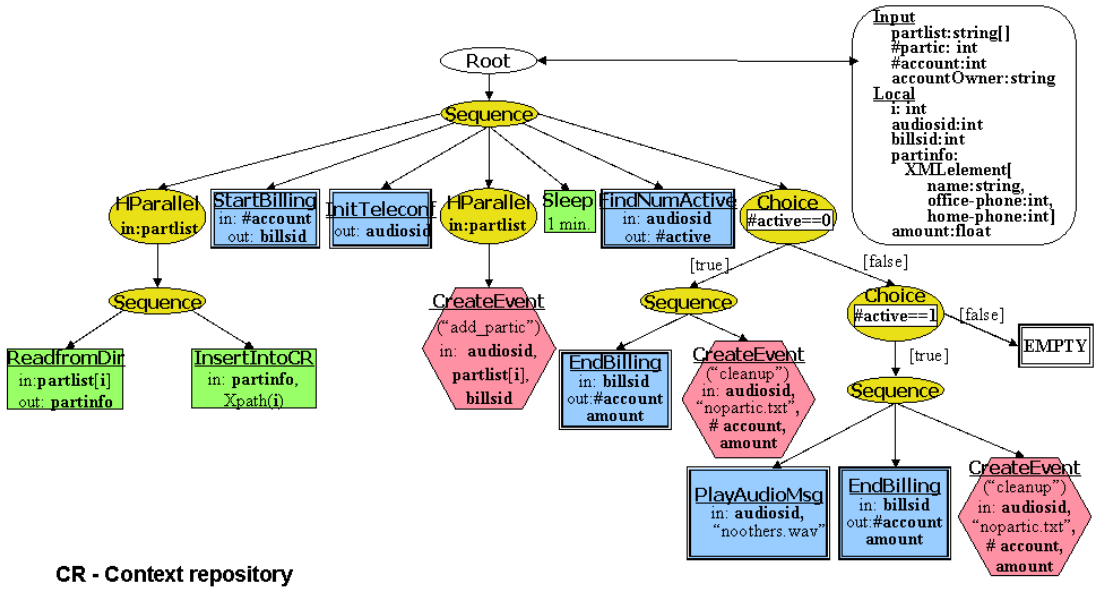
η ύπαρξη υπηρεσιών «παρουσίας» που εντοπίζουν την παρουσία ή την αποσύνδεση κάποιου συμμετέχοντα από το δίκτυο, και υπηρεσιών χρέωσης που ασχολούνται με την χρέωση των συνεδριών. Στο παράδειγμά που θα περιγράψουμε εδώ θα περιοριστούμε για λόγους απλότητας και συντομίας στην ύπαρξη (και αλληλεπίδραση) δύο Ηλεκτρονικών Υπηρεσιών:

b) Της *audio_conf_service* η οποία χρησιμοποιείται για να εκκινεί τηλεδιασκέψεις, να προσθέτει συμμετέχοντες σε τηλεδιασκέψεις και να παρακολουθεί πότε κάποιος συμμετέχων έχει κλείσει το τηλέφωνο. Αυτή η υπηρεσία είναι «session oriented» δηλαδή επικοινωνεί με ένα «τηλεπικοινωνιακό gateway» το οποίο βασίζεται σε πρωτόκολλα όπως το SIP ή το H.323 για να δημιουργεί συνεδρίες (session) ηχητικής τηλεδιάσκεψης, ανταλλαγής γραπτών μηνυμάτων κ.τ.λ

c) Της *billing_service* η οποία διατηρεί προπληρωμένους λογαριασμούς, παρακολουθεί πόσα χρήματα έχουν «καταναλωθεί» μέχρι την συγκεκριμένη χρονική στιγμή από την τηλεδιάσκεψη και δημιουργεί ένα γεγονός όταν/άν ένας προπληρωμένος λογαριασμός μείνει χωρίς χρήματα. Η υπηρεσία αυτή δεν αναπτύσσει με την «αυστηρή» έννοια που αναπτύσσει η «audio_conf_service» παρόλαυτά το γεγονός ότι η διαδικασία χρέωσης έχει μεγάλη διάρκεια και δημιουργεί αλλά αποκρίνεται και σε «ασύγχρονα» γεγονότα θα μας οδηγήσει συχνά στην χρήση του όρου «συνεδρία χρέωσης».

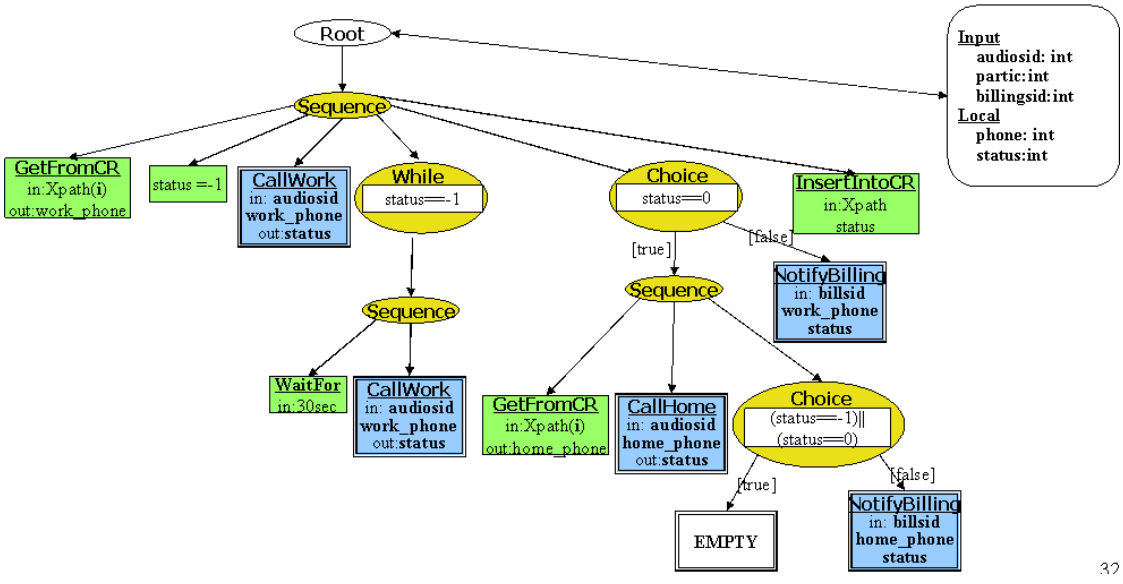
Το «σχήμα» αυτής της διαδικασίας (process schema) περιλαμβάνει πέντε «σχήματα» ενεργών διαγραμμάτων ροής, τα *Start_audio_conference*, *Add_participant*, *Participant_hang_up*, *Out_of_money* και *Clean_up*. Ας δούμε αναλυτικά τι λειτουργικότητα έχει καθένα από αυτά τα ενεργά διαγράμματα ροής:

**Start_audio_conference** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.22** συλλέγει δεδομένα για τους συμμετέχοντες από ένα κατάλογο (συμπεριλαμβανομένων των τηλεφώνων σπιτιού και εργασίας), εκκινεί την ηχητική τηλεδιάσκεψη (audio conference), ειδοποιεί την υπηρεσία χρέωσης (billing_service) για την νέα τηλεδιάσκεψη, και μετά δημιουργεί γεγονότα που προκαλούν την ενεργοποίηση του διαγράμματος ροής «Add_participant» για κάθε συμμετέχων. Αν ο συμμετέχων «εντοπιστεί» και προστεθεί στην τηλεδιάσκεψη τότε ειδοποιείται η υπηρεσία χρέωσης.




**Εικόνα 3.22 Το διάγραμμα ροής «start_audio_conference» της διαδικασίας Ηλεκτρονικής Τηλεδιάσκεψης.**

**Add_participant** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.23** επιχειρεί να προσθέσει ένα συμμετέχων στην ηχητική τηλεδιάσκεψη, καλώντας πρώτα στο γραφείο του και αν δεν απαντήσει, καλώντας στο σπίτι του. Αν και δεν θα φανεί στο απλοποιημένο αυτό παράδειγμα αυτό το διάγραμμα ροής μπορεί να ενεργοποιηθεί και κατά την διάρκεια της τηλεδιάσκεψης από γεγονότα που προέρχονται από την υπηρεσία παρουσίας (που εδώ όμως δεν παρουσιάζεται).



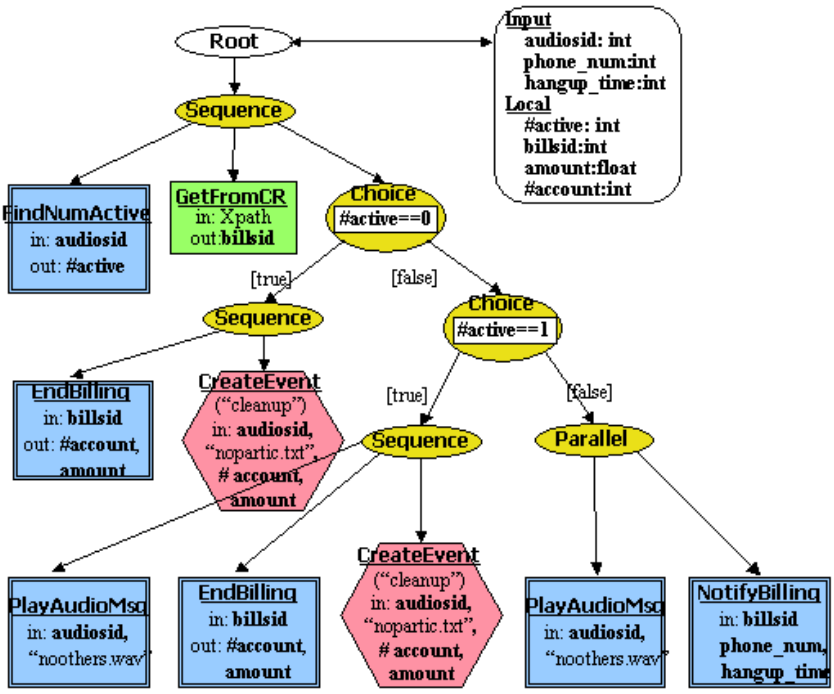

**Εικόνα 3.23 Το διάγραμμα ροής «add_participant» της διαδικασίας Ηλεκτρονικής Τηλεδιάσκεψης**

**Participant_hang_up** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.24** ενεργοποιείται από ένα γεγονός που δημιουργείται όταν ένας συμμετέχων αποσυρθεί από την ηχητική τηλεδιάσκεψη (κλείσει το τηλέφωνο του). Σε αυτή την περίπτωση εάν μένουν

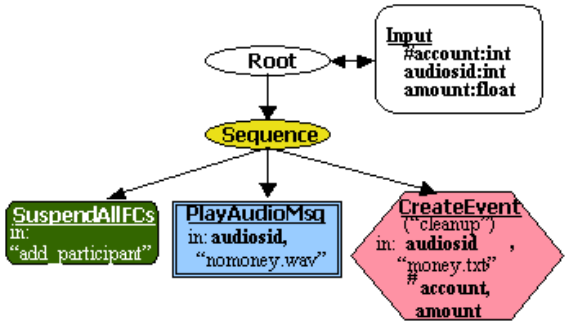
«ενεργοί» δύο οι παραπάνω συμμετέχοντες, απλά ειδοποιείται η υπηρεσία χρέωσης. Σε περίπτωση όμως που παραμένουν «ενεργοί» ένας ή κανένας συμμετέχοντας τότε δημιουργείται το γεγονός που ενεργοποιεί το διάγραμμα ροής «Clean_up».

**Out_of_money** Αυτό το διάγραμμα ροής που παρουσιάζεται στην **Εικόνα 3.25** ενεργοποιείται αν η υπηρεσία χρέωσης (billing service) δημιουργήσει ένα γεγονός που να δείχνει ότι ο λογαριασμός στον οποίο χρεώνεται η τηλεδιάσκεψη έχει υπόλοιπο μηδέν. Σε αυτή τη περίπτωση δημιουργείται ένα ηχητικό μήνυμα που ενημερώνει τους συμμετέχοντες ότι τελείωσαν τα χρήματα και «καλεί» το διάγραμμα ροής «Clean_up». Πολύ σημαντικό στοιχείο είναι ότι η ενεργοποίηση του διαγράμματος ροής «Out_of_money» ακυρώνει τυχόν εκκρεμή διαγράμματα ροής «Add_participant» μιας και δεν έχει νόημα να προστεθεί ένας συμμετέχων σε μία τηλεδιάσκεψη που τερματίζεται.

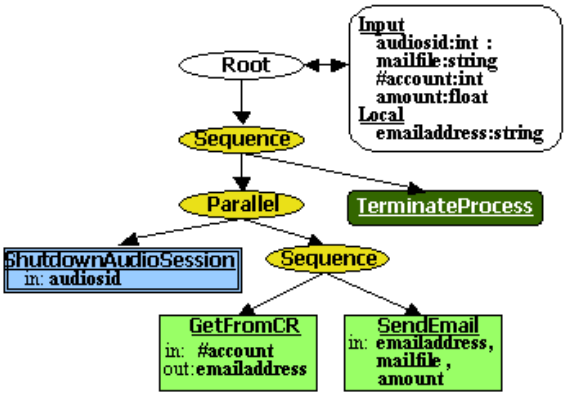
**Clean_up** Αυτό το διάγραμμα ροής που απεικονίζεται στην **Εικόνα 3.26** αιτείται από την *audio_conf_service* να τερματίσει την τηλεδιάσκεψη.




**Εικόνα 3.24 Το διάγραμμα ροής «participant_hang_up» της διαδικασίας Ηλεκτρονικής Τηλεδιάσκεψης**




**Εικόνα 3.25 Το διάγραμμα ροής «out_of_money» της διαδικασίας Ηλεκτρονικής Τηλεδιάσκεψης**




**Εικόνα 3.26 Το διάγραμμα ροής «clean_up» της διαδικασίας Ηλεκτρονικού Εμπορίου**

Τα διαγράμματα ροής που μόλις παρουσιάσαμε αφορούν την απλοποιημένη περίπτωση στην οποία αναπτύσσονται μόνο δύο αντικείμενα συνεδρίας, η συνεδρία ηχητικής τηλεδιάσκεψης (audio conference session) και η συνεδρία χρέωσης (billing session). Οι αλληλεπιδράσεις γίνονται ακόμα πιο περίπλοκες στην περίπτωση που αναπτύσσονται ακόμα περισσότερα αντικείμενα συνεδρίας, όπως στο αρχικό μας σενάριο στο οποίο εκτός από συνεδρία ηχητικής τηλεδιάσκεψης περιγράφεται και η ύπαρξη συνεδρίας γραπτών μηνυμάτων που χρησιμοποιείται για την «σιωπηρή» ειδοποίηση των συμμετεχόντων για κάποια γεγονότα (όπως για παράδειγμα ότι κάποιος έχασε την σύνδεση του και δεν συμμετέχει πλέον στην ηχητική τηλεδιάσκεψη) αλλά και για την περαιτέρω μεταξύ τους επικοινωνία. Σε τέτοιες περιπτώσεις η μηχανή εκτέλεσης του AZTEC πρέπει να μπορεί να αντιστοιχίσει τα γεγονότα που δημιουργούνται όχι μόνο με κάποιο ενεργό στιγμιότυπο διαδικασίας XASC αλλά και με κάποια ενεργή συνεδρία που έχει αναπτυχθεί στα πλαίσια αυτής της διαδικασίας. Στην επόμενη ενότητα θα περιγράψουμε το σενάριο ηλεκτρονικής τηλεδιάσκεψης στην ολοκληρωμένη του μορφή, δηλαδή με δύο «ταυτόχρονες» συνεδρίες τηλεδιάσκεψης, μία ηχητικής και μία γραπτών μηνυμάτων και με δύο συνεδρίες χρέωσης, μία για κάθε μία συνεδρία τηλεδιάσκεψης. Επίσης θα

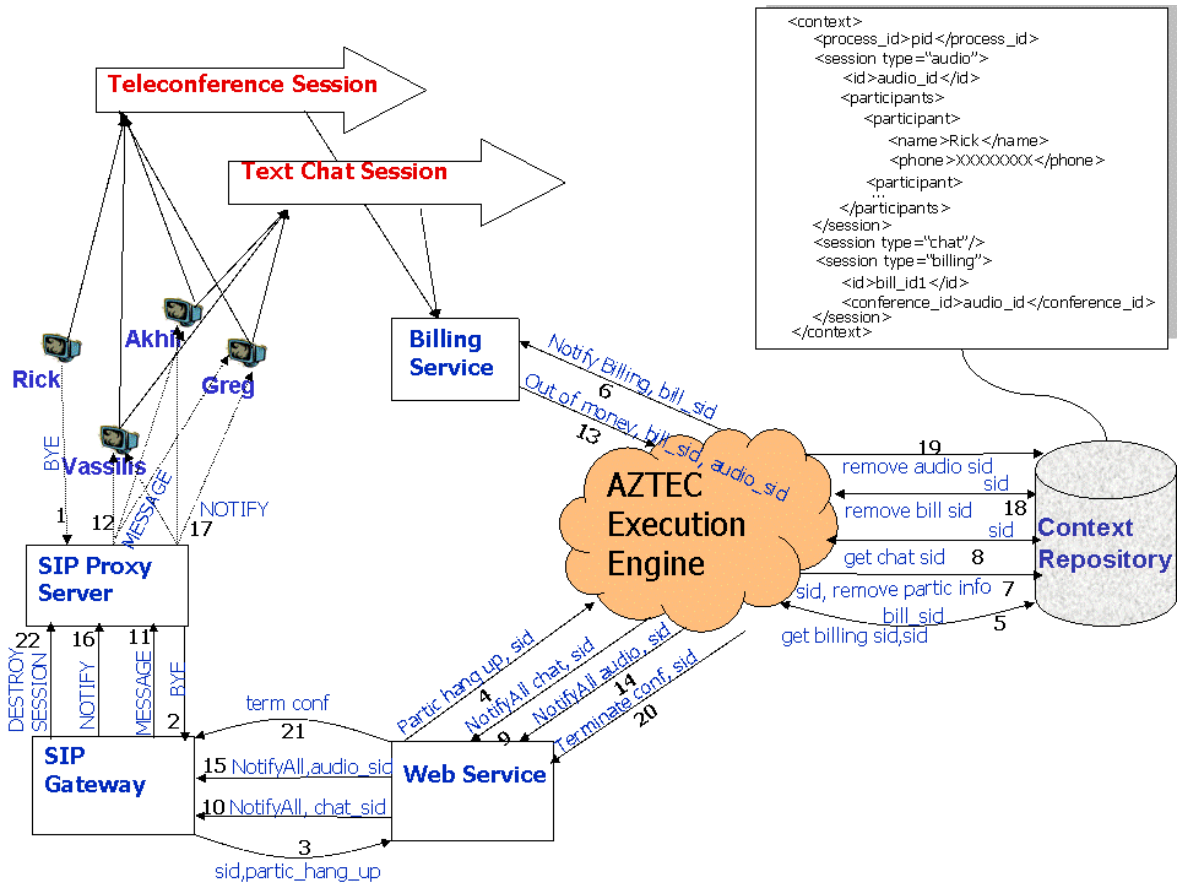
τονίσουμε το ρόλο του «Context Repository» σαν μία «αποθήκη πληροφοριών» στην οποία αποθηκεύονται πληροφορίες σχετικές με τις διαφορετικές συνεδρίες που έχουν αναπτυχθεί στα πλαίσια της διαδικασίας.

## *3.6 Αλληλεπιδράσεις ενεργών συνεδριών και το Context Repository*

Η διατήρηση πληροφοριών σχετικών με τις διάφορες συνεδρίες που αναπτύσσονται στα πλαίσια μίας διαδικασίας διατηρούνται στην «αποθήκη δεδομένων» της πλατφόρμας AZTEC, δηλαδή στο «Context Repository». Στο σχήμα της **Εικόνας 3.27** βλέπουμε ότι το αναγνωριστικό της συνεδρίας από την οποία προέρχεται ένα γεγονός ή προς την οποία «απευθύνεται» κάποιο γεγονός επισυνάπτεται πάντα στα δεδομένα πού «φέρει» το γεγονός.

Στην περίπτωση του παραδείγματος τηλεδιάσκεψης το «Context Repository» θα περιλαμβάνει πληροφορίες για τρεις συνεδρίες, την συνεδρία ηχητικής τηλεδιάσκεψης, την συνεδρία γραπτής τηλεδιάσκεψης και την συνεδρία χρέωσης. Για κάθε συνεδρία περιλαμβάνει χαρακτηριστικές πληροφορίες όπως τον τύπο της συνεδρίας (π.χ. «audio», «text» ή «billing») το αναγνωριστικό της συνεδρίας (session_id) τους συμμετέχοντες στην συνεδρία (αν μιλάμε για της δυο συνεδρίες τηλεδιάσκεψης) και ότι άλλες πληροφορίες κρίνονται απαραίτητες. Η διατήρηση πληροφοριών σχετικών με τις συνεδρίες στο «Context Repository» είναι απαραίτητη για ένα ακόμη λόγο πλην της συσχέτισης των γεγονότων που δημιουργούνται με «ενεργές συνεδρίες». Όπως βλέπουμε στην Εικόνα 3.27 η Ηλεκτρονική Υπηρεσία που χειρίζεται την τηλεδιάσκεψη αποτελεί πρακτικά ένα «wrapper» για το «gateway» που χειρίζεται την δημιουργία συνεδριών (είτε με χρήση του πρωτοκόλλου SIP είτε του H.323 κ.λ.π). Το «gateway» αυτό μπορεί να διαχειρίζεται πολλαπλές συνεδρίες από πολλούς διαφορετικούς «πελάτες» (clients) και πολλών διαφορετικών ειδών (δηλαδή ηχητικές, γραπτών μηνυμάτων κ.λ.π) και αποθηκεύει πληροφορίες για αυτές τις «ενεργές συνεδρίες» σε ένα εξυπηρετητή «proxy».

