ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη χρήσης ουρών μηνυμάτων ως μέσου επικοινωνίας σε marketplace υπηρεσιών Cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αργυρή-Ελπίδα Δ. Κουϊμιτζή

Επιβλέπουσα : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ ΗΛΕΚΤΡΟΝΙΚΗΣ ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ
ΠΛΗΡΟΦΟΡΙΚΗΣ

Μελέτη χρήσης ουρών μηνυμάτων ως μέσου επικοινωνίας σε marketplace υπηρεσιών Cloud

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αργυρή-Ελπίδα Δ. Κουϊμιτζή

Επιβλέπουσα : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 29η Φεβρουαρίου 2016.

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Βασίλειος Λούμος	Συμεών Παπαβασιλείου
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Φεβρουάριος 2016

.....
Αργυρή-Ελπίδα Δ. Κουϊμιτζή

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά την καθηγήτρια Θεοδώρα Βαρβαρίγου, που μου έδωσε την ευκαιρία να δουλέψω στο εργαστήριο Distributed Knowledge and Media Systems Group, καθώς επίσης και για τις πολύτιμες συμβουλές τις.
Επίσης θα ήθελα να ευχαριστήσω ιδιαίτερα τους Μουλό Βρεττό και Κωνσταντίνο Στεφανάτο, οι οποίοι μου αφιέρωσαν χρόνο και όρεξη και μου έδωσαν πολλές και χρήσιμες συμβουλές για προβλήματα υλοποίησης. Η υπομονετική καθοδήγησή τους ήταν καθοριστικής σημασίας για την εκπόνηση αυτής της διπλωματικής εργασίας.
Τέλος, θα ήθελα να ευχαριστήσω τους γονείς μου για την αγάπη και την στήριξή τους, όλα αυτά τα χρόνια για να εκπληρωθούν τα όνειρά μου, καθώς και τον αδερφό μου Νίκο, τον Μιχάλη και όλους τους φίλους μου για την αμέριστη κατανόηση και υποστήριξη τους.

Περίληψη

Η ραγδαία ανάπτυξη της τεχνολογίας Cloud καθώς και η ολοένα συχνότερη ανάπτυξη εφαρμογών και συστημάτων που βασίζονται σε τέτοιες υπηρεσίες, μετατρέπει την διαδικασία αναζήτησης του κατάλληλου κάθε φορά συνδυασμού υπηρεσιών Cloud, σε ένα μια υπόθεση ιδιαίτερα δύσκολη και πολύπλοκη, αφού ο όγκος των δεδομένων που πρέπει να επεξεργαστεί κανείς για να καταλήξει στον συνδυασμό που προσφέρει την αποδοτικότερη λύση, είναι τόσο μεγάλος και έχει τέτοια χαρακτηριστικά, που ανάγει την διαδικασία αυτή σε πρόβλημα Big Data.
Στα πλαίσια της παρούσας διπλωματικής εργασίας, αναπτύχθηκε ένα σύστημα που υλοποιεί μια πλατφόρμα-αγορά (marketplace) Cloud, βασισμένη στο marketplace 4CaaSt και το οποίο χρησιμοποιεί διαφορετικό τρόπο επικοινωνίας ανάμεσα στα διάφορα στοιχεία του, αφού χρησιμοποιεί ουρές μηνυμάτων (message queues) αντί για πακέτα δεδομένων. Στόχος της παρούσας διπλωματικής εργασίας, είναι η συγκριτική μελέτη δύο ξεχωριστών τρόπων χρήσης των ουρών μηνυμάτων ως μέσου επικοινωνίας του marketplace, όπου στον πρώτο χρησιμοποιούνται πολλές ξεχωριστές ουρές, ενώ στον δεύτερο μία κοινή ουρά.

Λέξεις-κλειδιά
Cloud, Message Queues, Big Data, Marketplace, ActiveMQ, 4CaaSt

Abstract

The rapid development of Cloud technology and the increasingly more frequent development of applications and systems based on such services, converts the procedure of searching every time, for the right combination of Cloud services, in a very difficult and complex task, because the volume of data, that has to be processed to end up in the combination which offers the more efficient solution, is so big and has such characteristics that makes this whole procedure into a Big Data problem.
In this thesis, the system which was developed, implements a Cloud marketplace, based on the 4CaaSt marketplace and uses a different way of communication between the components of the system, since it uses message queues instead of data packages. The aim of this thesis is the comparison of using message queues as the way of communication of the marketplace, in two different ways: in the first one, many different queues are used between the different components of the system and in the second one, there is only one common queue.