Όταν ένα «ενεργό διάγραμμα ροής» θέλει να εκτελέσει μία λειτουργία, όπως για παράδειγμα να παίξει ένα ηχητικό μήνυμα σε όλους τους συμμετέχοντες της ηχητικής τηλεδιάσκεψης, πρέπει στην κλήση SOAP της αντίστοιχης μεθόδου στην Ηλεκτρονική Υπηρεσία που χειρίζεται την τηλεδιάσκεψη να περιλαμβάνει και το «session_id» της ηχητικής τηλεδιάσκεψης, ειδάλλως το «gateway» δεν θα μπορεί να συσχετίσει την «ενέργεια» με κάποια ενεργή συνεδρία.




**Εικόνα 3.27 Οι αλληλεπιδράσεις μεταξύ των αντικειμένων συνεδρίας και της πλατφόρμας AZTEC**

Παρόμοιες απαιτήσεις υπάρχουν και όταν πρέπει να «ενημερωθεί» (updated) η συνεδρία χρέωσης σε μία Ηλεκτρονική Υπηρεσία η οποία διατηρεί πολλές ταυτόχρονες συνεδρίες χρέωσης. Ας τα δούμε όλα αυτά λίγο πιο διαισθητικά στο σενάριο τηλεδιάσκεψης που μελετάμε

Άν κάποια στιγμή ο χρήστης «Rick» αποφασίσει να φύγει από την τηλεδιάσκεψη ή χάσει την σύνδεση του δημιουργείται ένα γεγονός που προωθείται απ;ο το τελικό σημείο επικοινωνίας (endpoint) δηλαδή τον «Rick», στο «gateway» και απο εκεί με τις κατάλληλες κλήσεις SOAP δημιουργείται ένα γεγονός τύπου «participant_hang_up» στην μηχανή εκτέλεσης του AZTEC. Αυτή την διαδικασία την βλέπουμε στα βήματα 1 έως 4 της **Εικόνας 3.27**. Στο γεγονός που δημιουργείται «περιέχεται» το «session_id» της τηλεδιάσκεψης στην οποία συμμετείχε και ο «Rick». Πρώτο βήμα είναι η ενημέρωση της υπηρεσίας χρέωσης για την ελάττωση του πλήθους των συμμετεχόντων κατά 1. Για να γίνει αυτό πρέπει να ανακτηθεί το «session_id» της συνεδρίας χρέωσης από το «Context Repository». Για να είμαστε πιο ακριβείς, αν χρεώνονται ανεξάρτητα (από διαφορετικό προπληρωμένο λογαριασμό) οι συνεδρίες ηχητικής τηλεδιάσκεψης και γραπτής τηλεδιάσκεψης τότε πρέπει να ανακτηθεί το «session_id» της συνεδρίας

χρέωσης που αφορά την ηχητική τηλεδιάσκεψη από την οποία αποχώρησε ο «Rick». Αυτό επιτυγχάνεται με μία λειτουργία του «Context Repository» χρησιμοποιώντας το «session_id» της ηχητικής τηλεδιάσκεψης. Αφού ανακτηθεί το «session_id» της συνεδρίας χρέωσης που αφορά την ηχητική τηλεδιάσκεψη στο βήμα 5, ενημερώνεται η υπηρεσία χρέωσης στο βήμα 6. Στο βήμα 7 αφαιρείται από το «Context Repository» (και πιο συγκεκριμένα από τα στοιχεία που αφορούν την συνεδρία ηχητικής τηλεδιάσκεψης) η «εγγραφή» του συμμετέχοντα «Rick» ο οποίος δεν συμμετέχει πλέον στην συνεδρία ηχητικής τηλεδιάσκεψης. Έπειτα σύμφωνα με το σενάριο μας πρέπει να ενημερωθούν όλοι οι συμμετέχοντες της συνεδρίας γραπτής τηλεδιάσκεψης, με γραπτό μήνυμα για την αποχώρηση του «Rick» από την ηχητική τηλεδιάσκεψη. Η ανάκτηση του «session_id» της συνεδρίας γραπτής τηλεδιάσκεψης γίνεται παρόμοια με πριν από το «Context Repository» στο βήμα 8. Η κατάλληλη Ηλεκτρονική Υπηρεσία καλείται (με μία «εξωτερική ενέργεια) περνώντας τόσο το «process_id» της διαδικασίας XASC όσο και το «session_id» της συνεδρίας γραπτής τηλεδιάσκεψης στο βήμα 9. Τα βήματα 10,11,12 δείχνουν πώς μετατρέπεται η κλήση SOAP σε μηνύματα προς τους συμμετέχοντες τις συνεδρίας γραπτής τηλεδιάσκεψης.

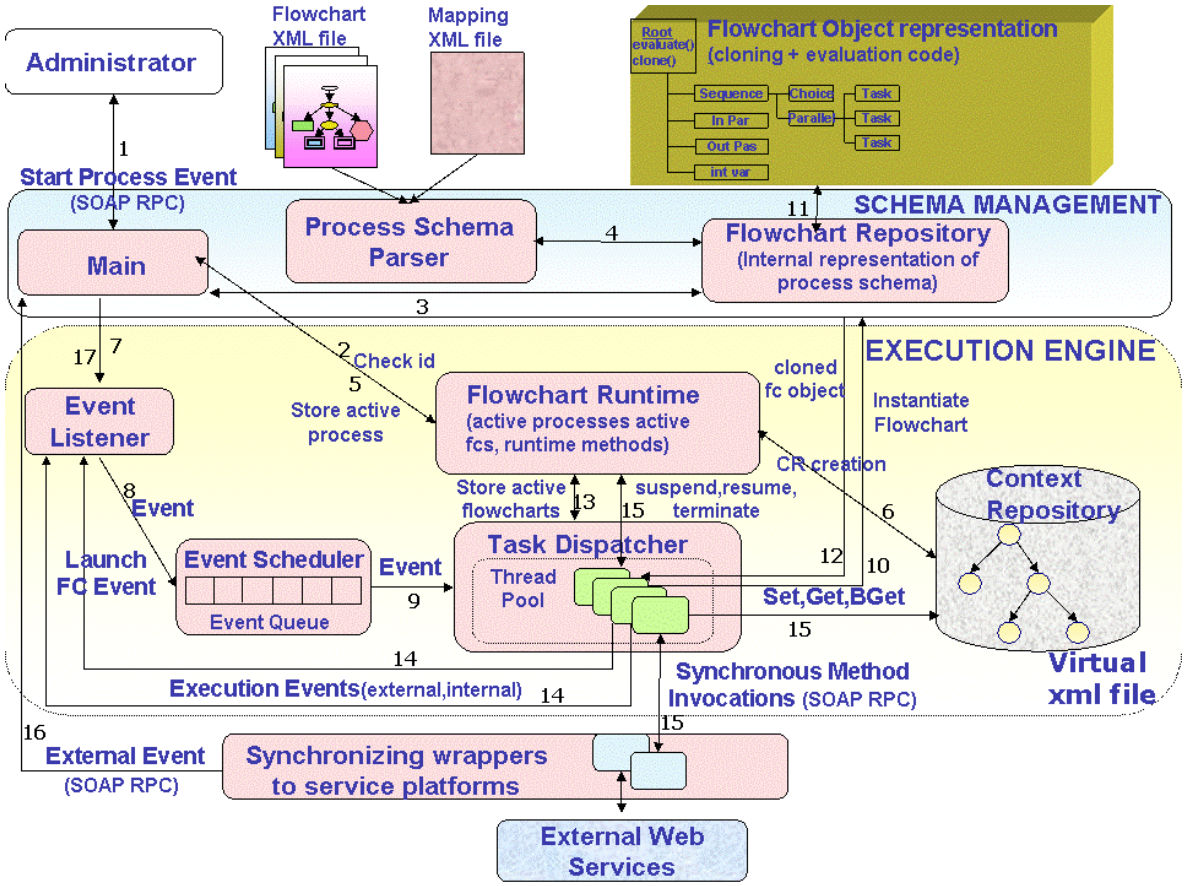
Το επόμενο «ασύγχρονο» γεγονός που συμβαίνει στο βήμα 13 είναι από την Ηλεκτρονική Υπηρεσία χρέωσης και ενημερώνει (με δημιουργία του γεγονότος που ενεργοποιεί το διάγραμμα ροής «out of money» ότι μία από τις δύο συνεδρίες χρέωσης της διαδικασίας αυτής δεν έχει πλέον χρήματα, δίνοντας ταυτόχρονα το «session_id» τόσο της συγκεκριμένης συνεδρίας χρέωσης όσο και της συνεδρίας τηλεδιάσκεψης που δεν έχει πλέον χρήματα. Στην **Εικόνα 3.27** υποθέτουμε ότι αυτή είναι η ηχητική τηλεδιάσκεψη. Από την στιγμή που τελείωσαν τα χρήματα της ηχητικής τηλεδιάσκεψης, αυτή θα σταματήσει αφού πρώτα ακούσουν όλοι οι συμμετέχοντες της ένα ηχητικό μήνυμα που θα τους ενημερώνει ότι τελείωσαν τα χρήματα. Αυτό γίνεται με την εκτέλεση μίας «εξωτερικής ενέργειας» επισυνάπτοντας το «session_id» της ηχητικής τηλεδιάσκεψης στα βήματα 14,15,16 και 17.Έπειτα στα βήματα 18 και 19 αφαιρούνται από το «Context Repository» οι «εγγραφές» που αφορούν τόσο την συνεδρία ηχητικής τηλεδιάσκεψης όσο και την συνεδρία χρέωσης της. Τέλος στα βήματα 20 έως 22 εκτελείται μία «εξωτερική ενέργεια» η οποία θα ζητήσει από το «gateway» να τερματίσει την συνεδρία ηχητικής τηλεδιάσκεψης. Παρόλαυτα η επικοινωνία μέσω της συνεδρίας γραπτής τηλεδιάσκεψης μπορεί να συνεχίζεται κανονικά όσο υπάρχουν χρήματα στον δικό της προπληρωμένο λογαριασμό.

# *Κεφάλαιο 4*

# *Η πλατφόρμα εκτέλεσης ενεργών διαγραμμάτων ροής AZTEC*

Μέρος της εργασίας αυτής περιλαμβάνει την σχεδίαση και την υλοποίηση της πλατφόρμας AZTEC για την σύνθεση και την εκτέλεση τόσο διακριτών (discrete) όσο και «session-oriented» Ηλεκτρονικών Υπηρεσιών. Η αρχική σχεδίαση και μία πρότυπη υλοποίηση της πλατφόρμας έλαβαν χώρα στα Bell Labs το καλοκαίρι του 2001. Μία από τις συνιστώσες τής πλατφόρμας AZTEC είναι η Μηχανή Εκτέλεσης (Execution Engine) των ενεργών διαγραμμάτων ροής που περιγράφηκαν διεξοδικά στην **Ενότητα 3.** Η Μηχανή Εκτέλεσης είναι «οδηγούμενη από γεγονότα» (event-driven) υπό την έννοια ότι η ενεργοποίηση ενός διαγράμματος ροής «προκαλείται» από την δημιουργία του αντίστοιχου «γεγονότος», είτε από κάποια εξωτερική Ηλεκτρονική Υπηρεσία είτε από κάποιο άλλο ενεργό διάγραμμα ροής με κάποια «ενέργεια γεγονότος». Η Μηχανή Εκτέλεσης μεταφράζει (interprets) τα «σχήματα διαδικασιών» (process schemas) και αλληλεπιδρά με εξωτερικές Ηλεκτρονικές Υπηρεσίες μέσω προγραμμάτων «wrappers» όπως φαίνεται στην **Εικόνα 4.1**. Υπάρχει η δυνατότητα ρητού ορισμού προτεραιοτήτων όσων αφορά την ενεργοποίηση διαγραμμάτων ροής στα πλαίσια της ίδιας διαδικασίας πού όπως αναφέρθηκε νωρίτερα είναι ένας από τους τρόπους αλληλεπίδρασης μεταξύ των ενεργών διαγραμμάτων ροής. Επίσης είναι δυνατή η ταυτόχρονη (concurrent) δημιουργία και εκτέλεση πολλαπλών στιγμιοτύπων τόσο της ίδιας διαδικασίας XASC όσο και διαφορετικών διαδικασιών.

Σημαντικό ρόλο παίζει και ο ανθρώπινος παράγοντας στην δημιουργία (creation), την διατήρηση (maintenance) και την παρακολούθηση (monitoring) της εκτέλεσης διαδικασιών στην πλατφόρμα AZTEC μέσω της συνιστώσας Διαχείρισης (Administration component). Στη συνέχεια του κεφαλαίου θα περιγράψουμε αναλυτικά την Μηχανής Εκτέλεσης και την συνιστώσα Διαχείρισης Σχήματος που χειρίζεται το «φόρτωμα» των «σχημάτων διαδικασίας» (process schemas) στην «Μηχανή Εκτέλεσης». Αφού περιγράψουμε την βασική λειτουργικότητα των διάφορων συνιστωσών της πλατφόρμας θα περιγράψουμε διαισθητικά την την «πορεία» που ακολουθεί ένα «γεγονός» (event) από την στιγμή που θα δημιουργηθεί μέχρι την στιγμή που θα εκτελεστεί.




**Εικόνα 4.1 Η αρχιτεκτονική της πλατφόρμας AZTEC**

Η πορεία αυτή περιγράφεται και με αριθμημένα βέλη (βήματα) στο διάγραμμα της **Εικόνας 4.1.**

## *4.1 Η Μηχανή Εκτέλεσης και η Διαχείριση Σχημάτων*

Καταρχήν να θυμηθούμε ότι το μοντέλο μας υποστηρίζει δύο ειδών «γεγονότα», τα «Γεγονότα Εκκίνησης» (Launch Events) και τα «Γεγονότα Εκτέλεσης» (Execution Events). Κάθε Γεγονός Εκκίνησης είναι «ζευγαρωμένο» (coupled) με ένα ενεργό διάγραμμα ροής, το οποίο ενεργοποιείται όταν δημιουργηθεί αυτό το γεγονός. Ένα Γεγονός Εκκίνησης μπορεί να ενεργοποιεί είτε το αρχικό (root) διάγραμμα ροής μίας διαδικασίας, οπότε πρακτικά ξεκινάει και ένα νέο στιγμιότυπο της διαδικασίας είτε ένα οποιοδήποτε διάγραμμα ροής στα πλαίσια μίας ήδη «ενεργής» (active) διαδικασίας. Υπάρχουν τρεις τρόποι δημιουργίας Γεγονότων Εκκίνησης. Ο πρώτος είναι από την συνιστώσα Διαχείρισης όποτε πρακτικά ο διαχειριστής του συστήματος ξεκινάει ένα στιγμιότυπο μίας διαδικασίας. Ο δεύτερος τρόπος είναι με κλήσεις SOAP RPC από εξωτερικές Ηλεκτρονικές Υπηρεσίες σε προγράμματα «wrappers» τα οποία πρακτικά παρέχουν την

διεπαφή του AZTEC με τον «έξω κόσμο». Μία τέτοια κλήση SOAP RPC σε ένα wrapper προκαλεί «εσωτερικά» του AZTEC την δημιουργία ενός «Γεγονότος Εκκίνησης». Ο τρίτος τρόπος δημιουργίας «Γεγονότων Εκκίνησης» είναι με την εκτέλεση «ενεργειών γεγονότων» από ήδη ενεργά διαγράμματα ροής.

Τα «Γεγονότα Εκτέλεσης» δημιουργούνται κάθε φορά που πρέπει να εκτελεστεί μία «ενέργεια» (Task) ενός διαγράμματος ροής, είτε αυτή είναι τύπου «εξωτερική» (external task), είτε «εσωτερική» (internal task), είτε «εσωτερική μέθοδος» (internal function),είτε «repository», είτε «αναμονής» (wait task), είτε «κενή» (empty task). Όταν πρέπει να εκτελεστεί μία «ενέργεια» δημιουργείται και προωθείται για εκτέλεση ένα «Γεγονός Εκτέλεσης» το οποίο πρακτικά «περιλαμβάνει» την ενέργεια αυτή. Χαρακτηρίσαμε νωρίτερα την Μηχανή Εκτέλεσης ως «οδηγούμενη από γεγονότα» (event-driven) ακριβώς για τον λόγο ότι όλες οι «ενέργειες» εκτελούνται με την μορφή γεγονότων, είτε αυτά είναι «Γεγονότα Εκκίνησης» είτε είναι «Γεγονότα Εκτέλεσης».

### 4.1.1 Η Συνιστώσα «Main»

Η «κεντρική» συνιστώσα της πλατφόρμας AZTEC («Main» στο διάγραμμα της **Εικόνας 4.1**) με την βοήθεια προγραμμάτων «wrappers» ορίζει την ορατή διεπαφή μίας διαδικασίας XASC προς εξωτερικές Ηλεκτρονικές Υπηρεσίες και εφαρμογές. Οι εφαρμογές αυτές μπορούν είτε να ξεκινήσουν μία νέα διαδικασία XASC, που εσωτερικά του AZTEC ισοδυναμεί με την δημιουργία του «γεγονότος» που ενεργοποιεί το αρχικό (root) διάγραμμα ροής μίας διαδικασίας, είτε να δημιουργήσουν ένα γεγονός που «απευθύνεται» σε ένα ήδη ενεργό στιγμιότυπο μίας διαδικασίας XASC. Η «Main» επικοινωνεί με διάφορες συνιστώσες ούτως ώστε να συσχετιστεί το «γεγονός» με κάποιο ενεργό στιγμιότυπο διαδικασίας ή αν το «γεγονός» αποσκοπεί στην δημιουργία ενός νέου στιγμιοτύπου διαδικασίας να φροντίσει ούτως ώστε να «φορτωθεί» το «σχήμα» αυτής της διαδικασίας στην «κύρια μνήμη» και να είναι έτοιμο προς εκτέλεση. Τέλος η «Main» ασχολείται με την προώθηση του γεγονότος που ελήφθη στην «ουρά εκτέλεσης».

### 4.1.2 Η συνιστώσα «Flowchart Runtime»

Η συνιστώσα «Flowchart Runtime» διατηρεί πληροφορίες της «κατάστασης» της Μηχανής Εκτέλεσης (runtime information) όπως τις διαδικασίες που είναι ενεργές και τα διαγράμματα ροής που είναι ενεργά κάθε στιγμή. Παράλληλα προσφέρει λειτουργίες

«τροποποίησης» αυτής της κατάστασης όπως οι «suspendAllFcs» και «resumeAllFcs» οι οποίες «παύουν» και «επανεκκινούν» όλα τα διαγράμματα ροής ενός συγκεκριμένου τύπου αντίστοιχα, και η «terminateProcess» η οποία τερματίζει μία ενεργή διαδικασία, αφαιρεί τα διαγράμματα ροής της από την λίστα με τα ενεργά διαγράμματα ροής και εν τέλει αφαιρεί την ίδια τη διαδικασία από την λίστα με τις ενεργές διαδικασίες. Η λίστα με τις ενεργές διαδικασίες στο «Flowchart Runtime» διατηρεί μαζί με τις ενεργές διαδικασίες και τα αναγνωριστικά τους (id), τόσο τα εσωτερικά (process id) που χρησιμοποιούνται στο εσωτερικό της «μηχανής εκτέλεσης» του AZTEC όσο και τα εξωτερικά (external process id) .τα οποία επισυνάπτονται στα «εισερχόμενα» μηνύματα για να μπορούν να συσχετισθούν με κάποιο ενεργό στιγμιότυπο διαδικασίας.

Το «Flowchart Runtime» αναλαμβάνει την δημιουργία μίας νέας διαδικασίας και την αποθήκευση της στην λίστα με τις ενεργές διαδικασίες, όταν αυτή προέρχεται από την λήψη του κατάλληλου «Γεγονότος Εκκίνησης» . Στα πλαίσια δημιουργίας ενός νέου στιγμιοτύπου μίας διαδικασίας δημιουργείται και ένα νέο στιγμιότυπο του «εικονικού» εγγράφου XML «Context Repository» το οποίο χρησιμοποιείται για τον διαμοιρασμό δεδομένων (σχετικών με τις διάφορες συνεδρίες που αναπτύσσονται) μεταξύ των διαγραμμάτων ροής της διαδικασίας. Προτού δημιουργήσει ένα νέο στιγμιότυπο μίας διαδικασίας, το «Flowchart Runtime» ελέγχει αν υπάρχει ήδη κάποια διαδικασία ενεργή με «external process id» ίδιο με αυτό που «φέρει» το γεγονός που αποσκοπεί στην εκκίνηση της διαδικασίας. Στην περίπτωση που ήδη υπάρχει διαδικασία ενεργή με το συγκεκριμένο «external process id», η δημιουργία νέας διαδικασίας δεν γίνεται, ειδάλλως ολοκληρώνεται κανονικά.

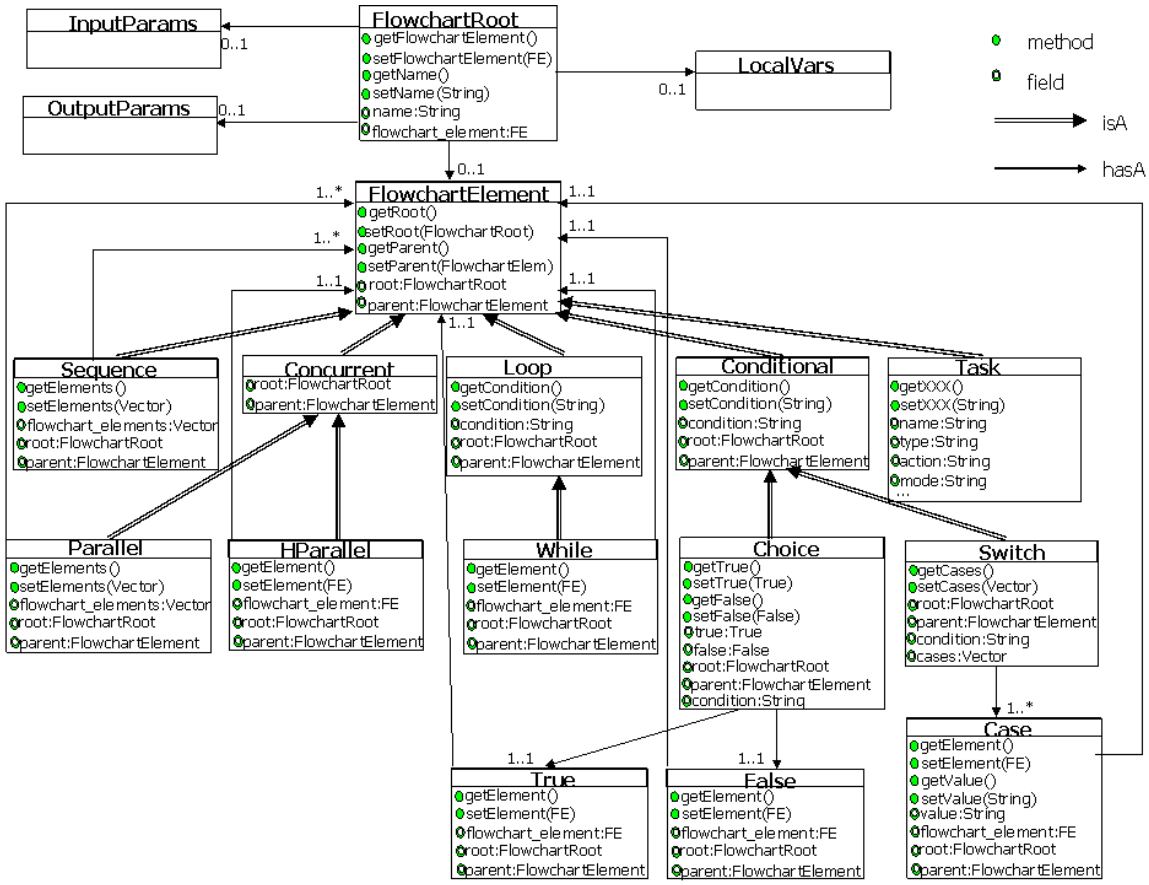
### 4.1.3 Η συνιστώσα «Flowchart Repository»

Τα κύρια συστατικά στοιχεία της συνιστώσας «Flowchart Repository» είναι η λίστα «Flowchart List» στην οποία αποθηκεύει τα διαγράμματα ροής της κάθε διαδικασίας (δηλαδή το «σχήμα» κάθε διαδικασίας) και ο πίνακας διασποράς (hash table) «Process Schemas» στον οποίο αποθηκεύονται «σχήματα» διαδικασιών (αντικείμενα τύπου «Flowchart List») με κλειδί το όνομα της διαδικασίας. Οι βασικές λειτουργίες που εκτελεί το «Flowchart Repository» είναι το φόρτωμα του «σχήματος» μίας διαδικασίας, και η δημιουργία -κατόπιν αίτησης- ενός κλώνου ενός διαγράμματος ροής «αποθηκευμένου» σε κάποια διαδικασία στην δομή «Process Schemas».

Την *πρώτη* φορά που δημιουργείται ένα στιγμιότυπο μίας διαδικασίας, πρακτικά «φορτώνεται» το σχήμα της διαδικασίας και αποθηκεύεται στη δομή «Process Schemas»

του «Flowchart Repository». Για τα υπόλοιπα στιγμιότυπα της συγκεκριμένης διαδικασίας που θα δημιουργηθούν από εκείνο το σημείο και μετά, το σχήμα της διαδικασίας δεν θα χρειαστεί να «φορτωθεί» ξανά. Τα διαγράμματα ροής μίας διαδικασίας είναι αποθηκευμένα με την μορφή αρχείων XML που ακολουθούν το συντακτικό της γλώσσας XASC, δημιουργημένα από τον διαχειριστή (administrator) του συστήματος. Κατά σύμβαση, το όνομα του αρχείου XML στο οποίο είναι αποθηκευμένο ένα διάγραμμα ροής, έχει όνομα ίδιο με το διάγραμμα ροής που περιλαμβάνει. Τα σύνολο των αρχείων XML που αποτελούν το «σχήμα» μίας συγκεκριμένης διαδικασίας είναι αποθηκευμένα σε ένα κατάλογο το όνομα του οποίου είναι ίδιο με το όνομα της διαδικασίας. Με αυτό τον τρόπο «σχήματα» διαφορετικών διαδικασιών είναι «αποθηκευμένα» σε διαφορετικούς καταλόγους με ονόματα ίδια με αυτά των αντίστοιχων διαδικασιών. Όλοι αυτοί οι κατάλογοι βρίσκονται μέσα σε ένα κατάλογο «ρίζα» (FLOWCHART_REPOSITORY_ROOT) ο οποίος καθορίζεται από τον διαχειριστή του συστήματος. Μέσα σε κάθε κατάλογο, που περιλαμβάνει το «σχήμα» μίας διαδικασίας, υπάρχει ένα αρχείο (list) το οποίο διατηρεί τα ονόματα των αρχείων XML που διατηρούν τα διαγράμματα ροής της διαδικασίας. Το «γεγονός» που «αποσκοπεί» στην δημιουργία ενός νέου στιγμιοτύπου μίας διαδικασίας περιλαμβάνει στα δεδομένα που μεταφέρει (έχουμε μιλήσει νωρίτερα για την έννοια των typed events που μεταφέρουν δεδομένα) και το όνομα της διαδικασίας που επιθυμεί να στιγμιοποιήσει (instantiate). Χρησιμοποιώντας αυτό το όνομα, το «Flowchart Repository» ξέρει από ποιόν κατάλογο πρέπει να «φορτώσει» τα αρχεία XML, δηλαδή το «σχήμα» (schema) της διαδικασίας αφού πρώτα έχει ανακτήσει τα ονόματα των αρχείων XML αυτών, από το αρχείο «list» στο συγκεκριμένο κατάλογο.

Τα διαγράματα ροής αποθηκεύονται (αφου πρώτα μετατραπούν κατάλληλα απο την συνιστώσα «Parser») στην «Flowchart List» με την μορφή ιεραρχίας αντικειμένων Java. Στην **Εικόνα 4.2** μπορούμε να δούμε διαγραμματικά τούς αφαιρετικους τύπους δεδομένων (ΑΤΔ) της αντικειμενοστρεφούς αναπαράστασης ενός διαγράμματος ροής στο εσωτερικό της Μηχανής Εκτέλεσης της πλατφόρμας AZTEC. Όταν έχει ολοκληρωθεί αυτή η διαδικασία και η «Flowchart List» περιλαμβάνει πλέον όλα τα διαγράμματα ροής της διαδικασίας με τη μορφή αντικειμένων αυτή αποθηκεύεται όπως αναφέρθηκε νωρίτερα στον πίνακα διασποράς «Process Schemas», με κλειδί το όνομα της διαδικασίας. Οποιοδήποτε άλλο «γεγονός» «αποσκοπεί» από εδώ και στο εξής στην δημιουργία ενός στιγμιοτύπου αυτής της διαδικασίας δεν θα χρειαστεί να ξαναφορτώσει τα διαγράμματα ροής από την αρχή.




**Εικόνα 4.2 Διάγραμμα εξειδικεύσεων και εξαρτήσεων του (ΑΤΔ) που αναπαριστά ένα διάγραμμα ροής στο AZTEC**

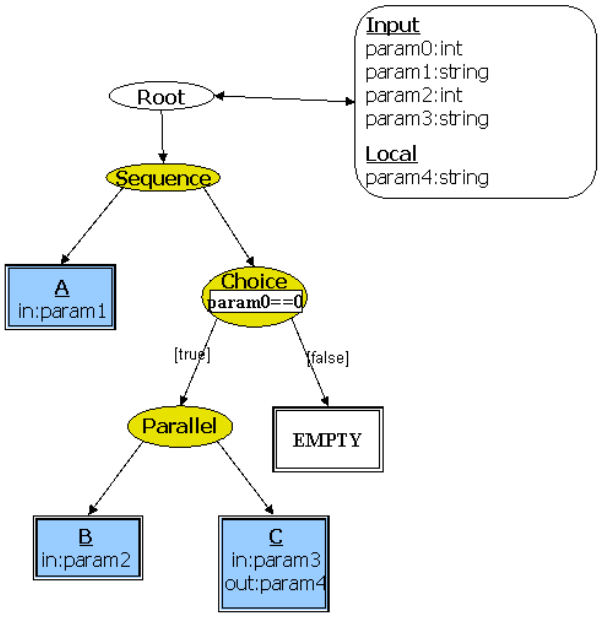
Τα διαγράμματα ροής που έχουν αποθηκευτεί για μία διαδικασία στο «Process Schemas» θα αποτελούν «πρότυπα» (patterns) για τα διαγράμματα ροής που πραγματικά θα εκτελούνται. Συγκεκριμένα, όταν «συμβαίνει» ένα «γεγονός» στα πλαίσια μίας διαδικασίας, το σχήμα αυτής της διαδικασίας ανακτάται από το «Process Schemas», το κατάλληλο διάγραμμα ροής ανακτάται χρησιμοποιώντας το όνομα του «γεγονότος», και ένας «***κλώνος***» αυτού του διαγράμματος ροής επιστρέφεται ο οποίος και πραγματικά εκτελείται. Σε περίπτωση που δεν γινόταν αυτή η «κλωνοποίηση», τότε την επόμενη φορά που θα δημιουργούταν ένα στιγμιότυπο της ίδιας διαδικασίας, θα έπρεπε να ξαναφορτωθεί το σχήμα της διαδικασίας, μιας και τα αντικείμενα Java που αναπαριστούν τα διαγράμματα ροής στο «Process Schemas» θα είχαν «εκτελεστεί» με τις «παραμέτρους» της προηγούμενης εκτέλεσης, και αυτά τα δεδομένα θα είχαν παραμείνει στις εσωτερικές δομές των αντικειμένων.

### 4.1.3.1 Εσωτερική αναπαράσταση των διαγραμμάτων ροής στο AZTEC

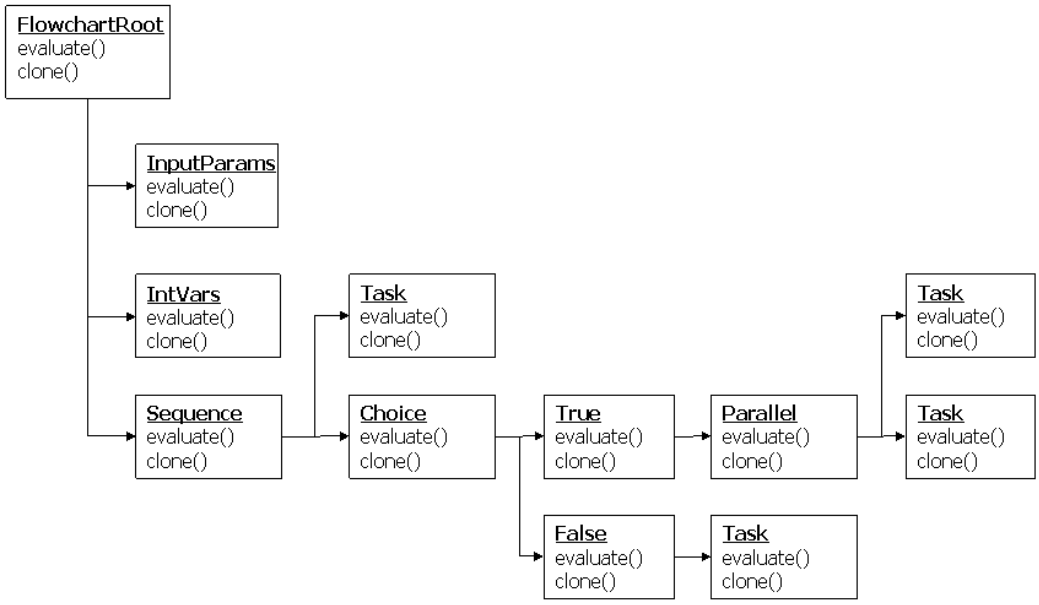
Είδαμε νωρίτερα στην **Εικόνα 4.2** τον αφαιρετικό τύπο δεδομένων που περιγράφει την αντικειμενοστρεφή αναπαράσταση ενός διαγράμματος ροής στο «εσωτερικό» του AZTEC Στην **Εικόνα 4.3 α)** βλέπουμε ένα απλοϊκό διάγραμμα ροής χωρίς κάποια ιδιαίτερη λειτουργικότητα και στο **β)** βλέπουμε την αντικειμενοστρεφή αναπαράσταση του στο AZTEC. Η ιεραρχία αντικειμένων της **Εικόνας 4.3 β)** είναι πρακτικά ένα στιγμιότυπο του ΑΤΔ της **Εικόνας 4.2.** Βλέπουμε ότι κάθε στοιχείο της XML αναπαράστασης του διαγράμματος ροής αναπαρίσταται με ένα αντικείμενο Java στο οποίο τα διάφορα «χαρακτηριστικά» (attributes) αναπαρίστανται ως πεδία (fields) του αντικειμένου. Τα αντικείμενα που αναπαριστούν στοιχεία όπως το «Sequence» ή το «Parallel» που έχουν μία λίστα από «ύπο-στοιχεία» στο διάγραμμα ροής, έχουν στα πεδία τους μία λίστα από υπό-αντικείμενα τα οποία θα αναπαριστούν αυτά τα «ύπο-στοιχεία». Η ιεραρχία αντικειμένων της **Εικόνας 4.3 β)** δεν είναι πλήρης χάριν συντομίας. Για να ήταν πλήρης θα έπρεπε να επιδεικνύεται για παράδειγμα ότι το αντικείμενο «InputParams» έχει μία λίστα με «υπό-αντικείμενα» τύπου «Param», και έπειτα ότι κάθε αντικείμενο τύπου «Param» έχει ένα «υπό-αντικείμενο» τύπου είτε «ValueString», είτε «ValueInt», είτε «ValueFloat», είτε «ValueBoolean», είτε «ValueSet» και πάει λέγοντας. Το αντικείμενο το οποίο είναι αποθηκευμένο στα διάφορα «Flowchart List» που περιλαμβάνει το «Process Schemas» είναι το «Flowchart Root» για κάθε διάγραμμα ροής. Όταν το «Flowchart Repository» θέλει να δημιουργήσει ένα «κλώνο» ενός διαγράμματος ροής, θέλει να δημιουργήσει ένα κλώνο ολόκληρης της ιεραρχίας αντικειμένων, από τη ρίζα («Flowchart Root») έως τα φύλλα («Tasks».) Για αυτό τον λόγο η κλωνοποίηση γίνεται αναδρομικά και κάθε αντικείμενο δημιουργεί ένα «κλώνο» του εαυτού του και καλεί τις μεθόδους **clone()** όλων των «υπό-αντικειμένων» του επιτυγχάνοντας έτσι *βαθιά «κλωνοποίηση»* (deep cloning) του αντικειμένου «Flowchart Root».

## 4.1.4 Η συνιστώσα «Parser»

Σε αυτή την ενότητα θα περιγράψουμε αναλυτικά την λειτουργικότητα της συνιστώσας «Parser» και της εσωτερικής, «αντικειμενοστρεφούς» (object-oriented) αναπαράστασης των στοιχείων της XASC σε αντικείμενα Java. Ο «Parser» χρησιμοποιεί ένα υπόβαθρο «δέσμευσης δεδομένων» (data binding framework) για την «εξαγωγή» (extraction) δεδομένων από κείμενα XML και την αναπαράσταση τους στα σχετικά αντικείμενα Java.




**α)Ένα απλό διάγραμμα ροής**




**β)Η αντικειμενοστρεφής αναπαράσταση του διαγράμματος ροής**

**Εικόνα 4.3**

Για την εσωτερική αναπαράσταση των διαγραμμάτων ροής στο AZTEC μία πολύ σημαντική απαίτηση είναι η δυνατότητα εύκολου χειρισμού όλων των τύπων δεδομένων του σχήματος XML τόσο των απλών όσο και των πολύπλοκων και η μετατροπή τους συχνά σε αντικείμενα της επιλογής μας (custom objects). Επίσης προτιμήθηκε μία αντικειμενοστρεφής (object) αναπαράσταση η οποία θα επιτρέπει την «αποτίμηση» (evaluation) του διαγράμματος ροής χωρίς την προγραμματιστική πολυπλοκότητα που

εισάγει η διάσχιση μοντέλων κειμένων. Η αποφυγή της διατήρησης τέτοιων κειμένων στην μνήμη καθ' όλη την διάρκεια της εκτέλεσης είναι επίσης επιθυμητή. Τελευταίο αλλά ίσως και σημαντικότερο είναι ότι επιθυμούμε την μεγαλύτερη δυνατή ευελιξία και «χαλαρή σύνδεση» (loose coupling) ανάμεσα στο «σχήμα» XML της γλώσσας XASC και του αντικειμενοστρεφούς εσωτερικού μοντέλου αναπαράστασης, μιας και η γλώσσα είναι ακόμα υπό ανάπτυξη και υπόκειται συχνών αλλαγών. Επίσης με την «ανεξαρτησία» αυτή η πλατφόρμα AZTEC θα μπορούσε να χρησιμοποιηθεί για την εκτέλεση διαδικασιών ορισμένων σε κάποια άλλη γλώσσα, με την δημιουργία του κατάλληλου «ορισμού αντιστοίχησης» (mapping definition).

#### 4.1.4.1 Μοντέλα δέσμευσης δεδομένων ή μοντέλα κειμένων;

Η προσέγγιση «δέσμευσης δεδομένων» (data binding) παρέχει ένα απλό και ευθύ τρόπο για την χρήση XML σε εφαρμογές Java. Χρησιμοποιώντας «δέσμευση δεδομένων» η εφαρμογή μπορεί να αγνοήσει την πραγματική δομή των κειμένων XML αντί να δουλεύει με το περιεχόμενο (data content) ολόκληρων τέτοιων κειμένων ελαττώνοντας σημαντικά την προγραμματιστική πολυπλοκότητα που εισάγει ο χειρισμός ολόκληρων κειμένων XML στην μνήμη.

Η «δέσμευση δεδομένων» παρέχει και άλλα πλεονεκτήματα πλην της προγραμματιστικής απλότητας. Απο την στιγμή που αφαιρεί (abstracts) τις λεπτομέρειες των κειμένων (documents) η «δέσμευση δεδομένων» απαιτεί λιγότερη μνήμη απ' ότι μία προσέγγιση με μοντέλα κειμένων (document model) όπως το DOM[DOM] ή το JDOM[JDOM] που δουλεύουν με κείμενα (documents) στην μνήμη. Παράλληλα η προσέγγιση με «δέσμευση δεδομένων» παρέχει γρηγορότερη προσπέλαση στα δεδομένα στο πρόγραμμα απ' ότι η προσέγγιση με μοντέλα κειμένων (document models) αφού δεν είναι απαραίτητη η διάσχιση της δομής του αντικειμένου για να ανακτηθούν τα δεδομένα. Τέλος «ειδικοί» (special) τύποι δεδομένων όπως αριθμοί ή ημερομηνίες μπορούν να μετατραπούν σε εσωτερικές «αντικειμενοστραφείς αναπαραστάσεις» (object representations) αντί να αντιμετωπίζονται σαν απλό κείμενο (plain text). Το πρόβλημα αυτό γίνεται ακόμα πιο περίπλοκο όταν θέλουμε να χειριστούμε πολύπλοκους τύπους δεδομένων (complex data types) των οποίων η ανάκτηση από μοντέλα κειμένων (document models) συνιστά μεγάλη προγραμματιστική πολυπλοκότητα.