Keywords

Cloud, Message Queues, Big Data, Marketplace, ActiveMQ, 4CaaSt

Περιεχόμενα

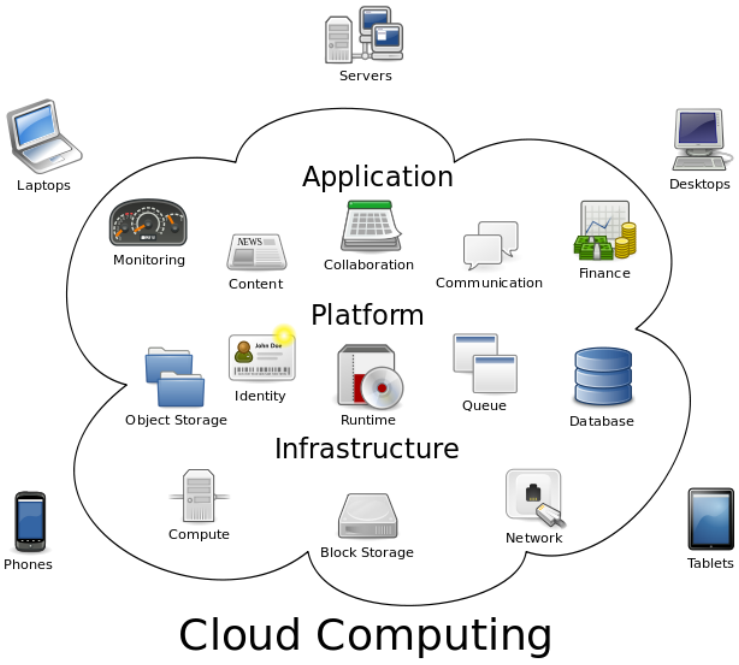
Πίνακας σχημάτων

Πίνακας διαγραμμάτων

Κεφάλαιο 1:Cloud

1.1 Εισαγωγή

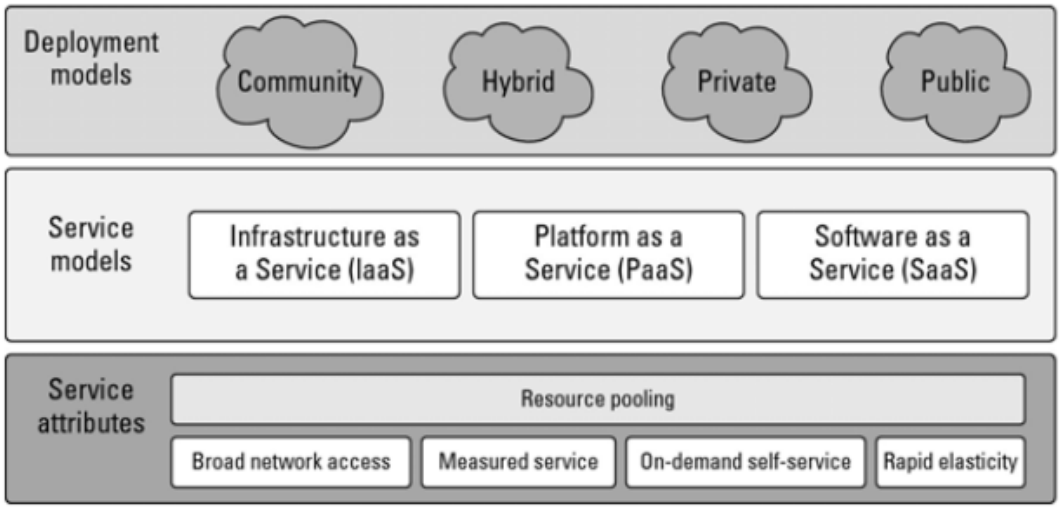
Το Cloud (υπολογιστικό νέφος) αποτελεί την πιο διαδεδομένη τεχνολογία των τελευταίων ετών ενώ η παρουσία της είναι αισθητή τόσο στην καθημερινή ζωή όσο και στον επαγγελματικό τομέα. Υπάρχουν αρκετά παραδείγματα δημοφιλών εφαρμογών και υπηρεσιών που χρησιμοποιούν την τεχνολογία αυτή και τα οποία προϋπήρχαν του ονόματός της. Κάποια από αυτά είναι το Facebook, το Gmail, το Hotmail, το Skype, το Paypal, κτλ.
Οι αυξημένες ανάγκες σε υπολογιστική ισχύ και αποθηκευτικό χώρο παράλληλα με την ραγδαία ανάπτυξη των τεχνολογιών δικτύωσης και επικοινωνίας μέσω τοπικών ή απομακρυσμένων δικτύων και την αύξηση της παραγωγής συσκευών προσιτών στο κοινό (λόγω ευκολίας χρήσης και μειωμένου κόστους) που αξιοποιούν τις τεχνολογίες αυτές, συνέβαλαν στη βελτίωση και στην περαιτέρω ανάπτυξη της τεχνολογίας του Cloud.



Σχήμα 1: Το μοντέλο του Cloud (πηγή: Wikipedia "Cloud computing")

1.2 Ορισμός Cloud

Το Cloud (Υπολογιστικό νέφος), ορίζεται σύμφωνα με το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας (NIST)[1], ως ένα μοντέλο που επιτρέπει την πανταχού παρούσα, άνετη και κατ'απαίτηση δικτυακή πρόσβαση σε μια κοινόχρηστη δεξαμενή υπολογιστικών πόρων (για παράδειγμα δίκτυα, εξυπηρετητές (servers), αποθηκευτικό χώρο, εφαρμογές και υπηρεσίες), που μπορούν να γίνουν διαθέσιμοι με ελάχιστη προσπάθεια διαχείρισης ή αλληλεπίδρασης με τον πάροχο της υπηρεσίας. Αυτό το μοντέλο Cloud διαθέτει 5 απαραίτητα χαρακτηριστικά ενώ κατηγοριοποιείται σε 3 διαφορετικά μοντέλα υπηρεσίας και σε 4 διαφορετικά μοντέλα ανάπτυξης.