Υπάρχουν αρκετά «υπόβαθρα δέσμευσης δεδομένων» (data binding frameworks) τα οποία παράγουν (generate) κώδικα γλώσσας Java από γραμματικές κειμένων XML

(XML document grammars). Τα σημαντικότερα από αυτά είναι τα JAXB[JAXB], Castor[CASTOR], JBind[JBIND], Quick[QUICK] και Jeus[ZEUS]. Όλα από τα παραπάνω είναι μη εμπορικά προϊόντα και όλα εκτός του JAXB μπορούν να χρησιμοποιηθούν τόσο σε εργασίες ανοιχτού κώδικα (open source projects) όσο και σε ιδιόκτητες «επικερδείς» εργασίες (proprietary projects). Τα JAXB, Castor και JBind προσφέρουν «δημιουργία κώδικα» (code generation) από περιγραφές «σχήματος» (Schema descriptions) κειμένων XML, ενώ τα Quick και Zeus υλοποιούν την «δημιουργία κώδικα» από περιγραφές γραμματικών DTD. Το Castor και το Quick επιπρόσθετα, παρέχουν την αντιστοίχηση (mapping) υπαρχόντων κλάσεων σε κείμενα XML σαν εναλλακτική της «αυτόματης δημιουργίας κώδικα».

### 4.1.4.2 Δέσμευση με αντιστοίχηση ή αυτόματη δημιουργία κώδικα;

Η «αυτόματη δημιουργία κώδικα» (code generation) είναι μία προσέγγιση που μπορεί να έχει πολλά πλεονεκτήματα σε ορισμένες περιπτώσεις, αλλά σε άλλες να μην είναι επιθυμητή. Χρησιμοποιώντας «αυτόματα δημιουργημένο» κώδικα εξασφαλίζεται ότι τα αντικείμενα δεδομένων (data objects) που δημιουργούνται είναι «κατάλληλα συνδεδεμένα» (properly linked) με τα κείμενα XML, αντίθετα με την προσέγγιση «δέσμευση με αντιστοίχηση» (mapped binding) όπου ο προγραμματιστής πρέπει να εξασφαλίσει ότι έχει καλύψει όλες τις πτυχές των δομών (structures). Όταν ένα «σχήμα» (Schema) δημιουργείται μπορούν να χρησιμοποιηθούν οι «πληροφορίες τύπου» (type information) που παρέχονται από την γραμματική για να δημιουργηθεί ο κατάλληλος κώδικας για τους τύπους δεδομένων (data types).

Παρόλ'αυτά η «αυτόματη δημιουργία κώδικα» έχει και μειονεκτήματα. Καταρχάς δημιουργεί «στενή εξάρτηση» (tight coupling) ανάμεσα στην δομή των αντικειμένων της εφαρμογής (application objects) και της δομής του κειμένου XML. Έπειτα μας περιορίζει στο να δουλεύουμε με απλές «κλάσεις δεδομένων» (simple data classes) αντί για πραγματικές «κλάσεις αντικειμένων» (object classes), και πιθανώς περιορίζει την ευελιξία (flexibility) στην δυνατότητα για μετασχηματισμούς κατ'επιλογή (custom

transformations) των δεδομένων κατά τις διαδικασίες του marshalling [3]και του unmarshalling[4].

Γενικά οι «δεσμεύσεις με αντιστοίχηση» όπως αυτές που παρέχουν το Castor και το Quick παρέχουν μεγαλύτερη ευελιξία από την «δημιουργία κώδικα». Δουλεύουν με «πραγματικές κλάσεις αντικειμένων» (true object classes) συνδυάζοντας τόσο δεδομένα όσο και «συμπεριφορά». Επιπρόσθετα δίνεται η δυνατότητα για «αποδέσμευση» (decoupling) μέχρι ενός βαθμού, των «κλάσεων αντικειμένων» (object classes) από το πραγματικό κείμενο XML. Μικρές αλλαγές στην δομή του κειμένου XML αντιμετωπίζονται αλλάζοντας τον ορισμό της «αντιστοίχησης» (mapping definition) αντί να απαιτούνται αλλαγές στον κώδικα της εφαρμογής. Σαν μειονέκτημα μπορεί να καταλογιστεί ότι ο ορισμός της αντιστοίχησης πρέπει να γίνει «χειροκίνητα» (manually) από τον προγραμματιστή.

Οι απαιτήσεις μας, υποδεικνύουν ρητά ότι το «υπόβαθρο» που τις ικανοποιεί σχεδόν στο σύνολο τους είναι το Castor. Ένα επιπρόσθετο κίνητρο χρήσης του Castor είναι ότι το [XMLJ] δείχνει ότι επιτυγχάνει την καλύτερη διαχείριση μνήμης από όλα τα προαναφερθέντα «υπόβαθρα».

Πριν προχωρήσουμε στην παρουσίαση του Castor ας θυμηθούμε την μορφή του «ΑΤΔ» που αναπαριστά ένα διάγραμμα ροής στο AZTEC. Όπως είδαμε στην **Εικόνα 4.2** το «ριζικό» αντικείμενο «Flowchart Root» έχει ένα υπό-αντικείμενο τύπου «Flowchart Element» και τα υπό-αντικείμενα «InputParams», «OutputParams» και «IntVars» τα οποία αναπαριστούν τις παραμέτρους του διαγράμματος ροής και δεν αναπαριστάθηκαν στο σχήμα για λόγους συντομίας. Στο σχήμα βλέπουμε ότι το «αφαιρετικό» αντικείμενο «Flowchart Element» αναπαριστά οποιοδήποτε από τα στοιχεία της XASC. Καθένα από αυτά τα αντικείμενα έχει με την σειρά του είτε ένα («HParallel», «While») είτε μια λίστα («Sequence», «Parallel») από υπό-αντικείμενα τύπου «Flowchart Element». Ακόμα υπάρχουν και κάποια αντικείμενα όπως το «Choice» και το «Switch» τα οποία έχουν υπό-αντικείμενα τύπου «True»/«False» και «Case» αντίστοιχα τα οποία με τη σειρά τους έχουν υπό-αντικείμενα τύπου «Flowchart

[3] Marshalling ονομάζεται η διαδικασία της δημιουργίας της XML αναπαράστασης ενός αντικειμένου στην μνήμη

[4] Unmarshalling ονομάζεται η διαδικασία της δημιουργίας ενός αντικειμένου στην μνήμη( και πιθανώς μίας ιεραρχίας απο αντικείμενα) απο μία XML αναπαράσταση.

Element». Να σημειωθεί ότι το «HParallel» έχει ένα μόνο υπό-αντικείμενο τύπου «Flowchart Element» στον ορισμό του, αλλά αυτό το υπό-αντικείμενο εκτελείται πολλές φορές παράλληλα.

Καταυτό τον τρόπο αναπαρίστανται όλοι οι πιθανοί συνδυασμοί μεταξύ των στοιχείων της XASC με οποιοδήποτε βαθμό αναδρομής. Ο αφαιρετικός τύπος δεδομένων που αναπαριστά τις παραμέτρους παρουσιάζεται στο Παράρτημα 1.

### 4.1.4.3 Castor

Το Castor επιτρέπει τον ορισμό της marsalling/unmarshalling συμπεριφοράς του σε ένα «αρχείο αντιστοίχησης» (mapping file). Εμείς ενδιαφερόμαστε μόνο για την unmarshalling συμπεριφορά του, δηλαδή την μετατροπή XML κειμένων σε αντικείμενα Java και αυτήν θα περιγράψουμε εδώ. Το «αρχείο αντιστοίχησης» είναι ένα κείμενο XML (XML document). Το κείμενο αυτό περιγράφει πώς οι ιδιότητες (properties) ενός αντικειμένου μεταφράζονται σε XML (στοιχεία (elements) ή χαρακτηριστικά (attributes)) και αντίστροφα. Ένα «περιορισμός» είναι ότι το «αρχείο αντιστοίχησης» πρέπει να περιγράφει με *σαφήνεια* και χωρίς διφορούμενο τρόπο (unambiguously) τον τρόπο με τον οποίο ένα στοιχείο (element) ή χαρακτηριστικό (attribute) XML, θα μεταφραστεί στο μοντέλο αντικειμένων (object model) κατά την διαδικασία του unmarshalling. Ένα παράδειγμα αρχείου αντιστοίχησης είναι αυτό της **Εικόνας 4.4 γ)**

Το «αρχείο αντιστοίχησης» περιγράφει για κάθε αντικείμενο, πώς τα «πεδία» του αντιστοιχίζονται στην XML. Ένα «πεδίο» είναι μία «αφαίρεση» για μία «ιδιότητα» (property) ενός αντικειμένου. Μπορεί να ανταποκρίνεται είτε άμεσα σε μία μεταβλητή κάποιας κλάσης (public class variable) είτε έμμεσα σε κάποια ιδιότητα (property) μέσω των αντίστοιχών μεθόδων «πρόσβασης» (accessors) get/set. Όταν κάποια πληροφορία απαραίτητη για την δημιουργία της αντιστοίχησης δεν βρεθεί στο «αρχείο αντιστοίχησης» τότε χρησιμοποιείται η προεπιλεγμένη (default) συμπεριφορά του Castor η οποία χρησιμοποιώντας το Java Reflection API «εξετάζει» τα αντικείμενα για να «αποφασίσει» τι να κάνει. Για να μην μακρηγορήσουμε δεν θα περιγράψουμε την προεπιλεγμένη συμπεριφορά του Castor και πώς συμπεριφέρεται όταν δεν βρίσκει κάποια από τις πληροφορίες αντιστοίχησης.

Το «ριζικό» στοιχείο του «αρχείου αντιστοίχησης» είναι το «mapping» το οποίο περιλαμβάνει μία περιγραφή της αντιστοίχησης με το στοιχείο «description», δυνατότητα επαναχρησιμοποίησης «αρχείων αντιστοίχησης» με το στοιχείο «include»

και ένα στοιχείο «class» για κάθε κλάση Java (αντικείμενο) για την οποία επιθυμούμε να δώσουμε πληροφορίες αντιστοίχησης. Στο αρχείο της **Εικόνας 4.4 γ)** περιλαμβάνεται μόνο το στοιχείο «class» (το οποίο είναι και το μόνο υποχρεωτικό στοιχείο) για λόγους απλότητας.

Το στοιχείο «class» περιλαμβάνει τα χαρακτηριστικά (attributes) «name» και «extends» για να περιγράψει το όνομα της κλάσης, και το όνομα της υπέρ-κλάσης την οποία επεκτείνει (αν υπάρχει τέτοια). Αντίστοιχα περιλαμβάνει το στοιχείο «map-to» το οποίο αντιστοιχίζει την κλάση με κάποιο στοιχείο XML. Το στοιχείο «map-to» χρησιμοποιείται μόνο για την «αντιστοίχηση» του «ριζικού» στοιχείου (root element) του κειμένου XML. Καθεμία από τις «ιδιότητες» (properties) του αντικειμένου Java περιγράφονται με ένα στοιχείο «field». Στην περίπτωση αντιστοίχησης της Εικόνας 4.5 παρέχουμε την αντιστοίχηση μόνο του αντικειμένου «Flowchart Foot» στο στοιχείο «flowchart root» της XASC, το οποίο δεν επεκτείνει καμία κλάση.

Το στοιχείο «map-to» περιλαμβάνει το χαρακτηριστικό (attribute) «xml» το οποίο αναπαριστά το όνομα του στοιχείου XML στο οποίο αντιστοιχεί η κλάση και στοιχεία για τον καθορισμό του URI και του «namespace» αυτού του στοιχείου XML. Στο παράδειγμα μας το αντικείμενο «Flowchart Root» αντιστοιχίζεται με το «map-to» στο στοιχείο «flowchart root» της XASC.

Το στοιχείο «field» περιλαμβάνει το χαρακτηριστικό «name» το οποίο σε περίπτωση που το χαρακτηριστικό «direct» έχει τιμή «true» πρέπει να έχει τιμή μία από τις «δημόσιες μεταβλητές» (public variables) της κλάσης. Σε περίπτωση που το χαρακτηριστικό «direct» έχει την τιμή «false» το «name» θα χρησιμοποιηθεί για να εξαχθεί το όνομα των μεθόδων πρόσβασης get/set το οποίο βέβαια μπορεί να δοθεί ρητά με τα χαρακτηριστικά «get-method» και «set-method». Το χαρακτηριστικό «type» περιγράφει τον τύπο του «πεδίου» (field). Σε περίπτωση που το «πεδίο» είναι μία «συλλογή» το οποίο περιγράφεται με το χαρακτηριστικό «collection» το χαρακτηριστικό «type» περιγράφει τον τύπο του αντικειμένου μέσα στην συλλογή. Το χαρακτηριστικό «type» μπορεί να πάρει τιμή οποιουδήποτε έγκυρου τύπου (object, string, integer κ.τ.λ) ενώ το χαρακτηριστικό «collection» μπορεί να πάρει μία από τις τιμές (array, arraylist, vector, hashtable, collection, set, map). Το χαρακτηριστικό «collection» πρακτικά περιγράφει την δομή στην οποία θα αποθηκευτούν στο αντικείμενο τα δεδομένα του πολύπλοκου τύπου (complex type) του κειμένου XML. Το όνομα του στοιχείου XML στο οποίο αντιστοιχίζεται αυτό το «πεδίο» του αντικειμένου προσδιορίζεται με το στοιχείο «bind-xml». Στο παράδειγμα μας όλα τα «πεδία» του αντικειμένου «Flowchart Root»

αναπαρίστανται άμεσα με «public» μεταβλητές γι'αυτό και το χαρακτηριστικό «direct» έχει τιμή «true». Τα πεδία αυτά είναι τα «input_params», «output_params», «int_vars», «flowchart_element» και «name» τα οποία έχουν τους τύπους που φαίνονται στα αντίστοιχα χαρακτηριστικά «field» της **Εικόνας 4.4 γ).** Κανένα από αυτά τα πεδία δεν έχει τύπο «συλλογής» γι'αυτό και δεν εμφανίζεται το χαρακτηριστικό «collection». Αν παρουσιάζαμε την αντιστοίχηση για παράδειγμα του αντικείμένου «Sequence» το οποίο έχει το πεδίο «flowchart elements» που είναι πρακτικά μία λίστα από αντικείμενα τύπου «Flowchart Element» το χαρακτηριστικό «collection» θα είχε την τιμή «list».

Το στοιχείο «bind-xml» περιλαμβάνει το χαρακτηριστικό «name» που περιγράφει το όνομα το στοιχείου XML το οποίο αντιστοιχεί σε αυτό το «πεδίο» (field) του αντικειμένου. Το χαρακτηριστικό «type» περιγράφει τον τύπο του «σχήματος» XML (XML Schema type) του στοιχείου αυτού ενώ το χαρακτηριστικό «node» το οποίο παίρνει τιμή «attribute», «element» ή «text» περιγράφει τον τύπο του στοιχείου αυτού στο κείμενο XML. Για παράδειγμα το πεδίο «input_params» του αντικειμένου «Flowchart Element» αντιστοιχίζεται στο στοιχείο XML «input_params» του στοιχείου «flowchart_root» το οποίο είναι «XML element». Αντίστοιχα το πεδίο «name» του αντικειμένου «Flowchart Root» αντιστοιχίζεται στο στοιχείο XML «name» του στοιχείου «flowchart_root» το οποίο είναι «XML attribute»..

Το Castor «αρχικοποιείται» με ένα αρχείο αντιστοίχησης σαν αυτό της **Εικόνας 4.4 γ)** το οποίο πρέπει να παρέχει μία «έγκυρη» αντιστοίχηση σε αρχεία που συμμορφώνονται στο σχήμα της Εικόνας **4.4 α)** με στιγμιότυπα της κλάσης (δηλαδή αντικείμενα) της **Εικόνας 4.4 β).** Παίρνοντας σαν είσοδο ένα αρχείο XML οποίο συμμορφώνεται στο προαναφερθέν «σχήμα» το Castor επιστρέφει το «ριζικό» αντικείμενο της ιεραρχίας αντικειμένων που ξεκινάει από το «Flowchart Root» (Μην ξεχνάμε ότι τα παραδείγματα της **Εικόνας 4.4** είναι μικρά κομμάτια από το συνολικό σχήμα, το συνολικό ΑΤΔ και το συνολικό αρχείο αντιστοίχησης αντίστοιχα). Το «»ριζικό» αυτό στοιχείο ο «Parser» το επιστρέφει στο «Flowchart Repository» που με τη σειρά του το αποθηκεύει.

Το Castor παίρνει σαν «είσοδο» ένα αρχείο XML σαν αυτό της **Εικόνας 4.4 β**) το οποίο συμμορφώνεται (conforms) με ένα «σχήμα» XML σαν αυτό της **Εικόνας 4.4 α)** και ένα αρχείο αντιστοίχησης σαν αυτό της **Εικόνας 4.4 γ)** και επιστρέφει το ριζικό

```
<xs:element name="flowchart_root">
        <xs:complexType>
                <xs:sequence>
                        <xs:element ref="input_params" minOccurs="0" maxOccurs="1"/>
                        <xs:element ref="output_params" minOccurs="0" maxOccurs="1"/>
                        <xs:element ref="int_vars" minOccurs="0" maxOccurs="1"/>
                        <xs:group ref="f_element" minOccurs="0" maxOccurs="1"/>
                </xs:sequence>
                <xs:attribute name="name" type="xs:string"/>
        </xs:complexType>
</xs:element>
<xs:group name="f_element">
        <xs:choice>
                <xs:element ref="task"/>
                <xs:element ref="sequence"/>
                <xs:element ref="choice"/>
                <xs:element ref="parallel"/>
                <xs:element ref="hparallel"/>
                <xs:element ref="while_do"/>
                <xs:element ref="switch"/>
        </xs:choice>
</xs:group>
```

**α) Η περιγραφή σε «Σχήμα» XML του στοιχείου «flowchart_root» ενός διαγράμματος ροής**

```
public class FlowchartRootImpl {

        public InputParams getInputParams() {return input_params;}

        public void setInputParams(InputParams ip) {input_params =(InputParamsImpl)ip;}

        public OutputParams getOutputParams() {return output_params;}

        public void setOutputParams(OutputParams op) {output_params = (OutputParamsImpl)op;}

        public IntVars getIntVars() {return int_vars;}

        public void setIntVars(IntVars iv) {int_vars = (IntVarsImpl)iv;}

        public FlowchartElement getFlowchartElement() {return flowchart_element;}

        public void setFlowchartElement(FlowchartElement fe) {flowchart_element = fe;}

        public String getName(){return _name;}

        public void setName(String name){_name = name;}

        public InputParamsImpl input_params;
        public OutputParamsImpl output_params;
        public IntVarsImpl int_vars;
        public FlowchartElement flowchart_element;
        public String _name;
}
```

**β) Ο ορισμός της κλάσης που περιγράφει το αντικείμενο «Flowchart Root»**

```
<mapping>
        <class name="aztec.elements.impl.FlowchartRootImpl">
                <map-to xml="flowchart_root"/>
                <field name="flowchart_element" type="aztec.elements.FlowchartElement" direct="true">
                        <bind-xml auto-naming="deriveByClass" matches="SequenceImpl ParallelImpl
TaskImpl WhileImpl ChoiceImpl SwitchImpl HParallelImpl" node="element"/>
                </field>
                <field name="input_params" type="aztec.parameters.impl.InputParamsImpl" direct="true">
                        <bind-xml name="input_params" node="element"/>
                </field>
                <field name="output_params" type="aztec.parameters.impl.OutputParamsImpl" direct="true">
                        <bind-xml name="output_params" node="element"/>
                </field>
                <field name="int_vars" type="aztec.parameters.impl.IntVarsImpl" direct="true">
                        <bind-xml name="int_vars" node="element"/>
                </field>
                <field name="_name" type="java.lang.String" direct="true">
                        <bind-xml name="name" node="attribute"/>
                </field>
        </class>
```

**γ) Το αρχείο αντιστοίχησης για την μετατροπή του στοιχείου XML «flowchart_root» στο αντικείμενο «Flowchart Root»**

**Εικόνα 4.4**

### 4.1.5 Η συνιστώσα «Event Listener»

Ο «Event Listener» έχει σαν μοναδική λειτουργικότητα να «ακούει» για γεγονότα, και όταν του προωθηθεί ένα γεγονός (από οποιονδήποτε το δημιουργήσει μίας και η **Εικόνα 4.1** επιδεικνύει ότι γεγονότα προς τον «Event Listener» μπορεί προέρχονται από πολλές διαφορετικές κατευθύνσεις) να το προωθεί με την σειρά του στην συνιστώσα «Event Scheduler». και να ειδοποιεί (notify) τον «Event Scheduler» ότι του προώθησε ένα νέο γεγονός.

### 4.1.6 Η συνιστώσα «Event Scheduler»

Ο «Event Scheduler» διατηρεί μία ουρά με τα γεγονότα που του έχουν προωθηθεί (Event Queue). Αν του ζητηθεί, ο «Event Scheduler» μπορεί να επιστρέψει στον «αιτών» το «γεγονός» με την μεγαλύτερη «προτεραιότητα εκτέλεσης» από αυτά που βρίσκονται στην «Event Queue». Αν την δεδομένη στιγμή που του ζητείται, η «Event Queue» είναι

άδεια τότε ο «Event Scheduler» «παγώνει» την εκτέλεση του και περιμένει μέχρις ότου ο «Event Listener» να του προωθήσει ένα «γεγονός» και να τον ειδοποιήσει (notify).

Στην «απλή» περίπτωση το «γεγονός» με την μεγαλύτερη «προτεραιότητα» είναι το πρώτο στην ουρά ακολουθώντας ένα αλγόριθμο FIFO (First In First Out). Παρόλ'αυτα αναφέραμε νωρίτερα ότι η Μηχανή Εκτέλεσης του AZTEC υποστηρίζει ρητά την «ανάθεση» προτεραιοτήτων στις εκτελέσεις των διαγραμμάτων ροής. Όταν ένα «Γεγονός Εκκίνησης» δημιουργείται, μέσα στα δεδομένα του περιλαμβάνει και την «προτεραιότητα εκτέλεσης» που θα έχει από την στιγμή που θα τοποθετηθεί στην «Event Queue». Να σημειωθεί ότι η προτεραιότητα αυτή ισχύει τόσο για την εκτέλεση του συγκεκριμένου «Γεγονότος Εκκίνησης», όσο και για την εκτέλεση των «ενεργειών» του διαγράμματος ροής που θα ενεργοποιηθεί σαν αποτέλεσμα της εκτέλεσης του «Γεγονότος Εκκίνησης» (Launch Event). Οι προτεραιότητες των διαφόρων, διαφορετικών διαγραμμάτων ροής της ίδιας διαδικασίες μπορούν να είναι είτε απόλυτα ορισμένες (π.χ. το διάγραμμα ροής add_participant έχει προτεραιότητα 1 και το out_of_money 3 οπότε τόσο το γεγονός out_of_money όσο και οι «ενέργειες» του σχετικού διαγράμματος ροής θα εκτελεστούν πρώτα από αυτές του add_participant) είτε «σχετικές» μεταξύ των διαφόρων διαγραμμάτων ροής. Οι τεχνικές ανάθεσης προτεραιοτήτων στα ενεργά διαγράμματα ροής είναι ένα πεδίο της μελλοντικής μας εργασίας.

### 4.1.7 Η συνιστώσα «Task Dispatcher»

Η συνιστώσα «Task Dispatcher» διατηρεί ένα σύνολο από «ενεργές» ίνες εκτέλεσης (worker threads) στις οποίες αναθέτει γεγονότα (οποιουδήποτε τύπου) για να τα εκτελέσουν. Από την στιγμή που «αρχικοποιείται» η πλατφόρμα AZTEC, και η Μηχανή Εκτέλεσης από τον διαχειριστή του συστήματος η συνιστώσα «Task Dispatcher» αιτείται συνεχώς και επαναληπτικά από τον «Event Scheduler» να του «δώσει» το γεγονός με την μεγαλύτερη προτεραιότητα από την «Event Queue» για να το «δρομολογήσει» για εκτέλεση. Σε περίπτωση που δεν υπάρχει γεγονός στην «Event Queue» την συγκεκριμένη στιγμή οπότε και ο «Event Scheduler» «περιμένει» (wait), αντίστοιχα «περιμένει» και ο «Task Dispatcher». Μόλις επιστραφεί ένα γεγονός στον «Task Dispatcher» αυτός το αναθέτει σε μία ξεχωριστή «ίνα εκτέλεσης», (execution thread) που αναπαρίσταται από την συνιστώσα «Task Helper», για να το εκτελέσει.

Αντί της δημιουργίας και της καταστροφής ενός αντικείμενου «Task Helper» (ενός thread) για την εκτέλεση κάθε «ενέργειας» έχει επιλεγεί η δημιουργία ενός συνόλου από επαναχρησιμοποιήσιμες «ίνες εκτέλεσης» (Pool of Worker Threads) από το οποίο ο

«Task Dispatcher» επιλέγει μια «διαθέσιμη» ίνα εκτέλεσης και της αναθέτει την εκτέλεση της «ενέργειας». Υπάρχουν αρκετοί λόγοι που συνηγορούν στην χρήση «Pool of Threads» αντί της αυθαίρετης δημιουργίας/καταστροφής «ινών εκτέλεσης». Θέματα όπως το μέγεθος που πρέπει να έχει το «Pool of Threads» και πώς αυτό πρέπει να μεταβάλλεται θα αναφερθούν αναλυτικά στην επόμενη υποενότητα. όπου θα αναφέρουμε και τις σχετικές επιλογές που έχουμε κάνει στην δική μας περίπτωση.

#### 4.1.7.1 Pool of Threads

Η πλατφόρμα AZTEC έχει σαν βασική απαίτηση της την εκτέλεση «ταυτόχρονα» και «ανεξάρτητα» τόσο διαφορετικών διαδικασιών XASC όσο και διαφορετικών διαγραμμάτων ροής της ίδιας διαδικασίας, ακόμα και διαφορετικών ενεργειών του ίδιου διαγράμματος ροής. Κύριο χαρακτηριστικό της λήψης γεγονότων προς εκτέλεση είναι η ασύγχρονη φύση τους μίας και δέν μπορεί να προβλεφθεί από πριν πότε θα συμβεί ένα γεγονός. Εκτός της «ταυτόχρονης» εκτέλεσης γεγονότων μία βασική απαίτηση σε μία πλατφόρμα σύνθεσης και εκτέλεσης Ηλεκτρονικών Υπηρεσιών όπως το AZTEC είναι οι όσο το δυνατόν μικρότεροι χρόνοι απόκρισης που συνυφάζεται με την μη δημιουργία καθυστερήσεων (bottlenecks) στο εσωτερικό της Μηχανής Εκτέλεσης. Επιπρόσθετα απαιτείται η σωστή συμπεριφορά του συστήματος ακόμα και στην περίπτωση λήψης γεγονότων με πολύ μεγάλους ρυθμούς και της εκτέλεσης πάρα πολλών διαδικασιών ταυτόχρονα. Τα παραπάνω παρέχουν μία αντιστοίχηση στις απαιτήσεις που έχουμε από το AZTEC με τις απαιτήσεις που υπάρχουν (τηρουμένων των αναλογιών) από τους εξυπηρετητές του διαδικτύου.

Το multithreading είναι μία «κοινώς αποδεκτή» (standard) τεχνική για τον χειρισμό αιτήσεων που λαμβάνονται σε εξυπηρετητές του παγκόσμιου ιστού (web servers), εξυπηρετητές βάσεων δεδομένων (database servers) κ.λ.π Για τέτοιου είδους εφαρμογές το multithreading μπορεί να επιτύχει καλύτερους χρόνους απόκρισης (responsiveness), κλιμάκωση (scalability) και μεγαλύτερο ρυθμό εξυπηρέτησης (throughput) βελτιώνοντας έτσι την επικοινωνία μεταξύ διαδικασιών. Η χρήση multithreading ενδείκνυται σε εφαρμογές με έντονη λειτουργικότητα εισόδου/εξόδου (intensive I/O operations) όπου πολύς χρόνος σπαταλιέται χωρίς να γίνεται χρήση των πραγματικών δυνατοτήτων του επεξεργαστή. Χρησιμοποιώντας «multithreading» αν μία «ίνα» «παγώσει», ο επεξεργαστής απλά εκτελεί τις υπόλοιπες «ενεργές ίνες» (active threads). Σημαντική βελτίωση μπορεί να επιτευχθεί και σε εξυπηρετητές (servers) που χρησιμοποιούν πολλαπλούς επεξεργαστές (SMP) όπου πρακτικά κάθε «ίνα» μπορεί να

«αντιστοιχεί» σε διαφορετικό επεξεργαστή επιτυγχάνοντας κατά αυτό τον τρόπο ισορρόπηση του φόρτου (load balancing).

Οι αρχιτεκτονικές που χρησιμοποιούνται για την υλοποίηση multithreading κατατάσσονται σε δύο γενικές κατηγορίες, τις «ίνα ανά αίτηση» (thread per request) και τις «thread pool» αρχιτεκτονικές[S98]. Η αρχιτεκτονική «ίνας ανά αίτηση» δημιουργεί μία «ίνα» για κάθε αίτηση που λαμβάνεται και την καταστρέφει όταν τελειώσει ο χειρισμός της αίτησης. Η «thread pool» αρχιτεκτονική δημιουργεί και διατηρεί ένα σύνολο (pool) από «ίνες». Όταν ληφθεί μία αίτηση, ο εξυπηρετητής την αναθέτει σε μία «ελεύθερη ίνα» (που δεν εκτελεί κάποια άλλη αίτηση) και όταν τελειώσει η εξυπηρέτηση της αίτησης «επιστρέφει» την «ίνα» στο «pool»

Παρόλ'αυτά η χρήση «multithreading» δεν είναι πανάκεια. Χρησιμοποιώντας μία αρχιτεκτονική «ίνας ανά αίτηση» το «multithreading» επιφέρει ένα κόστος εκτέλεσης (run-time overhead) που οφείλεται στην συνεχή δημιουργία και καταστροφή «ινών» . Για παράδειγμα μπορούμε να αναφέρουμε ότι η δημιουργία μίας «ίνας» στο λειτουργικό σύστημα Windows NT ή στο Solaris απαιτεί ένα «megabyte» εικονικής μνήμης (virtual memory) για την στοίβα της «ίνας» (thread stack)[LB96]. Μεγάλοι ρυθμοί λήψης αιτήσεων ισοδυναμούν με πολύ συχνές δεσμεύσεις και αποδεσμεύσεις μνήμης προκαλώντας έτσι μειωμένη απόδοση (performance bottlenecks) Από την άλλη με τη χρήση αρχιτεκτονικών «thread pool», η διατήρηση «ινών» στο «pool» επιφέρει το κόστος του «context switching». Κάθε «ίνα» «τρέχει» σε μία «ελαφριά» (lightweight) διαδικασία (LPW) η οποία μπορεί να θεωρηθεί σαν μία εικονική CPU. Η διαδικασία του «context switching» αναφέρεται στην αφαίρεση μίας «ίνας» από την LPW και την αντικατάσταση της από μία άλλη (Ο επεξεργαστής για να αποφύγει φαινόμενα «λιμοκτονίας» (starvation) πρέπει να μοιράζει τον χρόνο του δίκαια ανάμεσα σε όλες τις «ίνες»). Σε ένα SPARC station 10/41 η διαδικασία του «context switch» ανάμεσα σε δύο «ίνες» διαρκεί περίπου 20 microsecond.

Από την στιγμή που οι αρχιτεκτονικές «thread pool» δημιουργούν «εξ'αρχής» ένα σύνολο από «ίνες» (pool of threads) μια αρχιτεκτονική «ίνας ανά αίτηση» μπορεί να θεωρηθεί σαν μία υπό-περίπτωση της αρχιτεκτονικής «thread pool» όπου το πλήθος των αρχικών «ινών» που δημιουργούνται είναι μηδέν. Επομένως το γενικότερο πρόβλημα έγκειται στον προσδιορισμό του «βέλτιστου» μεγέθους για το «thread pool» . Παρόλο που το κόστος του «context switching» είναι μικρότερο από την δημιουργία νέων «ινών» το επιπλέον κόστος της διατήρησης ενός μεγάλου «thread pool» μπορεί να υποσκελίσει

αυτό το όφελος καταναλώνοντας πόρους του συστήματος με την μορφή αυξημένων απαιτήσεων μνήμης και προγραμματισμού (memory and scheduling overhead).

Η συνήθης τακτική για τον προσδιορισμό του «βέλτιστου» μεγέθους του «thread pool» είναι ένας συνδυασμός ευριστικών κανόνων και πρακτικών εμπειριών. Ένας ευριστικός κανόνας χρησιμοποιείται για τον προσδιορισμό του «βέλτιστου» αρχικού μεγέθους του «thread pool» και από εκεί και μετά οι διαχειριστές αναλαμβάνουν την παρακολούθηση και την αλλαγή του μεγέθους του «pool» σε περίπτωση που παρατηρηθούν καθυστερήσεις (bottlenecks). Ένας τέτοιος ευριστικός κανόνας είναι ότι το μέγεθος του «thread pool» πρέπει να είναι διπλάσιο από τον αριθμό των CPUs του μηχανήματος που χρησιμοποιείται. Ένας άλλος τέτοιος κανόνας (που χρησιμοποιείται από τον εξυπηρετητή Microsoft IIS) είναι ότι το αρχικό μέγεθος του «pool» πρέπει να 10 «ίνες» και αυξάνει ανάλογα με τον ρυθμό άφιξης αιτήσεων. Επίσης ευριστικά το μέγιστο μέγεθος ενός «thread pool» δεν πρέπει να ξεπερνάει το «διπλάσιο» του αριθμού των «megabytes» μνήμης RAM του μηχανήματος. Όλοι αυτοί οι κανόνες είναι ευριστικοί και δεν παρέχουν καμία θεωρητική εξασφάλιση ότι το μέγεθος του «thread pool» που θα δημιουργηθεί δεν θα είναι πολύ μεγάλο ή πολύ μικρό. Το [LML00] παρέχει ένα θεωρητικό υπόβαθρο του προσδιορισμού του «βέλτιστου» μεγέθους του «thread pool» παρέχοντας μαθηματικές σχέσεις και αλγορίθμους οι οποίοι συσχετίζουν το μέγεθος του «thread pool», τον φόρτο του συστήματος και τα διάφορα κόστη.

Στην Μηχανή Εκτέλεσης του AZTEC η συνιστώσα «Task Dispatcher» είναι αυτή που διατηρεί ένα «pool of threads».Κάθε «ίνα» είναι ένα αντικείμενο «Task Helper» το οποίο αναλαμβάνει την εκτέλεση ενός γεγονότος. Στην πρωτότυπη υλοποίηση δεν ξεφύγαμε από την «παγίδα» να χρησιμοποιήσουμε κάποιους ευριστικούς κανόνες τόσο για το αρχικό και το μέγιστο μέγεθος του «pool» όσο και για τον ρυθμό με τον οποίο αυτό αυξομειώνεται ανάλογα με τον αριθμό των ελεύθερων «ινών». Ακολουθώντας προσέγγιση παρόμοια με αυτή του εξυπηρετητή Apache [APACHE2] αρχικοποιούμε το «pool» με αρχικό μεγεθος=50 «ίνες», αυξάνουμε το «pool» κάθε φορά που το πλήθος των ελεύθερων «ινών» γίνει μικρότερο του 10 κατά αρχικό μέγεθος/ 2, μειώνουμε το μέγεθος του «pool» (καταστρέφοντας «ίνες» )όταν το πλήθος των ελεύθερων ινών ξεπεράσει αρχικό μέγεθος + αρχικό μέγεθος/ 2 ενώ θέτουμε ως «άνω όριο» στο μέγεθος τού «thread pool» το διπλάσιο της μνήμης RAM σε «megabyte» του υπολογιστή που «τρέχει» το AZTEC. Κάθε φορά που αναθέτει ένα γεγονός σε ένα «Task Helper» ο «Task Dispatcher», τον σημειώνει ως «κατειλημμένο» ενώ όταν τελειώσει την εκτέλεση του ο «Task Helper» θέτει τον «εαυτό» του «διαθέσιμο» και ενημερώνει τον «Task Dispatcher» για να «ενημερώσει» τον μετρητή των «διαθέσιμων ινών».

Αρκετές έρευνες έχουν δείξει ότι οι αρχιτεκτονικές «thread pool» μπορούν να βελτιώσουν σημαντικά την απόδοση και να μειώσουν τον «χρόνο απόκρισης» (response time) του συστήματος. [S98][C97]

### 4.1.8 Η συνιστώσα «Task Helper»

Οι συνιστώσες «Task Helpers», είναι οι ίνες ελέγχου που αναλαμβάνουν την εκτέλεση γεγονότων όλων των ειδών, είτε αυτά είναι «Γεγονότα Εκκίνησης» είτε «Γεγονότα Εκτέλεσης». Στην περίπτωση μάλιστα που μιλάμε για την εκτέλεση «Γεγονότων Εκτέλεσης» που αντιστοιχούν σε εξωτερικές ενέργειες (external tasks) θα δούμε ότι οι «Task Helpers» υλοποιούν τα προγράμματα «wrappers» που αναλαμβάνουν την δημιουργία κλήσεων SOAP RPC για την επικοινωνία του AZTEC με εξωτερικές Ηλεκτρονικές Υπηρεσίες.

#### 4.1.8.1 Γεγονότα Εκκίνησης

##### *4.1.8.1.1 Αποτίμηση των στοιχείων του διαγράμματος ροής*

Στην περίπτωση των «γεγονότων εκκίνησης», ο «Task Helper» αναλαμβάνει την αντιστοίχηση του γεγονότος με κάποιο ενεργό διάγραμμα ροής. Αφού γίνει αυτό και ανακτηθεί το αντικείμενο που αναπαριστά το συγκεκριμένο διάγραμμα ροής γίνεται η «ενεργοποίηση» του, πρακτικά καλώντας την μέθοδο **evaluate()** του αντικειμένου «Flowchart Root» που αναπαριστά την «ρίζα» του διαγράμματος ροής Λόγω της δομημένης φύσης των διαγραμμάτων ροής της XASC η «εκτέλεση» των διαφόρων ενεργειών (Tasks) (που βρίσκονται στα «φύλλα» του διαγράμματος ροής) αντιστοιχεί σε μία «depth-first» διάσχιση του διαγράμματος ροής, δηλαδή της ιεραρχίας από αντικείμενα Java που σαν «ριζικό» αντικείμενο έχει το «Flowchart Root».

Η διάσχιση αυτή επιτυγχάνεται με μία αναδρομική «αποτίμηση» (recursive evaluation) της σημασιολογίας του κάθε στοιχείου XASC, δηλαδή του κάθε αντικειμένου στην ιεραρχία των αντικειμένων που ξεκινάει από το «Flowchart Root». Κάθε αντικείμενο έχει μία μέθοδο **evaluate()** η οποία εκτελεί κάποιες ενέργειες και έπειτα καλεί «αναδρομικά» τις μεθόδους **evaluate()** των «υπό-αντικειμένων» του ανάλογα πάντα με την σημασιολογία του στοιχείου XASC που «αναπαριστά». Η εκτέλεση του διαγράμματος ροής θα έχει ολοκληρωθεί, όταν «επιστρέψει» η μέθοδος **evaluate()** του αντικειμένου «Flowchart Root». Υπό αυτή την έννοια, ο συγκεκριμένος «Task Helper» (δηλαδή η

συγκεκριμένη «ίνα εκτέλεσης») έχει αναλάβει την εκτέλεση ολόκληρού του διαγράμματος ροής μιας και θα είναι «κατειλημμένος» έως ότου τελειώσει η αναδρομική εκτέλεση του διαγράμματος ροής.

Για να μιλήσουμε με παραδείγματα η **evaluate()** του αντικειμένου «Flowchart Root» της **Εικόνας 4.3** καλεί τις **evaluate()** των αντικειμένων «Input Params», «Output Params» και «Int Vars» τα οποία αναπαριστούν τις παραμέτρους εισόδου, εξόδου και τις τοπικές μεταβλητές του διαγράμματος ροής και έπειτα καλεί την **evaluate()** του αντικειμένου που αναπαριστά το στοιχείο XASC που «κρέμεται» από την «ρίζα» του διαγράμματος ροής, το οποίο σε αυτή την περίπτωση είναι το «Sequence» (από την ρίζα του διαγράμματος ροής μπορεί να «κρέμεται» μόνο ένα στοιχείο XASC όπως είδαμε και στην **Εικόνα 4.3 β)**). Οι **evaluate()** των αντικειμένων που αναπαριστούν τις παραμέτρους του διαγράμματος ροής προκαλούν την «αποθήκευση» των παραμέτρων αυτών στη δομή «Flowchart Variables» τού αντικειμένου «Flowchart Root» που διατηρεί όλες τις «καθολικά ορατές» (global) μεταβλητές στα πλαίσια του διαγράμματος ροής. Η **evaluate()** του αντικειμένου που αναπαριστά το στοιχείο «Sequence» καλεί «*σειριακά*» τις **evaluate()** όλων των υπό-αντικειμένων του δηλαδή του «False» και του «True». Με τον όρο σειριακά εννοούμε ότι η **evaluate()** του αντικειμένου «Task» θα κληθεί αφού έχει επιστρέψει η **evaluate()** του «Choice». Η κλήση της **evaluate()** του «Choice» προκαλεί την κλήση των **evaluate()** των αντικειμένων «True» και «False» και πάει λέγοντας. Αντίστοιχα η **evaluate()** του αντικειμένου που αναπαριστά το στοιχείο «Parallel» καλεί «*παράλληλα*» (concurrently) τις **evaluate()** όλων των υπό-αντικειμένων του και αντίστοιχα συμβαίνει και για τα υπόλοιπα στοιχεία XASC (While,Choice,HParallel).