Σχήμα 2: The NIST cloud computing definition (πηγή: Cloud Computing Bible, Barrie Sosinsky)

1.3 Χαρακτηριστικά Cloud

Σε συνέχεια του ορισμού του Cloud, το Εθνικό Ινστιτούτο Προτύπων και Τεχνολογίας, θεωρεί ως απαραίτητα χαρακτηριστικά του Cloud, τα παρακάτω:

1) **Εξυπηρέτηση κατ'απαίτηση**: Ένας πελάτης μπορεί να προμηθευτεί υπολογιστικούς πόρους χωρίς την ανάγκη αλληλεπίδρασης με τον πάροχο υπηρεσιών Cloud.

2) **Ευρεία πρόσβαση δικτύου**: Η πρόσβαση στους πόρους του Cloud είναι διαθέσιμη μέσω δικτύου χρησιμοποιώντας τυποποιημένες μεθόδους με τέτοιο τρόπο που παρέχει πρόσβαση, ανεξάρτητη από την πλατφόρμα, σε όλους τους τύπους των πελατών. Αυτό ωστόσο, περιλαμβάνει ένα μείγμα ετερογενών λειτουργικών συστημάτων και πλατφορμών, όπως φορητοί υπολογιστές (laptops), κινητά τηλέφωνα και ταμπλέτες.

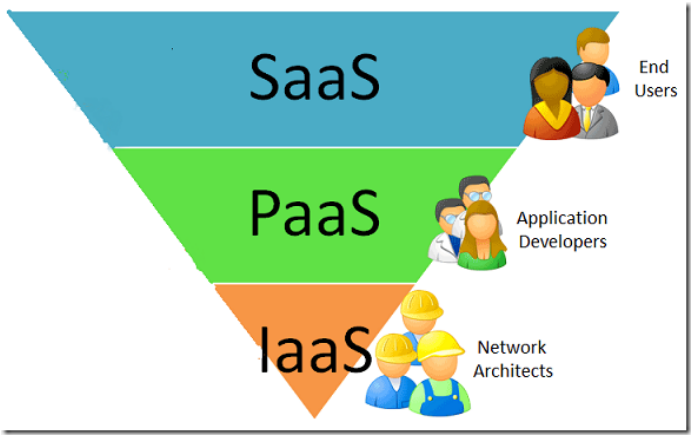
3) **Δυναμική απόκτηση και επανεκχώρηση πόρων**: Ο πάροχος υπηρεσιών Cloud δημιουργεί πόρους που συνδυάζονται σε ένα σύστημα που υποστηρίζει πολλούς και διαφορετικούς χρήστες. Φυσικά και εικονικά συστήματα εκχωρούνται ή επανεκχωρούνται δυναμικά σύμφωνα με τις εκάστοτε ανάγκες. Υπάρχει μια αίσθηση ανεξαρτησίας της τοποθεσίας , με την έννοια ότι ο πελάτης δεν έχει κανέναν έλεγχο ή γνώση της ακριβούς τοποθεσίας των παρεχομένων πόρων, εκτός ίσως από έναν γενικότερο προσδιορισμό αυτής όπως για παράδειγμα η χώρα ή η περιοχή. Μερικά παραδείγματα υπολογιστικών πόρων συμπεριλαμβάνουν αποθηκευτικό χώρο, επεξεργαστές, μνήμη και εύρος ζώνης δικτύου.

4) **Γρήγορη ελαστικότητα**: Οι πόροι μπορούν να διατεθούν γρήγορα και ελαστικά. Το σύστημα μπορεί να προσθέσει πόρους, είτε χρησιμοποιώντας πιο δυνατούς υπολογιστές είτε χρησιμοποιώντας περισσότερους υπολογιστές του ίδιου είδους. Στον καταναλωτή, οι διαθέσιμες δυνατότητες μοιάζουν απεριόριστες και μπορούν να διατεθούν σε οποιαδήποτε ποσότητα και οποιαδήποτε στιγμή.

5) **Μετρούμενη υπηρεσία**: Οι πόροι Cloud κοστολογούνται ανάλογα με τη χρήση τους. Ένας πελάτης δηλαδή, χρεώνεται βάσει ενός γνωστού μετρικού συστήματος όπως το μέγεθος του αποθηκευτικού χώρου που χρησιμοποιήθηκε, ο αριθμός των συναλλαγών, η Είσοδος/Εξοδος ή το εύρος του δικτύου, το μέγεθος της υπολογιστικής δύναμης που χρησιμοποιήθηκε κ.ο.κ.

1.4 Μοντέλα υπηρεσιών

Κατά την εξέλιξη του Cloud, διαφορετικοί πάροχοι προσφέρουν Cloud που συνδέονται με διαφορετικές υπηρεσίες το καθένα. Ανάλογα με τον συνδυασμό υπηρεσιών που προσφέρουν, τα Cloud χωρίζονται σε διαφορετικά μοντέλα υπηρεσιών (Σχήμα 3), τα οποία έχουν την εξής μορφή: "XaaS" ή "Something as a Service"[2].



Σχήμα 3: Τα μοντέλα υπηρεσιών του Cloud (πηγή: http://blog.samisa.org/2011/07/cloud-computing-explained.html)

Υπάρχουν τρία είδη υπηρεσιών που είναι καθολικά αποδεκτά:

1) **Υποδομή ως υπηρεσία (Infrastructure as a Service-IaaS)**: Παρέχει εικονικές μηχανές, εικονικό αποθηκευτικό χώρο, εικονική υποδομή και άλλα υλικά στοιχεία ως πόρους που μπορούν οι πελάτες να προμηθευτούν. Ο πάροχος υπηρεσιών IaaS χειρίζεται όλη την υποδομή, ενώ ο πελάτης είναι υπεύθυνος για όλες τις πτυχές της ανάπτυξης. Αυτές μπορεί να περιλαμβάνουν το λειτουργικό σύστημα, εφαρμογές και αλληλεπιδράσεις χρηστών με το σύστημα.

Παραδείγματα παρόχων IaaS : Amazon Elastic Compute Cloud (EC2), AT&T, bluelock, ca technologies, cloud scaling, DataPipe, Eucalyptus Systems, GoGrid, HP, logicworks, NaviSite, OpSource, RackSpace Cloud, Terremark, Verizon, etc.

2) **Πλατφόρμα ως υπηρεσία (Platform as a Service-PaaS)**: Παρέχει εικονικές μηχανές, λειτουργικά συστήματα, εφαρμογές, υπηρεσίες, πλαίσια ανάπτυξης, συναλλαγές και δομές ελέγχου. Ο πελάτης μπορεί να αναπτύξει τις εφαρμογές του στο Cloud ή να χρησιμοποιήσει εφαρμογές που προγραμματίστηκαν χρησιμοποιώντας γλώσσες και εργαλεία που υποστηρίζονται από τον πάροχο υπηρεσιών PaaS. Ο πάροχος των υπηρεσιών διαχειρίζεται την υποδομή νέφους, τα λειτουργικά συστήματα και το απαιτούμενο λογισμικό. Ο πελάτης είναι υπεύθυνος για την εγκατάσταση και την διαχείριση της εφαρμογής που αναπτύσσει.

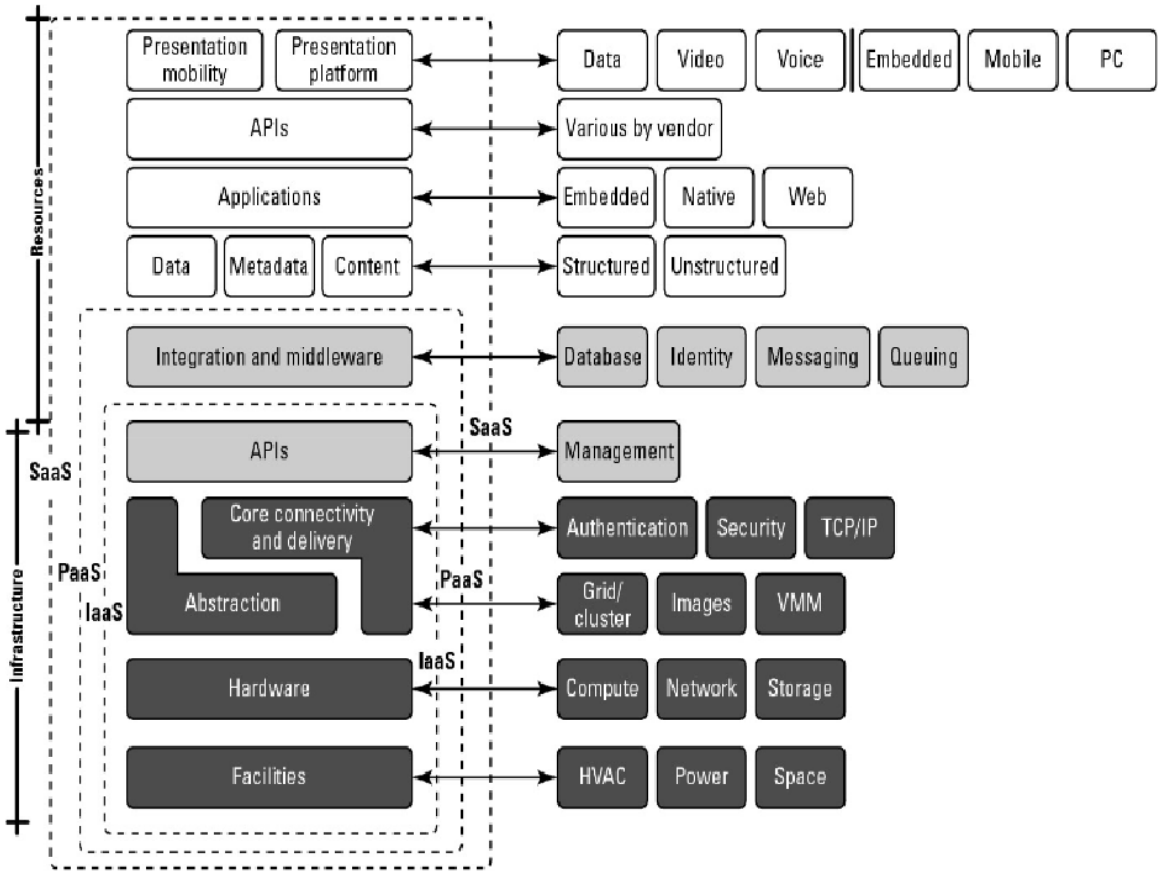
Παραδείγματα παρόχων PaaS: Amazon Web Services, AppScale, salesforce.com, Google, Windows Azure, GridGain, openstack, ThinkGrid, κλπ.