Η αναδρομική αποτίμηση (recursive evaluation) των στοιχείων του διαγράμματος ροής θα οδηγήσει τελικά στην αποτίμηση των «ενεργειών» (Tasks) του διαγράμματος ροής. Η αποτίμηση του αντικειμένου που «αναπαριστά» μία «ενέργεια» στην XASC εξαρτάται από τον τύπο της «ενέργειας». Σε περίπτωση που η «ενέργεια» είναι «ενέργεια γεγονότος», τότε θα δημιουργηθεί ένα «Γεγονός Εκκίνησης» του οποίου η εκτέλεση θα προκαλέσει την ενεργοποίηση ενός νέου διαγράμματος ροής της διαδικασίας. Το γεγονός αυτό θα μεταφέρει τα δεδομένα εισόδου του διαγράμματος ροής που θα ενεργοποιηθεί (αντικείμενο «Input Params») τα οποία θα προέρχονται από την αποτίμηση των δεδομένων της «ενέργειας γεγονότος» (αντικείμενο «Task Input Params»).

### *4.1.8.1.2 Αποτίμηση παραμέτρων και εμβέλεια*

Όταν ένα στοιχείο της XASC έχει παραμέτρους εισόδου όπως το «Choice», το «While»,το «HParallel» και το «Task», η **evaluate()** του χειρίζεται την «αποτίμηση» και αυτών των παραμέτρων (καλώντας την **evaluate()** του αντίστοιχου αντικειμένου «Task Input Params») εκτός από την «αναδρομική» αποτίμηση των υποστοιχείων XASC (κάτι το οποίο για λόγους συντομίας δεν αναπαραστάθηκε στην **Εικόνα 4.3 β)**). Η αποτίμηση (evaluation) των παραμέτρων των διάφορων αντικειμένων (στοιχείων XASC) γίνεται αναζητώντας τις τιμές τους στην «καθολική» (global) δομή «Flowchart Variables» του αντικειμένου «Flowchart Root» που διατηρεί τις μεταβλητές δεδομένων του διαγράμματος ροής, σε περίπτωση οι παράμετροι του στοιχείου δεν ορίζονται ρητά, στα πλαίσια του στοιχείου. Για να δώσουμε ένα απλοϊκό παράδειγμα, στο κομμάτι κώδικα XASC της **Εικόνας 4.5** βλέπουμε ότι υπάρχουν δύο τρόποι για να αποτιμηθούν οι παράμετροι του στοιχείου Choice. Στο **α)** η παράμετρος «account» ορίζεται ρητά στα «πλαίσια» του Choice και για αυτό αποτιμάται χωρίς αναζήτηση της τιμής της στην δομή «Flowchart Variables» ενώ στο **β)** η τιμή της παραμέτρου θα αναζητηθεί στην δομή «Flowchart Variables» του αντικειμένου «Flowchart Root». Να σημειωθεί ότι με αυτό τον τρόπο υποστηρίζουμε τον ορισμό μεταβλητών με «τοπική εμβέλεια» που δεν υποστηρίζεται για παράδειγμα από γλώσσες όπως η BPEL4WS η οποία επιτρέπει μόνο τον ορισμό «containers» με καθολική εμβέλεια.

```
<flowchart_root name="root">
   …
   <choice condition="account==0">
      <task_input_params>
        <in_param name = "account">
         <ValueInt>some_number</ValueInt>
        </in_param>
      <task_input_params>
      …
   </choice>
</flowchart_root>
```

**α)**

```
<flowchart_root name="root">
   <input_params>
      <param>
        <ValueInt>account</ValueInt>
      </param>
   </input_params>
   …
   <choice condition="account==0">
      <task_input_params>
        <in_param>
            <ValueVariable>account</ValueVariable>
        </in_param>
      <task_input_params>
      …
   </choice>
</flowchart_root>
```

**β)**

**Εικόνα 4.5 Αποτίμηση παραμέτρων στοιχείου XASC α) ρητά β) με αναζήτηση στην δομή Flowchart Variables**

### 4.1.8.2 Γεγονότα Εκτέλεσης

Στην προηγούμενη υποενότητα μελετήσαμε την περίπτωση της «ενέργειας γεγονότος». Σε περίπτωση που η «ενέργεια» είναι οποιουδήποτε άλλου είδους (εξωτερική, εσωτερική, εσωτερική μέθοδος, repository, αναμονής ή κενή), θα δημιουργηθεί ένα «Γεγονός Εκτέλεσης» το οποίο πρακτικά θα «περιέχει» την «ενέργεια» και τις παραμέτρους εισόδου της οι οποίες θα χρησιμοποιηθούν σαν παράμετροι σε οποιαδήποτε κλήση μεθόδου (τοπική ή απομακρυσμένη) προκληθεί από την εκτέλεση του «Γεγονότος Εκτέλεσης». Το «γεγονός» αυτό (όπως και το «Γεγονός Εκκίνησης» στην προηγούμενη περίπτωση) προωθείται από τον «Task Helper» που «εκτελεί» το διάγραμμα ροής, στον «Event Listener»

Ο «Task Helper» αναλαμβάνει εκτός από την δημιουργία «Γεγονότων Εκτέλεσης» (στα πλαίσια της αποτίμησης ενός διαγράμματος ροής) και την εκτέλεση τους κάνοντας τις απαραίτητες ενέργειες ανάλογα με τον τύπο της «ενέργειας» που μεταφέρει το γεγονός. Από την στιγμή που θα ολοκληρωθεί η εκτέλεση της ενέργειας (ότι τύπου και αν είναι) και ανακτηθούν τυχόν αποτελέσματα, ο «Task Helper» ειδοποιεί τον «Task

Dispatcher» ότι είναι και πάλι διαθέσιμος (available) για να του ανατεθούν γεγονότα προς εκτέλεση.

#### *4.1.8.2.1 Εξωτερικές Ενέργειες*

Στην περίπτωση που η ενέργεια είναι τύπου «εξωτερική» (external task) σημαίνει ότι η εκτέλεση της θα ισοδυναμεί με την κλήση κάποιας εξωτερικής Ηλεκτρονικής Υπηρεσίας από την οποία μπορεί είτε να περιμένουμε κάποια αποτελέσματα (τύπος επικοινωνίας solicit-response) είτε όχι (notification). Σε αυτή την περίπτωση ο «Task Helper» «υλοποιεί» το πρόγραμμα «wrapper» με το οποίο το AZTEC επικοινωνεί με εξωτερικές Ηλεκτρονικές Υπηρεσίες χρησιμοποιώντας κλήσεις SOAP RPC. Σε περίπτωση που επιστραφούν κάποια αποτελέσματα από την κλήση της Ηλεκτρονικής Υπηρεσίας αυτά θα μετατραπούν στην κατάλληλη μορφή και θα αποθηκευτούν στην δομή «Flowchart Variables» του αντικειμένου «Flowchart Root» ούτως ώστε να είναι «προσβάσιμα» από τα υπόλοιπα «στοιχεία» και «ενέργειες» του διαγράμματος ροής

#### *4.1.8.2.2 Εσωτερικές Ενέργειες*

Στην περίπτωση που η ενέργεια είναι τύπου «εσωτερική» (internal task) πρακτικά θα ισοδυναμεί με την αποτίμηση ενός αυθαίρετου (arbitrary) κομματιού κώδικα Java (π.χ. i=i+1 στα πλαίσια ενός στοιχείου While). Η αποτίμηση αυτή γίνεται με την χρήση τού μεταφραστή «bsh Interpreter» (Bean Shell Interpreter). Πρακτικά ο μεταφραστής «φορτώνεται» με τις παραμέτρους εισόδου («Task Input Params») της «εσωτερικής ενέργειας» και η «έκφραση» αποτιμάται χρησιμοποιώντας τις τιμές αυτών των παραμέτρων. Οι παράμετροι εξόδου («Task Return Params») της «εσωτερικής ενέργειας» αποθηκεύονται και πάλι στην δομή «Flowchart Variables» για να γίνουν καθολικά ορατές.

#### *4.1.8.2.3 Εσωτερικές μέθοδοι του AZTEC*

Στην περίπτωση που η ενέργεια είναι τύπου «εσωτερική μέθοδος» (internal function) η εκτέλεση της ισοδυναμεί με την εκτέλεση μίας μεθόδου ορισμένης εσωτερικά στην Μηχανή Εκτέλεσης του AZTEC. Τέτοιου είδους μέθοδοι μπορεί να είναι, μέθοδοι για συγκεκριμένες περιπτώσεις (case-specific) (όπως π.χ. η SendEMail), ή μέθοδοι οι οποίες διαχειρίζονται και πιθανώς αλλάζουν την εκτέλεση ενεργών διαγραμμάτων ροής και διαδικασιών όπως η «suspendAlFcs», «resumeAllFcs» και «terminateProcess». Η κλήση των μεθόδων αυτών γίνεται με την τεχνική «reflection» της Java.

Κλήση της μεθόδου «suspendAllFcs» προκαλεί την παύση της εκτέλεσης όλων των διαγραμμάτων ροής και των «ενεργειών» διαγραμμάτων ροής ενός συγκεκριμένου τύπου. Πρακτικά αυτό που κάνει αυτή η μέθοδος είναι ότι σχολιάζει (annotates) το αντικείμενου «Flowchart Root» κάθε διαγράμματος ροής του συγκεκριμένου τύπου σαν «suspended». Ο «Event Scheduler» πριν προωθήσει ένα γεγονός στον «Task Dispatcher» για εκτέλεση, κοιτάει το αντικείμενο «Flowchart Root» που σχετίζεται με αυτό το γεγονός και σε περίπτωση που είναι σχολιασμένο ως «suspended» το «προσπερνάει». Όταν κληθεί η «resumeAllFcs» αντίστοιχα σχολιάζει τα αντικείμενα «Flowchart Root» των διαγραμμάτων ροής ενός συγκεκριμένου τύπου ως «running». Με αυτό τον τρόπο συνεχίζεται η εκτέλεση των γεγονότων που νωρίτερα είχαν «προσπεραστεί».

Η κλήση της μεθόδου «terminateProcess» σχολιάζει το αντικείμενο «Flowchart Root» όλων των διαγραμμάτων ροής μίας διαδικασίας ως «terminated». Ο «Event Scheduler» όταν συναντάει στην «Event Queue» ένα γεγονός του οποίου το «Flowchart Root» είναι σχολιασμένο ως «terminated», το αφαιρεί. Επίσης ο «Task Helper» επικοινωνεί με το «Flowchart Runtime» και αφαιρεί τόσο τα διαγράμματα ροής, όσο και το συγκεκριμένο στιγμιότυπο της διαδικασίας από τις λίστες με τα ενεργά διαγράμματα ροής και τις διαδικασίες αντίστοιχα.

#### *4.1.8.2.4 Ενέργειες του «Context Repository»*

Στην περίπτωση που η ενέργεια είναι τύπου «repository» η εκτέλεση της ισοδυναμεί με μία από τις ενέργειες insert, set, get, bget ή delete στο «εικονικό» (virtual) έγγραφο XML, «Context Repository». Όπως έχουμε αναφέρει και νωρίτερα το Context Repository είναι ένα έγγραφο XML στην κύρια μνήμη (main-memory XML file) το οποίο δημιουργείται όταν δημιουργείται ένα στιγμιότυπο μίας διαδικασίας και καταστρέφεται όταν καταστρέφεται αυτό το στιγμιότυπο. Όπως έχουμε ξαναπεί το Context Repository χρησιμοποιείται για τον «διαμοιρασμό» δεδομένων μεταξύ των ενεργών διαγραμμάτων ροής μίας διαδικασίας και συχνά για τον συγχρονισμό της εκτέλεσης τους. Παρακάτω παραθέτουμε τις ενέργειες του «Context Repository»

- Η ενέργεια «insert» δημιουργεί μία νέα καταχώρηση (ένα νέο στοιχείο) στο «Context Repository» με μία συγκεκριμένη τιμή. Η «θέση» του νέου στοιχείου, μέσα στο «Context Repository» προσδιορίζεται με μία έκφραση XPath [XPATH].
- Η ενέργεια «set» αλλάζει την τιμή ενός στοιχείου στο «Context Repository» σε ένα συγκεκριμένο XPath.

- Η ενέργεια «get» ανακτά την τιμή ενός στοιχείου στο «Context Repository» σε ένα συγκεκριμένο XPath. Σε περίπτωση που στο συγκεκριμένο XPath δεν βρεθεί κάποιο στοιχείο, η ενέργεια «get» επιστρέφει null.
- Η ενέργεια «bget» έχει παρόμοια λειτουργικότητα με την «get» με μόνη διαφορά ότι σε περίπτωση που δεν βρεθεί στοιχείο στο συγκεκριμένο XPath, η «bget» «περιμένει» (blocks) έως ότου δημιουργηθεί ένα στοιχείο. Αυτό το χαρακτηριστικό λειτουργεί ως μέσο συγχρονισμού μιας και το διάγραμμα ροής που «περιέχει» την ενέργεια «bget» «παύει» την εκτέλεση του έως ότου ένα άλλο διάγραμμα ροής δημιουργήσει το ζητούμενο στοιχείο.
- Η ενέργεια «delete» ανακτά την τιμή ενός στοιχείου σε ένα XPath στο «Context Repository» και στη συνέχεια διαγράφει το στοιχείο αυτό από το Context Repository.

#### *4.1.8.2.5 Ενέργειες αναμονής και η «κενή» ενέργεια*

Στην περίπτωση που η ενέργεια είναι τύπου «αναμονής» (wait) η ίνα εκτέλεσης απλά μπλοκάρει για το προκαθορισμένο χρονικό διάστημα ενώ η εκτέλεση της «κενής» (empty) ενέργειας δεν έχει κανένα αντίκτυπο

### 4.1.9 Αλληλεπιδράσεις των συνιστωσών κατά τον χειρισμό ενός γεγονότος

Έχοντας περιγράψει αναλυτικά την λειτουργικότητα των διαφόρων συνιστωσών της πλατφόρμας AZTEC μπορούμε να περιγράψουμε την πορεία που ακολουθεί ένα γεγονός απο την στιγμή της δημιουργίας του μέχρι την εκτέλεση του. Η πορεία αυτή αναπαρίσταται με αριθμημένα βήματα στο σχήμα της **Εικόνας 4.1**. Όταν το γεγονός που δημιουργείται αντιστοιχεί στην δημιουργία ενός νέου στιγμιοτύπου μίας διαδικασίας (βήμα 1), το πρώτο πράγμα που γίνεται είναι η επικοινωνία της Main, με την συνιστώσα Flowchart Runtime (βήμα 2) η οποία θα ελέγξει αν υπάρχει κάποιο ήδη ενεργό στιγμιότυπο διαδικασίας με το ίδιο «external process id» οπότε και δεν θα δημιουργηθεί το νέο στιγμιότυπο διαδικασίας. Στο επόμενο βήμα (βήμα 3) η «Main» θα επικοινωνήσει με την συνιστώσα «Flowchart Repository» της «Διαχείρισης Σχημάτων» (Schema Management) η οποία είτε θα φορτώσει το «σχήμα» της διαδικασίας, είτε θα συμπεράνει ότι το «σχήμα» αυτό είναι ήδη φορτωμένο κατά την δημιουργία ενός προηγούμενου στιγμιότυπου της διαδικασίας. Ενδιάμεσα στο βήμα 4, το «Flowchart Repository» θα «περάσει» στον «Parser» τα κείμενα XML που αντιστοιχούν στο «σχήμα» της διαδικασίας και θα ανακτήσει από αυτόν την αντικειμενοστρεφή αναπαράσταση τους. Εν συνεχεία (βήμα 5) η «Main» θα επικοινωνήσει με το «Flowchart Runtime» το

οποίο θα δημιουργήσει το νέο στιγμιότυπο διαδικασίας και στα πλαίσια αυτής ένα στιγμιότυπο του «εικονικού» κειμένου XML «Context Repository» (βήμα 6).

Αφού το νέο στιγμιότυπο διαδικασίας έχει δημιουργηθεί, το «Γεγονός Εκκίνησης» που θα ενεργοποιήσει το «αρχικό» (root) διάγραμμα ροής της διαδικασίας προωθείται στην συνιστώσα «Event Listener» (βήμα 7). Στο βήμα 8, ο «Event Listener» προωθεί το γεγονός στον «Event Scheduler» και τον ειδοποιεί σε περίπτωση που είχε «σταματήσει» (sleep) την εκτέλεση του. Ο «Event Scheduler» με την σειρά του προωθεί το γεγονός στον «Task Dispatcher» (βήμα 9) ο οποίος το αναθέτει στον πρώτο «ελεύθερο» «Task Helper» που βρίσκει στο «Pool of Threads» ο οποίος αναλαμβάνει την εκτέλεση του η οποία ουσιαστικά προκαλεί την ενεργοποίηση του κατάλληλου διαγράμματος ροής. Για να ανακτήσει αυτό το διάγραμμα ροής ο «Task Helper» «επικοινωνεί» ,με το «Flowchart Repository» στο βήμα 10 και του ζητάει το διάγραμμα ροής με το συγκεκριμένο όνομα στα πλαίσια της συγκεκριμένης διαδικασίας. Το «Flowchart Repository» δημιουργεί ένα «κλώνο» του ζητούμενου διαγράμματος ροής στο βήμα 11 και το επιστρέφει στον «Task Helper» στο βήμα 12. Στο σημείο αυτό ο «Task Helper» ενεργοποιεί το διάγραμμα ροής καλώντας την **evaluate()** του. Στο βήμα 13 το αντικείμενο που αναπαριστά το διάγραμμα ροής προστίθεται στην λίστα με τα ενεργά διαγράμματα ροής του «Flowchart Runtime». Τα διάφορα βήματα 14 αντιστοιχούν στην κατάσταση όπου η αποτίμηση του διαγράμματος ροής έχει φτάσει στις «ενέργειες» η οποίες προωθούνται με την μορφή «γεγονότων» (είτε «εκκίνησης» είτε «εκτέλεσης») στον «Event Listener» ενώ το βήμα 15 αναπαριστά την κλήση εξωτερικών Ηλεκτρονικών Υπηρεσιών στα πλαίσια της εκτέλεσης «εξωτερικών ενεργειών».

Μέχρι τώρα έχουμε περιγράψει την δημιουργία γεγονότων με τους τρεις από τους τέσσερις τρόπους που επιδεικνύει η **Εικόνα 4.1.** Έχουμε περιγράψει την δημιουργία ενός «Γεγονότος Εκκίνησης» που εκκινεί ένα νέο στιγμιότυπο μίας διαδικασίας (βήμα 1), έχουμε περιγράψει την δημιουργία ενός «Γεγονότος Εκκίνησης» από ένα διάγραμμα ροής με μία «ενέργεια γεγονότος» (βήμα 14) και έχουμε περιγράψει την δημιουργία από ένα διάγραμμα ροής «Γεγονότων Εκτέλεσης» (βήμα 14) που χειρίζονται την εκτέλεση των ενεργειών του διαγράμματος ροής. Ο τελευταίος τρόπος δημιουργίας «Γεγονότων Εκκίνησης» είναι από εξωτερικές Ηλεκτρονικές Υπηρεσίες ή εφαρμογές μέσω των προγραμμάτων «wrappers», που δίνουν την ορατή διεπαφή του AZTEC «προς τα έξω», με την χρήση κλήσεων SOAP RPC (βήμα 16). Όταν ένα τέτοιο «γεγονός» συμβαίνει καταρχάς πρέπει να έχει προσαρτημένο ένα «external_process_id» που θα το συσχετίσει με κάποιο ενεργό στιγμιότυπο μίας διαδικασίας. Η «κύρια» συνιστώσα (Main) της πλατφόρμας AZTEC στην οποία προωθείται το «γεγονός» από τους «wrappers»

επικοινωνεί με το «Flowchart» Runtime που ελέγχει αν όντως υπάρχει ενεργό στιγμιότυπο της συγκεκριμένης διαδικασίας με το συγκεκριμένο «external_process_id». Σε περίπτωση που δεν υπάρχει επιστρέφεται μήνυμα λάθους στην εφαρμογή που δημιούργησε το γεγονός (που εκτέλεσε την κλήση SOAP RPC) και το γεγονός δεν εκτελείται. Σε αντίθετη περίπτωση το «Γεγονός Εκκίνησης» προωθείται στον «Event Listener» ακολουθώντας την πορεία εκτέλεσης που περιγράψαμε νωρίτερα.

## *4.2 Αποτίμηση απόδοσης*

Στα πλαίσια της υλοποίησης της πλατφόρμας AZTEC πραγματοποιήσαμε μετρήσεις απόδοσης τόσο για να εκτιμήσουμε την αποδοτικότητα της «μηχανής εκτέλεσης» όσο και για να αναγνωρίσουμε πιθανά σημεία καθυστέρησης (bottlenecks) και να επιχειρήσουμε να τα αντιμετωπίσουμε. Οι μετρήσεις αυτές είχαν σαν απαίτηση τον έλεγχο του συστήματος τόσο για μικρούς όσο και για πολύ μεγάλους ρυθμούς δημιουργίας νέων διαδικασιών ούτως ώστε να μελετήσουμε την συμπεριφορά του συστήματος σε όσο το δυνατόν πιο ρεαλιστικές συνθήκες χρήσης.