3) **Λογισμικό ως υπηρεσία (Software as a Service-SaaS)**: Πρόκειται για ένα πλήρως λειτουργικό περιβάλλον με εφαρμογές, διαχείριση και διεπιφάνεια χρήστη. Στο μοντέλο SaaS, η εφαρμογή παρέχεται στον πελάτη μέσω μιας διεπιφάνειας (συνήθως μέσω ενός προγράμματος περιήγησης-browser) και η ευθύνη του πελάτη αρχίζει και τελειώνει με την είσοδο του σε αυτήν την διεπιφάνεια και την διαχείριση των δεδομένων

του καθώς και με οποιαδήποτε άλλη αλληλεπίδραση χρήστη. Για οτιδήποτε από την εφαρμογή μέχρι την υποδομή, είναι υπεύθυνος ο πάροχος.
Παραδείγματα παρόχων SaaS: Abiquo, Accelops, Akamai, AppDynamics, MeghaWare, CloudSwitch, Cumulux, Eloqua, Oracle On Demand, Marketo, SAP BusinessByDesign, etc.

Τα τρία διαφορετικά μοντέλα υπηρεσιών σαν σύνολο, είναι γνωστά ως το SPI μοντέλο του Cloud. Έχουν αναφερθεί πολλά άλλα μοντέλα υπηρεσιών: Αποθηκευτικός χώρος ως υπηρεσία (Storage as a Service- StaaS), Ταυτότητα ως υπηρεσία (Identity as a Service- IdaaS) κ.ο.κ. Παρόλα αυτά, οι υπηρεσίες SPI υπερκαλύπτουν όλες τις άλλες πιθανότητες.

Είναι χρήσιμο να σκεφτεί κανείς τα μοντέλα υπηρεσιών του Cloud σαν μια στοίβα υλικού/λογισμικού. Μια τέτοια αναπαράσταση καλείται Αναφορικό Μοντέλο Υπολογιστικού Νέφους (Cloud Reference Model) και φαίνεται στο παρακάτω σχήμα (Σχήμα 4). Στον πάτο της στοίβας βρίσκεται το υλικό ή η υποδομή που περιλαμβάνει το δίκτυο. Κινούμενος προς την κορυφή της στοίβας, κάθε μοντέλο υπηρεσιών κληρονομεί τις δυνατότητες του μοντέλου υπηρεσιών που βρίσκεται παρακάτω από αυτό. Το μοντέλο IaaS έχει τα χαμηλότερα επίπεδα ενσωματωμένης λειτουργικότητας και ενσωμάτωσης, και το μοντέλο SaaS έχει τα υψηλότερα.

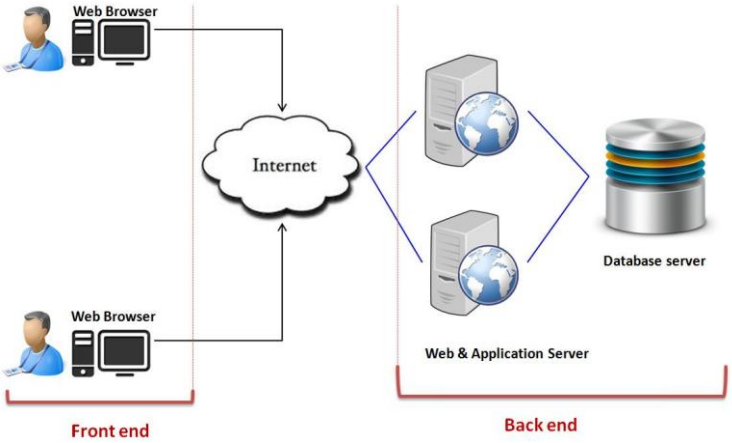


Σχήμα 4: Το Αναφορικό Μοντέλο Cloud (Cloud Reference Model) (πηγή: Cloud Computing Bible, Barrie Sosinsky)

1.5 Αρχιτεκτονική συστημάτων Cloud

Ένα τυπικό σύστημα Cloud[3], αποτελείται όπως φαίνεται στο παρακάτω σχήμα (Σχήμα 5) από τα εξής επιμέρους στοιχεία:

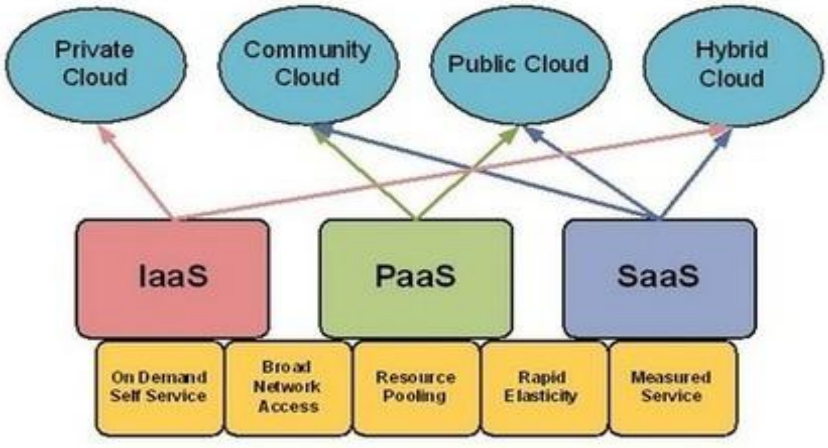
- Μια πλατφόρμα front-end, η οποία μπορεί να είναι ένας fat ή thin client ή μια κινητή συσκευή;
- Πλατφόρμες back-end, όπως εξυπηρετητές (servers) ή αποθηκευτικός χώρος (storage);
- Μια παράδοση (delivery) που βασίζεται σε Cloud;
- Ένα δίκτυο (Internet, Intranet, Intercloud).