### 4.2.1 Σενάριο

Για την διεξαγωγή των μετρήσεων χρησιμοποιήσαμε σαν σενάριο (benchmark workflow) το παράδειγμα Ηλεκτρονικού Εμπορίου που έχει περιγραφεί σε προηγούμενες ενότητες. Το παράδειγμα αυτό καλύπτει σε ικανοποιητικό βαθμό τις βασικές απαιτήσεις από μία γλώσσα περιγραφής Ροών Εργασίας ή σύνθεσης Ηλεκτρονικών Υπηρεσιών. Εστιάσαμε στην «πλήρως αυτοματοποιημένη» περίπτωση κατά την οποία η πληρωμή γίνεται μόνο μέσω πιστωτικής κάρτας, όποτε δεν υπάρχει αλληλεπίδραση με τον πελάτη (εκτός κάποιου ενημερωτικού e-mail). Η επιλογή αυτή βασίζεται στο γεγονός ότι θέλουμε να «αξιολογήσουμε» καθαρά την απόδοση «μηχανής εκτέλεσης» χωρίς «εξωτερικές» καθυστερήσεις που μπορεί να προέρχονται από την αλληλεπίδραση με χρήστες. Μοναδικός «αστάθμητος» παράγοντας θα μπορούσε να είναι η περίπτωση μη έγκυρης πιστωτικής κάρτας, γεγονός που θα μείωνε τον χρόνο ολοκλήρωσης της εκτέλεσης μίας διαδικασίας κατά πολύ γι' αυτό την μελετάμε σαν ξεχωριστή περίπτωση.

### 4.2.3 Μετρικές

Οι ανεύρεση μετρικών (metrics) για την αξιολόγηση μίας Πλατφόρμας Εκτέλεσης Ηλεκτρονικών Υπηρεσιών ή ενός Συστήματος Διαχείρισης Ροών Εργασίας είναι μία πρόκληση. Η αξιολόγηση τέτοιων συστημάτων δεν έχει «ωριμάσει» τόσο όσο για παράδειγμα η αξιολόγηση Συστημάτων Διαχείρισης Βάσεων Δεδομένων όπου τα παραδείγματα και οι μετρικές που χρησιμοποιούνται είναι δεδομένες. Μερικές ιδέες αφορούσαν τον μέσο χρόνο εκτέλεσης μίας διαδικασίας (turnaround time), το μέσο ρυθμό εξυπηρέτησης διαδικασιών (throughput) και τον μέσο χρόνο εκτέλεσης μίας «απλής» ενέργειας (step response time), όλα σε συνάρτηση με τον ρυθμό «άφιξης» νέων στιγμιοτύπων διαδικασίας (νέων παραγγελιών στο παράδειγμα Ηλεκτρονικού Εμπορίου). Αυτές οι μετρικές μπορούν να προσαρμοστούν στα ιδιαίτερα χαρακτηριστικά της δικής μας μηχανής.

Ο μέσος χρόνος εκτέλεσης σε σχέση με το ρυθμό άφιξης νέων στιγμιοτύπων διαδικασίας είναι μια αρκετά γενική μετρική και χρησιμοποιείται ως έχει. Ο μέσος ρυθμός εξυπηρέτησης νέων διαδικασιών πρακτικά αναπαριστά το ποσοστό των αιτήσεων (παραγγελιών) που ολοκληρώθηκαν σε συνάρτηση με τον ρυθμό άφιξης τέτοιων αιτήσεων. Το AZTEC υλοποιεί την «ουρά γεγονότων» του με μία μη φραγμένη ουρά (unbounded queue) που έχει ως αποτέλεσμα να μην απορρίπτεται καμία «αίτηση» αλλά το σύστημα να φτάνει σε «αδιέξοδο» (deadlock) σε περίπτωση πολύ μεγάλων ρυθμών λήψης νέων αιτήσεων. Όπως θα εξηγήσουμε και παρακάτω αυτό οφείλεται στο γεγονός ότι όλα τα «threads» καταλαμβάνονται από ολόκληρα «διαγράμματα ροής» και δεν υπάρχουν «threads» για να εκτελέσουν τις «απλά» ενέργειες. Για αυτό το λόγο η μέτρηση του ρυθμού εξυπηρέτησης διαδικασιών δεν έχει προς το παρόν νόημα μιας και εξαρτάται από το σημείο στο οποίο θα συμβεί το «αδιέξοδο» σε σχέση με το μέγεθος του «παραθύρου εκτέλεσης» (execution window) των πειραμάτων. Σαν δεύτερη μετρική χρησιμοποιήσαμε τον χρόνο καθυστέρησης ενός γεγονότος (είτε γεγονότος εκκίνησης, είτε γεγονότος εκτέλεσης) στην «ουρά γεγονότων» της μηχανής. Ο συνδυασμός της «μη-φραγμένης» ουράς γεγονότων με το «φραγμένο» «thread pool» από εξυπηρετητές μας κάνει να πιστεύουμε ότι αυτό είναι ένα πολύ ενδιαφέρον σημείο για «αξιολόγηση».

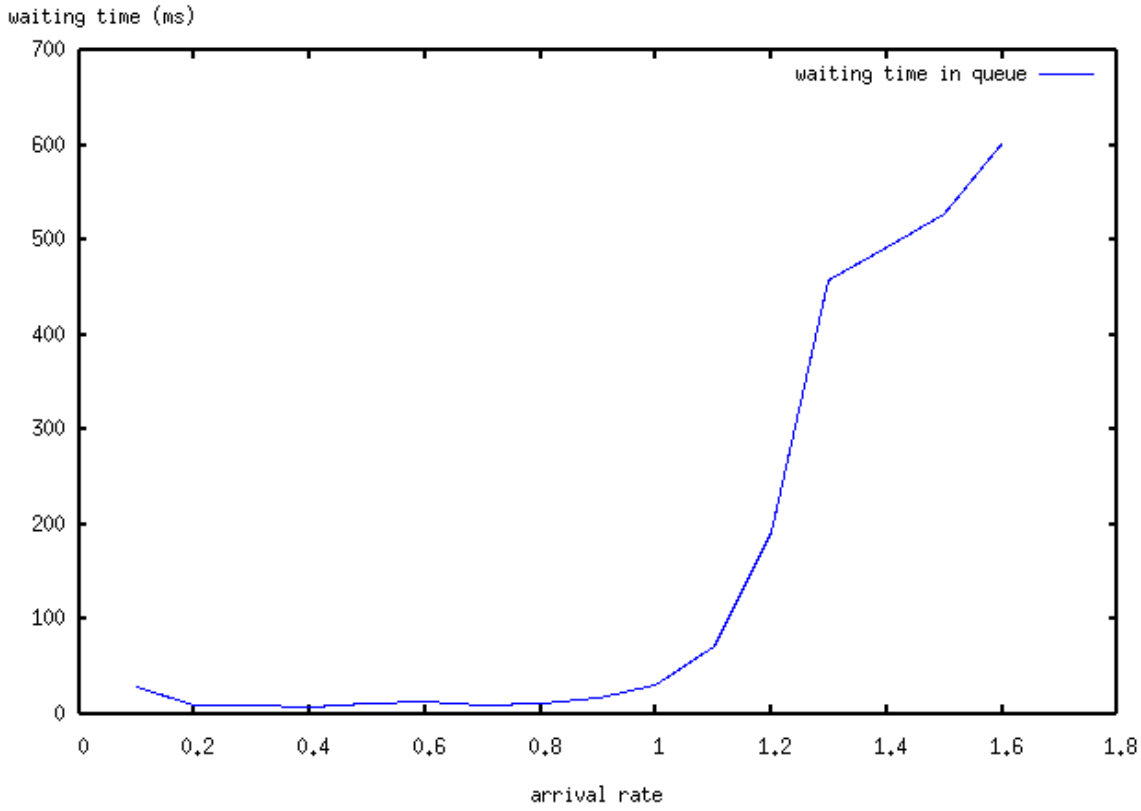
### 4.2.3 Περιβάλλον

Το περιβάλλον εκτέλεσης των πειραμάτων αποτελείται από τέσσερις συνιστώσες. Η πρώτη είναι μία συνθετική γεννήτρια αιτήσεων νέων διαδικασιών που χρησιμοποιεί μία

κατανομή Poisson για την δημιουργία αιτήσεων σε ένα συγκεκριμένο «παράθυρο χρόνου». Μεταβάλλοντας τον παράγοντα *λ* της κατανομής Poisson μπορούμε να επιτύχουμε από πολύ μικρούς ρυθμούς δημιουργίας αιτήσεων νέων διαδικασιών (π.χ. κατά μέσο όρο 1 αίτηση ανά 10 δευτερόλεπτα) μέχρι και πολύ μεγάλους ρυθμούς (π.χ κατά μέσο όρο 2 αιτήσεις το δευτερόλεπτο). Ο ρυθμός αυτός είναι αρκετά μεγάλος όμως θέλουμε να αξιολογήσουμε την συμπεριφορά του AZTEC ακόμα και σε τέτοιες ακραίες συνθήκες για να βρούμε τα «όρια» της «μηχανής εκτέλεσης». Η δεύτερη συνιστώσα είναι η συνιστώσα παρακολούθησης η οποία «παρακολουθεί» και καταχωρεί σε αρχείο (logs) τόσο τις χρονικές στιγμές δημιουργίας και τερματισμού διαδικασιών (σε συνδυασμό με τα αναγνωριστικά της κάθε διαδικασίας) όσο και της χρονικές στιγμές εισόδου και εξόδου κάθε γεγονότος από την «ουρά γεγονότων». Τα στοιχεία αυτά χρησιμοποιούνται για τον υπολογισμό των μέσων χρόνων εκτέλεσης διαδικασίας και καθυστέρησης στην ουρά αντίστοιχα σε σχέση με τον ρυθμό δημιουργίας νέων στιγμιοτύπων διαδικασίας για το συγκεκριμένο πείραμα. Οι εξωτερικές Ηλεκτρονικές Υπηρεσίες του παραδείγματος αναπαραστάθηκαν με την μορφή εφαρμογών «stub». Πρακτικά μία τέτοια εφαρμογή παίρνει κάποια δεδομένα εισόδου, «παγώνει» για ένα χρονικό διάστημα σχετικό με ποία υπηρεσία αναπαριστά και έπειτα επιστρέφει κάποια αποτελέσματα (ασφαλώς μπορεί και να μην επιστρέφει αποτελέσματα). Η πλατφόρμα AZTEC «τρέχει» σε ένα υπολογιστή με επεξεργαστή Pentium IV 2.7Gb και μνήμη RAM 512Mb.

### 4.2.4 Συμπεράσματα

Στην **Εικόνα 4.6** βλέπουμε την γραφική παράσταση της μέσης καθυστέρησης ενός γεγονότος στην «ουρά γεγονότων» σαν συνάρτηση του ρυθμού άφιξης αιτήσεων δημιουργίας νέων διαδικασιών. Στην **Εικόνα 4.7** βλέπουμε την γραφική παράσταση του μέσου χρόνου εκτέλεσης μίας διαδικασίας σαν συνάρτηση του ρυθμού άφιξης αιτήσεων δημιουργίας νέων διαδικασιών. Παρατηρούμε ότι από μικρούς ρυθμούς άφιξης (λ=0.2) έως και αρκετά μεγάλους (λ=1) το σύστημα συμπεριφέρεται με σταθερό τρόπο. Η μέση καθυστέρηση στην «ουρά γεγονότων» είναι πολύ μικρή (~2ms) και ο μέσος χρόνος εκτέλεσης των διαδικασιών είναι σταθερός. Σε αυτό το «διάστημα» ο ρυθμός λήψης νέων «γεγονότων» είναι μικρότερος από τον ρυθμό εξυπηρέτησης του «thread pool» με αποτέλεσμα να υπάρχει πάντα ένα «ελεύθερο thread» για να εκτελέσει κάθε «γεγονός». Αυτό το δεδομένο αλλάζει όταν ο ρυθμός λήψης αιτήσεων για νέες διαδικασίας γίνεται




**Εικόνα 4.6 Γραφική παράσταση της μέσης καθυστέρησης ενός γεγονότος στην ουρά γεγονότων σαν συνάρτηση του ρυθμού άφιξης νέων αιτήσεων για διαδικασίες**

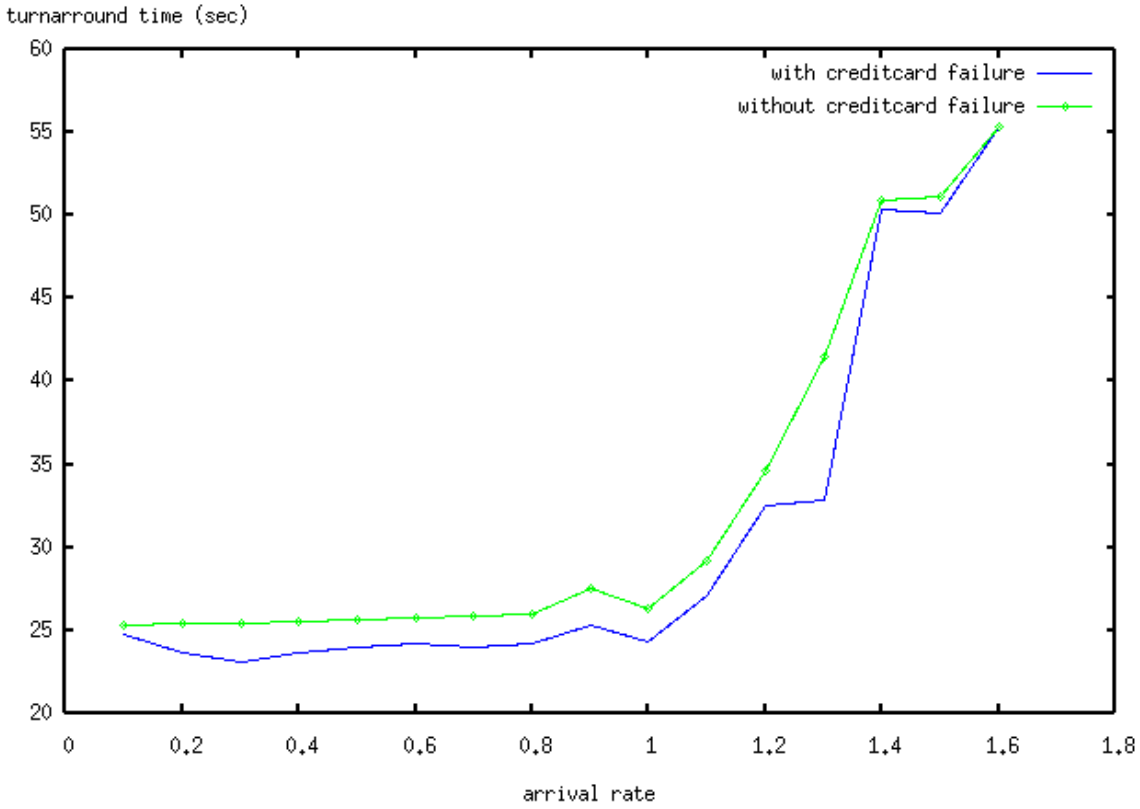
ακόμα μεγαλύτερος με τον παράγοντα λ της κατανομής «Poisson» να παίρνει τιμή μεγαλύτερη του 1[5].

Μία πρόκληση ήταν να βρεθεί το μέγιστο δυνατό μέγεθος του «thread pool» για το οποίο δεν «τελειώνει» η μνήμη του μηχανήματος. Για το συγκεκριμένο μηχάνημα που χρησιμοποιήσαμε αυτό το μέγεθος είναι τα 200 «threads» [6]. Στο συγκεκριμένο παράδειγμα που μελετάμε με ρυθμούς άφιξης γεγονότων για τους οποίους ο παράγοντας λ είναι μεγαλύτερος του 1 όλα τα «threads» του «pool» καταλαμβάνονται κάποια στιγμή με αποτέλεσμα τα «γεγονότα» να πρέπει να περιμένουν αρκετή ώρα στην ουρά έως ότου ελευθερωθεί κάποιο «thread» για να τα εκτελέσει. Αυτό το πρόβλημα γίνεται φυσικά όλο και πιο έντονο όσο ο ρυθμός άφιξής νέων γεγονότων αυξάνεται. Στην περίπτωση που ο παράγοντας λ της κατανομής «Poisson» της «γεννήτριας αιτήσεων» πάρει τιμή 1.8 ή παραπάνω θα παρατηρηθεί το φαινόμενο του αδιεξόδου. Ο λόγος γι'

[5] λ>1 πρακτικά σημαίνει ότι κατά μέσο όρο λαμβάνονται περισσότερες από 1 αιτήσεις το δευτερόλεπτο· λ=1.6 σημαίνει ότι κατά μέσο όρο λαμβάνονται περίπου 2~2.5 αιτήσεις το δευτερόλεπτο για την δημιουργία νέων διαδικασιών

[6] Να υπενθυμίσουμε ότι έχουμε «δυναμική» διαχείριση του «thread pool» αρχικοποιώντας το με μία αρκετά μικρή τιμή και αυξομειώνοντας το ανάλογα με τον αριθμό των ελεύθερων «threads» έως ένα πάνω όριο που του θέτουμε

αυτό είναι ότι ο αριθμός των νέων «γεγονότων εκτέλεσης» που μπαίνει στην ουρά είναι τόσο μεγάλος που κάποια στιγμή όλα τα «threads» θα είναι κατειλημμένα με «ολόκληρα» διαγράμματα ροής μη αφήνοντας ελεύθερα «threads» για την εκτέλεση των «απλών» ενεργειών.




**Εικόνα 4.7 Γραφική παράσταση του μέσου χρόνου εκτέλεσης μίας διαδικασίας σαν συνάρτηση του ρυθμού άφιξης νέων αιτήσεων για διαδικασίες**

Είναι προφανές ότι από τις μετρήσεις αυτές βγαίνουν κάποια πολύ χρήσιμα συμπεράσματα και κίνητρα για βελτιστοποίηση της «μηχανής εκτέλεσης». Μία τεχνική που ίσως δώσει μία λύση στο πρόβλημα είναι η δημιουργία δύο «ουρών γεγονότων» και δύο «pool of threads» αντί για μία. Η μία ουρά θα «χρησιμοποιείται» για τα «εξωτερικά γεγονότα εκτέλεσης» και θα είναι φραγμένη. Αυτό σημαίνει πώς όταν η ουρά (οπότε και το «thread pool» ) «γεμίζει» δεν θα «δέχεται» νέα γεγονότα έως ότου δημιουργηθεί ο κατάλληλος «ελεύθερος χώρος». Με αυτή την αλλαγή θα αποκτήσει νόημα και ο υπολογισμός του μέσου ρυθμού εξυπηρέτησης (throughput) μιας και θα υπάρχει ένα ποσοστό των γεγονότων που θα απορρίπτεται. Μία άλλη ιδέα που μπορεί να ερευνηθεί είναι η εκτέλεση ενός διαγράμματος ροής να μην αναλαμβάνεται εξ' ολοκλήρου από ένα «dedicated thread» από την αρχή έως το τέλος της εκτέλεσης του. Πράγματι «σύγχρονες» ενέργειες σε ένα διάγραμμα ροής οι οποίες διαρκούν αρκετή ώρα θα διατηρούν «κατειλημμένο» το «thread» χωρίς να είναι «ενεργό» αφού θα περιμένει μία απάντηση.

Βέβαια η υλοποίηση μίας τέτοιας τεχνικής δεν είναι τετριμμένη μιας και πρέπει να διατηρείται με κάποιο τρόπο η «κατάσταση» εκτέλεσης του διαγράμματος ροής το οποίο «σταμάτησε στη μέση». Όταν θα φτάνει η απάντηση το «υπόλοιπο» διάγραμμα ροής θα συνεχίζει την εκτέλεση του σε νέο «thread».

Ένας περιοριστικός παράγοντας είναι η υλοποίηση των μετρήσεων με ένα συγκεκριμένο παράδειγμα σχήματος διαδικασίας. Πολύ ενδιαφέρουσα θα ήταν η μελέτη των χαρακτηριστικών εκείνων των διαγραμμάτων ροής τα οποία θα δημιουργούσαν ενδιαφέρουσες μετρήσεις (π.χ μέγεθος διαγραμμάτων ροής, χρήση «ύπο-διαγραμμάτων ροής» αντί για μεγάλα «μονολιθικά» διαγράμματα ροής κ.τ.λ). Μία πιο άμεση προσέγγιση θα είναι η αναπαράσταση του σχήματος διαδικασίας του παραδείγματος Ηλεκτρονικού Εμπορίου με πιο «ευέλικτο» τρόπο ούτως ώστε τα διαγράμματα ροής να έχουν τον μικρότερο δυνατό χρόνο εκτέλεσης. Σαν συνολικό συμπέρασμα μπορούμε να πούμε ότι οι πρώτες αυτές μετρήσεις απόδοσης του ΑΖΤΕC χωρίς να είναι αρνητικές παρόλαυτα αποτελούν εφαλτήριο για περαιτέρω βελτιστοποιήσεις και της «μηχανής εκτέλεσης» και των σχεδιαστικών αρχών (design principles) των διαφόρων σχημάτων διαδικασίας.