Σχήμα 5: Αρχιτεκτονική συστημάτων cloud (πηγή: http://www.guru99.com/cloud-computing-for-beginners.html)

1.6 Μοντέλα ανάπτυξης

Υπάρχουν διάφορα μοντέλα ανάπτυξης ενός Cloud και το μοντέλο που επιλέγεται κάθε φορά, εξαρτάται από την ιδιοκτησία, το μέγεθος και την προσβασιμότητα αυτού.



Σχήμα 6: Τα μοντέλα ανάπτυξης του υπολογιστικού νέφους (πηγή: http://www.scoop.it/t/trends-in-cloud-computing)

α) **Ιδιωτικό νέφος (Private Cloud)**: Η υποδομή Cloud παρέχεται για αποκλειστική χρήση από έναν μοναδικό οργανισμό που αποτελείται από πολλαπλούς χρήστες (για παράδειγμα μονάδες επιχείρησης). Μπορεί να κατέχει, να διαχειρίζεται και να λειτουργεί το

υπολογιστικό νέφος, ένας οργανισμός, ένα τρίτο πρόσωπο ή ένας συνδυασμός αυτών, ενώ μπορεί να βρίσκεται εντός ή εκτός των εγκαταστάσεων τους.

β)**Νέφος κοινότητας (Community Cloud)**: Η υποδομή Cloud παρέχεται για αποκλειστική χρήση από μία συγκεκριμένη κοινότητα χρηστών, από οργανισμούς που έχουν κοινά ενδιαφέροντα (για παράδειγμα κοινή αποστολή, πολιτική απαιτήσεων ασφαλείας και ενδιαφέροντα συμβατότητας). Μπορεί να κατέχει, να διαχειρίζεται και να λειτουργεί το Cloud, ένας ή περισσότεροι από τους οργανισμούς της κοινότητας, ένα τρίτο πρόσωπο ή ένας συνδυασμός αυτών, ενώ μπορεί να βρίσκεται εντός ή εκτός των εγκαταστάσεων τους.

γ) **Δημόσιο νέφος (Public Cloud)**: Η υποδομή Cloud παρέχεται για δημόσια χρήση. Μπορεί να κατέχει, να διαχειρίζεται και να λειτουργεί το Cloud μία επιχείρηση, ένας ακαδημαϊκός ή κυβερνητικός οργανισμός ή ένας συνδυασμός αυτών. Βρίσκεται εντός των εγκαταστάσεων του παρόχου του Cloud.

δ) **Υβριδικό νέφος (Hybrid Cloud)**: Η υποδομή Cloud είναι μια σύνθεση δύο ή περισσοτέρων ξεχωριστών υποδομών νέφους (ιδιωτικό, κοινότητας ή δημόσιο), που παραμένουν μοναδικές οντότητες, αλλά είναι δεμένες (δεσμευμένες) μαζί μέσω πρότυπης ή ιδιοκτησιακής τεχνολογίας που επιτρέπει την φορητότητα δεδομένων και εφαρμογών.

1.7 ΠλεονεκτήματαCloud

Τα πέντε χαρακτηριστικά του Cloud που αναφέρθηκαν παραπάνω, αποτελούν ήδη κάποια από τα πλεονεκτήματά του. Έχει ωστόσο κι άλλα πλεονεκτήματα η χρήση αυτού[2], όπως:

- Χαμηλότερο κόστος: Το γεγονός ότι τα δίκτυα Cloud λειτουργούν με υψηλότερη αποτελεσματικότητα και υψηλότερο ποσοστό χρήσης, προκύπτουν συνήθως σημαντικές μειώσεις του κόστους.
- Ευκολία χρήσης: Αναλόγως με το είδος της προσφερόμενης υπηρεσίας, είναι πιθανό να μην απαιτούνται υλικό ή άδειες λογισμικού για να εκτελεστεί η υπηρεσία αυτή.
- Ποιότητα υπηρεσίας: Η ποιότητα της υπηρεσίας μπορεί να αποκτηθεί στο πλαίσιο σύμβασης με τον προμηθευτή.
- Αξιοπιστία: Η κλίμακα των δικτύων Cloud και η δυνατότητά τους να παρέχουν εξισορρόπηση υπερβάλλοντος φορτίου που μπορεί να προκύψει και αντιμετώπιση αστοχίας του συστήματος, τα καθιστά αξιόπιστα.
- Εξωτερική διαχείριση: Η ανάπτυξη ενός Cloud, επιτρέπει σε κάποιον άλλον την διαχείριση της υπολογιστικής υποδομής, ενώ ο πελάτης διαχειρίζεται την επιχείρησή του. Αυτό έχει ως αποτέλεσμα αξιοσημείωτες μειώσεις στο κόστος πρόσληψης υπαλλήλων τεχνολογίας πληροφοριών.
- Απλουστευμένη συντήρηση και αναβάθμιση: Επειδή το σύστημα είναι συγκεντρωτικό, είναι εύκολη η εφαρμογή διορθώσεων και ενημερώσεων. Αυτό σημαίνει ότι ο πελάτης έχει πάντα πρόσβαση στις πιο πρόσφατες εκδόσεις λογισμικού.

1.8 Μειονεκτήματα-ΠροβλήματαCloud

Παρά τα πολλά του πλεονεκτήματα, το Cloud δεν είναι πανάκεια και δεν είναι πάντα η καταλληλότερη επιλογή.

Τα σημαντικότερα προβλήματα που προκύπτουν από την χρήση του Cloud είναι η ευάλωτη ασφάλεια και η παραβίαση της ιδιωτικότητας. Δεδομένου ότι πολλοί διαφορετικοί χρήστες χρησιμοποιούν τους ίδιους φυσικούς πόρους, μπορούν εύκολα να αλλοιωθούν τα δεδομένα του κάθε χρήστη. Παράλληλα, δεν υπάρχει μεγάλη ασφάλεια, καθώς μια ενδεχόμενη παραβίαση του χώρου στον οποίο βρίσκονται οι φυσικές υποδομές του υπολογιστικού νέφους, μπορεί να οδηγήσει στην υποκλοπή δεδομένων και κατ'επέκταση στην παραβίαση της ιδιωτικότητας.

Επιπλέον, προκύπτουν και ζητήματα νομικής φύσεως, καθώς λόγω της κατανεμημένης φύσης των συστημάτων Cloud, είναι πρακτικά δύσκολος ο εντοπισμός της φυσικής τοποθεσίας, στην οποία βρίσκονται τα δεδομένα ενός χρήστη ή όπου γίνεται η εκτέλεση των εφαρμογών του. Αναλόγως με την χώρα στην οποία βρίσκεται η φυσική υποδομή ενός Cloud, μπορεί να ισχύει και διαφορετική νομοθεσία σε ό,τι αφορά την ιδιωτικότητα των δεδομένων. Αυτό έχει ως αποτέλεσμα, να μην γνωρίζει τελικά ο χρήστης, σε ποια νομοθεσία υπόκεινται τα δεδομένα και οι δραστηριότητές του και τελικά να υφίσταται παραβίαση της ιδιωτικότητάς του, χωρίς να το γνωρίζει.

Άλλα μειονεκτήματα του Cloud είναι η αδυναμία του να ικανοποιήσει συστήματα και εφαρμογές που απαιτούν τη μεταφορά μεγάλων ποσοτήτων δεδομένων, λόγω της έμφυτης αδυναμίας της συνδεσιμότητας WAN που απαιτεί το νέφος για να λειτουργήσει.

Τέλος, δεδομένου ότι μια εφαρμογή Cloud απευθύνεται σε πολλούς διαφορετικούς χρήστες (είτε πρόκειται για επιχειρήσεις, είτε για ιδιώτες), είναι δύσκολο να έχει μεγάλη προσαρμοστικότητα στις απαιτήσεις και επιθυμίες του εκάστοτε χρήστη. Αντίθετα, μια εφαρμογή που κατασκευάζεται εντός των εγκαταστάσεων μιας επιχείρησης μπορεί να είναι κατασκευασμένη με τρόπο προσωποποιημένο, που να καλύπτει όλες τις ανάγκες και επιθυμίες της εκάστοτε επιχείρησης (ή του εκάστοτε χρήστη).

Κεφάλαιο 2:Big Data

2.1 Εισαγωγή

Καθημερινά παράγονται χιλιάδες δεδομένα με μεγάλη ποικιλία περιεχομένου. Από δημοσιεύσεις σε ιστοσελίδες και φωτογραφίες σε μέσα κοινωνικής δικτύωσης, μέχρι επιστημονικές μετρήσεις κάθε είδους. Αυτή η τεράστια παραγωγή δεδομένων, καθώς και η ανάγκη επεξεργασίας αυτών, έχει επηρεάσει βαθιά και τις επιχειρήσεις, αφού τα παραδοσιακά συστήματα βάσεων δεδομένων, όπως οι σχεσιακές βάσεις δεδομένων αλλά και οι διάφορες συνηθισμένες τεχνικές διαχείρισης δεδομένων έχουν φτάσει στα όριά τους και αποτυγχάνουν να κλιμακωθούν στα Big Data.

Σχήμα 7: Παραγωγή Big Data με πολλούς τρόπους και σε πολλές μορφές (

Για να αντιμετωπιστούν λοιπόν οι προκλήσεις των Big Data, μία νέα γενιά τεχνολογιών αναδύθηκε[4]. Πολλές από αυτές τις τεχνολογίες έχουν ομαδοποιηθεί κάτω από τον όρο "NoSQL". Σε κάποιες περιπτώσεις, αυτές οι νέες τεχνολογίες είναι πιο περίπλοκες από τις παραδοσιακές βάσεις δεδομένων και σε άλλες περιπτώσεις είναι πιο απλές. Μπορούν να κλιμακωθούν σε πολύ μεγαλύτερο όγκο δεδομένων, αλλά για την αξιοποίησή τους, απαιτεί την χρήση διαφόρων νέων τεχνικών, αφού δεν υπάρχει μία που να καλύπτει όλες τις περιπτώσεις.

Πρωτοπόρος στην προσπάθεια ανάπτυξης νέων τεχνικών, για την επεξεργασία των Big Data, ήταν η Google με το προγραμματιστικό μοντέλο MapReduce και τα κατανεμημένα συστήματα αρχείων, ενώ ακολούθησαν η Amazon με τον αποθηκευτικό χώρο Dynamo και η κοινότητα ανοιχτού λογισμικού με εργαλεία όπως τα εξής: Hadoop, HBase, MongoDB, Cassandra, RabbitMQ, κτλ.

2.2 Ορισμός

Έχουν γίνει αρκετές προσπάθειες να δοθεί ένας ορισμός για τα Big data[5]. Δεν υπάρχει ωστόσο κάποιος που να περιγράφει με ακρίβεια τον όρο αυτό και για τον λόγο αυτό δίνονται στη συνέχεια τρεις πιθανοί ορισμοί.