# Κεφάλαιο 5

# Συμπεράσματα και Μελλοντική Εργασία

## 5.1 Συμπεράσματα

Το «παράδειγμα» Ηλεκτρονικών Υπηρεσιών έχει βρει μεγάλη υποστήριξη στον τομέα της «ηλεκτρονικής συνεργασίας» (collaboration) εξ'αιτίας των πολύ ελκυστικών ιδιοτήτων που το διέπουν όπως η δυναμική ανεύρεση «επιχειρησιακών συνεργατών» (business partners) και υπηρεσιών που ταιριάζουν στο μέγιστο δυνατό βαθμό στις απαιτήσεις οργανισμών, επιχειρήσεων ή και απλών εφαρμογών. Η «αρθρωτή» (modular) φύση των Ηλεκτρονικών Υπηρεσιών και η «διαλειτουργικότητα» (interoperability) που τις χαρακτηρίζει (εξ'αιτίας της «δεδομένης» (standard) διεπαφής τους άσχετα με το εσωτερικό περιβάλλον δημιουργίας τους) καθιστά δυνατή την επαναχρησιμοποίηση τους και την σύνθεση πολύπλοκων Ηλεκτρονικών Υπηρεσιών απο άλλες ήδη υπάρχουσες.

Στα πλαίσια αυτής της εργασίας αναγνωρίσαμε δύο μεγάλες κατηγορίες Ηλεκτρονικών Υπηρεσιών, τις διακριτές και τις Ηλεκτρονικές Υπηρεσίες που αναπτύσσουν συνεδρίες. Η σύνθεση διακριτών Ηλεκτρονικών Υπηρεσιών επιτυγχάνεται ικανοποιητικά απο τα μοντέλα σύνθεσης Ηλεκτρονικών Υπηρεσιών που έχουν προταθεί τα οποία περιγράφουν τον ορισμό μίας «υψηλού επιπέδου» διαδικασίας και την «ενορχήστρωση» της «κλήσης» διακριτών Ηλεκτρονικών Υπηρεσιών στα πλαίσια αυτής της διαδικασίας. Σε αυτά τα μοντέλα η αλληλεπίδραση της «διαδικασίας» με κάθε Ηλεκτρονική Υπηρεσία θεωρείται σαν μία ανεξάρτητη ενέργεια που δεν επηρεάζει τις υπόλοιπες Ηλεκτρονικές Υπηρεσίες.

Τέτοια μοντέλα δεν καταφέρουν να εκφράσουν με επιτυχία την σύνθεση Ηλεκτρονικών Υπηρεσιών που αναπτύσσουν συνεδρίες μιας και η ύπαρξη συνεδριών συνεπάγεται την δημιουργία ασύγχρονων γεγονότων στα οποία πρέπει η διαδικασία (δηλαδή η σύνθετη Ηλεκτρονική Υπηρεσία) να «αποκρίνεται» επηρεάζοντας ανάλογα τις υπόλοιπες ενεργές συνεδρίες (active sessions). Για τον σκοπό αυτό παρουσιάστηκε το μοντέλο ενεργών διαγραμμάτων ροής που είδαμε εκτενώς στην **Ενότητα 3** και η

πλατφόρμα AZTEC στην **Ενότητα 4** για την εκτέλεση τέτοιων διαγραμμάτων ροής. Συνοψίζοντας είδαμε ότι στο μοντέλο «ενεργών διαγραμμάτων ροής» ένα «σχήμα διαδικασίας» δεν αντιστοιχεί σε μία «υψηλού επιπέδου» διαδικασία αλλά σε ένα σύνολο απο ενεργά διαγράμματα ροής καθένα απο τα οποία ενεργοποιείται απο ένα τύπο γεγονότος και των οποίων η εκτέλεση έχει «αντίκτυπο» στην κατάσταση των ενεργών συνεδριών.

Στην **Ενότητα 3** μελετήσαμε την εκφραστικότητα του μοντέλου «ενεργών διαγραμμάτων ροής» και της γλώσσας XASC για τον ορισμό τους σε συνδυασμό με (in conjunction with) ένα σύνολο «προτύπων» (patterns) ροών εργασίας και επικοινωνίας τα οποία αναπαριστούν πρακτικά ένα πλήρες σύνολο απο απαιτήσεις οι οποίες μπορεί να «ζητηθούν» απο μία γλώσσα σύνθεσης Ηλεκτρονικών Υπηρεσιών ή ορισμού Ροών Εργασίας. Ολοκληρώνοντας αυτή τη μελέτη καταλήξαμε στο συμπέρασμα ότι η γλώσσα XASC δεν υστερεί σε εκφραστικότητα σε σχέση με τις «πρότυπες» γλώσσες «BPEL4WS» και «BPML». Απεναντίας η εισαγωγή του δομικού στοιχείου HParallel (που πρακτικά χειρίζεται την εκτέλεση πολλές φορές παράλληλα της ίδιας ενέργειας παραμετροποιημένης πιθανώς με τιμές απο μία ή περισσότερες λίστες δεδομένων) βοήθησε στην αναπαράσταση καταστάσεων τις οποίες οι άλλες γλώσσες είτε αδυνατούσαν να αναπαραστήσουν είτε τις αναπαριστούσαν με πολύπλοκο τρόπο.

Η αδυναμία «άμεσης» αναπαράστασης ορισμένων προτύπων απο την γλώσσα XASC πηγάζει απο την αδιαπραγμάτευτη σχεδιαστική επιλογή μας για την ύπαρξη ενός μικρού συνόλου δομικών στοιχείων τα οποία θα ακολουθούν τις βασικές αρχές των δομημένων Ροών Εργασίας (structured workflows) και θα έχουν «καθαρή» σημασιολογία που θα καθιστά εφικτή την παρακολούθηση της εκτέλεσης ενός διαγράμματος ροής, την επερώτηση και πιθανώς την αλλαγή της κατάστασης εκτέλεσης του. Επιπρόσθετα ορισμένα απο τα χαρακτηριστικά που αντιπροσώπευαν τα πρότυπα αυτά είναι εντελώς «ξένα» στο είδος των εφαρμογών που έχουμε σαν πρωταρχικό κίνητρο να αναπαραστήσουμε. Επίσης κάτι που θα αναφερθεί και στις προτάσεις για μελλοντική εργασία είναι ότι η αναπαράσταση του προτύπου «Ακύρωση Ενέργειας» απαιτεί την μελέτη των «ιδιοτήτων συναλλαγής» (transactional properties) του μοντέλου «ενεργών διαγραμμάτων ροής».

Όσων αφορά τα «επικοινωνιακά πρότυπα» τα συμπεράσματα είναι παρόμοια με την παρατήρηση ότι η «εντελώς» ασύγχρονη φύση του AZTEC προσδίδει μία διαφορετική φιλοσοφία επικοινωνίας μεταξύ των διαγραμμάτων ροής και των εξωτερικών Ηλεκτρονικών Υπηρεσιών. Πιο συγκεκριμένα στο μοντέλο «ενεργών διαγραμμάτων ροής» δεν υποστηρίζεται η ύπαρξη ενός δομημένου στοιχείο το οποίο θα

«μπλοκάρει» περιμένοντας για την λήψη ενός μηνύματος ή ένα γεγονότος απο την στιγμή που εγγενώς το μοντέλο υποστηρίζει την αντιστοίχηση «εισερχόμενων» μηνυμάτων ή γεγονότων σε ενέργειες (διαγράμματα ροής). Παρόλαυτά αναγνωρίζοντας την σημασία της ύπαρξης και «ρητής» αναμονής για ένα γεγονός ή μήνυμα σε ορισμένες περιπτώσεις προβλημάτων, θα προτείνουμε στην μελλοντική εργασία την επέκταση ενός απο τα ήδη υπάρχοντα δομικά στοιχεία ούτως ώστε να έχει και τέτοια λειτουργικότητα.

## *5.2 Μελλουτική Εργασία*

Οι προτάσεις για επεκτάσεις και μελλοντική εργασία τόσο για το μοντέλο «ενεργών διαγραμμάτων ροής» όσο και για την πλατφόρμα AZTEC είναι αρκετές μιας και τα δύο βρίσκονται σε μία πρώτη έκδοση ή οποία επιτρέπει αρκετές προσθήκες ορισμένες απο τις οποίες έχουμε ήδη αναφέρει σε προηγούμενα σημεία.

Η «δομημένη» αναμονή για ένα μήνυμα ή γεγονός (δηλαδή με την χρήση ενός δομικού στοιχείου) δεν υποστηρίζεται προς το παρόν στο μοντέλο «ενεργών διαγραμμάτων ροής» μιας και η «απόκριση» και ο χειρισμός ασύγχρονων γεγονότων είναι το βασικό χαρακτηριστικό αυτού του μοντέλου. Παρόλαυτα αναγνωρίζουμε ότι η αναμονή για ένα γεγονός στα πλαίσια της εκτέλεσης ενός διαγράμματος ροής θα ήταν χρήσιμη (ή και απαραίτητη) σε ορισμένες κατηγορίες προβλημάτων. Για παράδειγμα στην περίπτωση που θέλουμε να επιτρέψουμε την «εκτέλεση» ενός μόνο γεγονότος απο ένα σύνολο υποψηφίων (αυτό που θα συμβεί πρώτο), αυτό απαιτεί πολύπλοκες ενέργειες συγχρονισμού χρησιμοποιώντας το «Context Repository» σαν μέρος αμοιβαίου αποκλεισμού που θα επιτρέπει μόνο στο πρώτο γεγονός να συμβεί. Για τον σκοπό αυτό προτείνουμε τον εμπλουτισμό της λειτουργικότητας του δομικού στοιχείου «Switch» της γλώσσας XASC. Η σημασιολογία του «Switch» είναι ότι επιλέγει ένα ανάμεσα σε πολλαπλά «μονοπάτια εκτέλεσης» ανάλογα με κάποια «δεδομένα ελέγχου» (control data). Η σημασιολογία αυτή μπορεί να εμπλουτιστεί ούτως ώστε να επιλέγει ένα ανάμεσα σε πολλαπλά «μονοπάτια εκτέλεσης» ανάλογα με την δημιουργία κάποιου γεγονότος ή της ολοκλήρωσης κάποιας «διορίας» (timeout).

Η μελέτη των «ιδιοτήτων συναλλαγής» (transactional properties) είναι μία άλλη εργασία που πρέπει να γίνει σχετικά με το μοντέλο «ενεργών διαγραμμάτων ροής» ούτως ώστε να είναι εφικτή η «ακύρωση» της εκτέλεσης μεμονωμένων ενεργειών αλλά και η δημιουργία ενεργειών «αντιστάθμισης» (compensation) για διαγράμματα ροής ή για «ομάδες ενεργειών» ενός διαγράμματος ροής.

Στα πλαίσια της πλατφόρμας AZTEC απαιτείται η μελέτη των τεχνικών ανάθεσης προτεραιοτήτων εκτέλεσης στα διαγράμματα ροής μίας διαδικασίας. Υπάρχει η επιλογή μεταξύ «απόλυτων» και «σχετικών» προτεραιοτήτων όπως αναφέραμε στην **Ενότητα 4.** Στην δεύτερη περίπτωση πρέπει να ερευνηθεί τι είδους συσχετίσεις μπορούν να επηρεάσουν την προτεραιότητα εκτέλεσης ανάμεσα σε δυο διαγράμματα ροής.

Ένας από τους μελλοντικούς στόχους μας είναι η υλοποίηση μίας συνιστώσας Σύνθεσης (Assembly component) η οποία θα παρέχει την δυνατότητα για «αυτόματη» δημιουργία «σχημάτων διαδικασιών» (process schemas) χρησιμοποιώντας ορισμούς υψηλότερου επιπέδου (higher-level specifications). Προς το παρόν, το «φόρτωμα» (loading) των «σχημάτων διαδικασίας» (process schemas) στην Μηχανή Εκτέλεσης γίνεται «στατικά» από την συνιστώσα «Διαχείριση Σχήματος» (Schema Management). Στόχος μας είναι η επίτευξη δυναμικής «Διαχείρισης Σχήματος» (dynamic Schema Management που θα επιτρέπει αλλαγές και τροποποιήσεις στο «σχήμα» των διαδικασιών όχι μόνο κατά την διάρκεια της σχεδίασης (design-time) αλλά και κατά την διάρκεια της εκτέλεσης (run-time) μίας διαδικασίας. Αυτές οι αλλαγές θα γίνονται με χρήση της συνιστώσας Σύνθεσης (Assembly component). Όπως έχουμε προαναφέρει η συμμετοχή του ανθρώπινου παράγοντα είναι σημαντική μέσω της συνιστώσας Διαχείρισης (Administration component). Βέβαια η συνιστώσα Διαχείρισης θα αποκτήσει ιδιαίτερη σημασία όταν θα είναι πλήρως λειτουργική η συνιστώσα Σύνθεσης (Assembly component) οπότε και οι προγραμματιστές θα αναπτύσσουν τα «πρότυπα» «σχημάτων διαδικασίας» (patterns) πάνω στα οποία θα στηρίζεται η αυτόματη δημιουργία «σχημάτων διαδικασίας». Επίσης οι προγραμματιστές (συνιστώσα Διαχείρισης) θα καθορίζουν τις πολιτικές που θα χρησιμοποιούνται κατά την δημιουργία «σχημάτων διαδικασίας» αλλά θα είναι και αυτοί που πιθανώς θα διεξάγουν τις δυναμικές αλλαγές στο σχήμα μίας διαδικασίας κατά την διάρκεια της εκτέλεσης της.

# *Βιβλιογραφία*

**[A98]** W.M.P. van der Aalst "*The Application of Petri Nets to Workflow Management*". The Journal of Circuits, Systems and Computers, 8(1):21--66, 1998.

**[ADH+02]** W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. "*Pattern-Based Analysis of BPML (and WSCI).*" QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.

**[AHK+02]** W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. "Workflow patterns". Technical report FIT-TR-2002-2, Faculty of IT, Queensland University of Technology, July 2002. Accessed from <http://www.tm.tue.nl/it/research/patterns>. To appear in Distributed and Parallel Databases, Kluwer.

**[APACHE2]** Apache HTTP server version 2: Apache worker Module, Available at < http://ejds.org/manual_apache/mod/worker.html>

**[C97]** John Calcote, "Thread Pools and Server Performance", Dr. Dobb's Journal, pp 60-64, July 1997

**[CASTOR]** The Castor project, Available at < http://www.castor.org/>

**[CHK+01]** V. Christophides, R. Hull, G. Karvounarakis, A. Kumar, G. Tong, and M. Xiong. "*Beyond discrete e-services: Composing session-oriented services in telecommunications*". In *Proc. of Workshop on Technologies for E-Services (TES), Springer LNCS volume 2193*, Rome, Italy, September 2001.

**[CHK01]** V. Christophides, R. Hull, and A. Kumar. "*Querying and splicing of XML workflows.*" In *Proc. of Intl. Conf. on Cooperating Information* Systems (CoopIS), 2001

**[CORBA]** Corba Technology homepage http://www.corba.org

**[DCOM]** DCOM, Available at: http://msdn.microsoft.com/library/backgrnd/ html/ msdn_dcomarch.htm

**[DOM]** Document Object Model (DOM), Available at <http://www.w3.org/DOM/>

**[EJB]** Enterprise Java Beans, http://java.sun.com/products/ejb/.

**[F93]** R. Fehling, "A Concept of Hierarchical Petri Nets with Building Blocks" APN'93 and also LNCS 674, 1993, pp 148-168

**[FGP+97]** F. Casati, P. Grefen, B. Pernici, G.Pozzi, and G. Sanchez. "*WIDE workflow model and architecure*", 1997. http://www.sema.es/projects/WIDE/Documents/

**[H87]** D. Harel. "*Statecharts: A visual formalism for complex systems*". Science of Computing, 8:231-274, 1987

**[H90]** D. Harel et al., "*STATEMATE: A Working Environment for the Development of Complex Reactive Systems*", IEEE Transactions on Software Engineering, 16(4), 1990

**[HJS91]** P. Huber, K. Jensen and R.M. Shapiro, "*Hierarchies in Coloured Petri-Nets*", APN'90 and also LNCS 483, 1991, pp 313-341

**[J92]** K. Jensen. "*Coloured Petri-Nets. Basic Concepts Analysis Methods and Practical Use*", volume 1, Basic Concepts of Monographs in Theoretical Computer Science. Springer – Verlag 1992

**[J94]** K. Jensen. "*Coloured Petri-Nets. Basic Concepts Analysis Methods and Practical Use*", volume 2, Basic Concepts of Monographs in Theoretical Computer Science. Springer – Verlag 1994

**[J97]** K. Jensen. "Coloured Petri-Nets. Basic Concepts Analysis Methods and Practical Use", volume 3, Basic Concepts of Monographs in Theoretical Computer Science. Springer – Verlag 1997

**[JAXB]** Java Architecture for XML Binding (JAXB), Available at < http://java.sun.com/xml/jaxb/>

**[JBIND]** JBind – A Java - XML binding framework, Available at < http://jbind.sourceforge.net/>

**[JDOM**] JDOM, Available at < http://www.jdom.org/docs/faq.html>

**[JINI]** Jini, http://www.sun.com/jini

**[KHB00]** B. Kiepuszewski, A. ter Hofstede and C. Bussler, "*On Structured Workflow Modelling*", Proceedings of CAISE 2000, Stockholm, Sweden.

**[L01]** Leymann F. "Web Services Flow Language", 2001 Available at: <http:www-306.ibm.com/software/solutions/ webservices/pdf/WSFL.pdf>

**[LA94]** F. Leymann and W. Altenhuber "*Managing business processes as an information resource*". IBM Systems Journal, Volume 33, Number 2, Page 326, 1994.

**[LB96]** Bil Lewis and Daniel J. Berg, "*Threads Primer-A Guide to Mutithreaded Programming*", SunSoft Press A Prentice Hall Title, 1996

**[LD94]** F. Leymann and D. Roller "*Business process management with flowmark*". In Proc. of COMPCON Spring 1994. IEEE, 1994.

**[LD97]** F. Leymann and D. Roller "*Workflow-based applications*", IBM Systems Journal, Volume 36, Number 1, 1997

**[LM95]** Lei, K. and Singh, M. "*A Comparison of Workflow Metamodels*", Proceedings of the ER-97 Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling, Los Angeles, CA 1995

**[LML00]** Yibei Ling, Tracy Mullen and Xiaola Lin, "Analysis of Optimal Thread Pool Size" Operation Systems Review, 2000

**[M89]** Mourata Tadao, "*Petri Nets: Properties, Analysis and Applications*", Proceedings of the IEEE, Vol. 77, No. 4, April, pp. 541-580, 1989

**[NJLRC]** NJLRC – New Jersey Law Revision Commission, *Draft Final Report relating to Standard Form Contracts*, New Jersey Law Revision Commission, USA, 1998.

**[PN]** Petri Nets: Tools And Software. http://www.daimi.au.dk/PetriNets/tools/

**[QUICK]** The Quick project Homepage, Available at < http://qare.sourceforge.net/web/2001-12/products/index.html#quick>

**[R74]** C. Ramchandani. "Analysis of asynchronous concurrent systems by timed Petri nets." PhD thesis, MIT, Boston, 1974.

**[RMB01]** W.A. Ruh, F.X. Maginnis, and W.J. Brown. "Enterprise Application Integration: A Wiley Tech Brief." John Wiley and Sons, Inc, 2001.

**[S98]** Douglas C Schmidt, "*Evaluating Architectures for Multithreaded Object Request Brokers*". Association for Computing Machinery, Communication of the ACM;New York, Vol 41, no. 10, pp 54-60, Oct. 1998

**[SOAP]** Simple Object Access Protocol (SOAP) 1.1, W3C Note 08, May 2000, http://www.w3.org/TR/SOAP/.

**[T01]** Thatte, S (ed.). "XLANG", 2001. Available at: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

**[T03]** Thatte, S (ed.). "Business Process Execution Language for Web Services", Version 1.1 May 2003. Available at:<http://www.ibm.com/developerworks/library/ws-bpel/>

**[UDDI**] IBM UDDI Registry, http://www-3.ibm.com/services/uddi/.

**[WAD+02]** P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. "*Pattern-Based Analysis of BPEL4WS*." QUT Technical report, FIT-TR-2002-04, Queensland University of Technology, Brisbane, 2002.

**[WFMC]** Workflow Management Coalition: The Workflow Reference Model http://www.wfmc.org

**[WSA]** Web Services architecture overview: The next stage of evolution for e-business, http://www-106.ibm.com/developerworks/library/w-ovr/?dwzone=ws

**[WSDL]** WSDL, http://msdn.microsoft.com/xml/general/wsdl.asp.

**[WT]** Workflow Technology- an introduction. White Paper

**[WW96]** D. Wodtke, G. Weikum, "*A Formal Foundation for Distributed Workflow Execution Based on State Charts*", Technical Report, University of Saarbruecken, 1996

**[XM]** XMethods, http://www.xmethods.com

**[XML]** W3C Extensible Markup Language (xml) 1.0 http://www.w3.org/TR/REC-xml

**[XMLJ]** XML and Java technologies: Data binding Part2: Performance Available at < http://www-106.ibm.com/developerworks/xml/library/x-databdopt2/>

**[XPATH]** The XML Path Language (XPath), Available at <http://www.w3.org/TR/xpath>

**[ZEUS]** The Enhydra Zeus project, Available at < http://zeus.objectweb.org/>