Το αγγλικό λεξικό της Οξφόρδης ορίζει ως Big Data[6] "δεδομένα πολύ μεγάλου μεγέθους, συνήθως σε τέτοιο βαθμό, ώστε ο χειρισμός και η διαχείριση τους, να παρουσιάζει σημαντικές λογιστικές προκλήσεις".
Παράλληλα, η ηλεκτρονική εγκυκλοπαίδεια Wikipedia υποστηρίζει ότι[7] "τα Big Data είναι ένας ευρύς όρος για συλλογές δεδομένων, τόσο μεγάλες ή σύνθετες, που οι παραδοσιακές εφαρμογές επεξεργασίας δεδομένων είναι ανεπαρκείς".
Τέλος, η εταιρία McKinsey σε μεγάλη έρευνα σχετικά με τα Big Data το 2011, έδωσε τον εξής ορισμό[8]: "Συλλογές δεδομένων των οποίων το μέγεθος είναι πέρα από τη δυνατότητα των συνηθισμένων εργαλείων λογισμικού βάσεων δεδομένων, να το συλλέξουν, να το αποθηκεύσουν, να το διαχειριστούν και να το αναλύσουν".
Στην πραγματικότητα τα Big Data είναι ένας συνδυασμός όλων των παραπάνω. Πρόκειται για δεδομένα πολύ μεγάλου μεγέθους και μεγάλης ποικιλίας περιεχομένου, τα οποία παράγονται με μεγάλο ρυθμό, με αποτέλεσμα ο όγκος τους να αυξάνεται συνεχώς και η αποθήκευση, η επεξεργασία και η διαχείρισή τους, να γίνονται ολοένα και πιο δύσκολες.

2.3 Χαρακτηριστικά των Big Data

Τα Big Data μπορούν να περιγραφούν από τα λεγόμενα 7Vs[9]: Όγκος (Volume), Ταχύτητα (Velocity), Ποικιλία (Variety), Μεταβλητότητα (Variability), Αξιοπιστία (Veracity), Απεικόνιση (Visualisation) και Αξία (Value).

Όγκος (Volume): Η κλίμακα αποτελεί σίγουρα έναν από τους λόγους που κάνουν τα Big Data μεγάλα. Η επανάσταση του Διαδικτύου και των φορητών συσκευών, φέρνει μαζί της και έναν μεγάλο όγκο δεδομένων από κοινωνικά δίκτυα και αισθητήρες συσκευών, ενώ η έκρηξη του ηλεκτρονικού εμπορίου σημαίνει ότι κάθε βιομηχανία κατακλύζεται από πληροφορίες, οι οποίες είναι πολύτιμες, αν γνωρίζει κανείς πώς να τις αξιοποιήσει.

Ταχύτητα (Velocity): Αναφέρεται στην αυξανόμενη ταχύτητα με την οποία τα δεδομένα δημιουργούνται καθώς και στην αυξανόμενη ταχύτητα με την οποία μπορούν να προσπελαστούν, αποθηκευτούν και αναλυθούν από σχεσιακές βάσεις δεδομένων. Οι πιθανότητες επεξεργασίας δεδομένων σε πραγματικό χρόνο, είναι ένας τομέας ιδιαίτερου ενδιαφέροντος, που επιτρέπει στις εταιρίες να κάνουν πράγματα όπως οι προσωποποιημένες διαφημίσεις στις ιστοσελίδες που επισκέπτεται κανείς, βασισμένες στο πρόσφατο ιστορικό αναζητήσεων και αγορών.

Ποικιλία (Variety): Το μεγαλύτερο ποσοστό των δεδομένων που παράγονται είναι αδόμητα και έχουν διαφορετικά σχήματα και μορφές, από γεω-χωρικά δεδομένα, μέχρι αναρτήσεις σε κοινωνικά δίκτυα, οι οποίες μπορούν να αναλυθούν για τα συναισθήματα και το περιεχόμενό τους και οπτικά δεδομένα όπως φωτογραφίες ή βίντεο.

Μεταβλητότητα (Variability): Αναφέρεται σε δεδομένα που συνεχώς αλλάζουν. Χαρακτηριστική περίπτωση που εμφανίζεται μεγάλη μεταβλητότητα αποτελούν τα δεδομένα που προκύπτουν από επεξεργασία γλώσσας. Αυτό συμβαίνει, διότι οι λέξεις ανάλογα με τα συμφραζόμενα μπορούν να αλλάζουν νόημα και να εκφράζουν διαφορετικό νόημα.

Αξιοπιστία (Veracity): Αυτό το χαρακτηριστικό, αναφέρεται στην ποιότητα των δεδομένων που παράγονται και αποθηκεύονται, η οποία μπορεί να ποικίλει σημαντικά, επηρεάζοντας την ακρίβεια τους και την αξιοπιστία των συμπερασμάτων της ανάλυσής τους. Η φύση των Big Data είναι ακατάστατη και γεμάτη θόρυβο και χρειάζεται πολύ δουλειά για να παραχθεί μια ακριβής συλλογή δεδομένων, πριν αρχίσει η ανάλυσή τους.

Απεικόνιση (Visualisation): Ένα άλλο χαρακτηριστικό των Big Data, είναι η δυσκολία εύρεσης τρόπου απεικόνισης των δεδομένων, με τρόπο προσιτό και κατανοητό. Οι διάφορες απεικονίσεις μπορεί να περιέχουν δεκάδες μεταβλητές και παραμέτρους, και η εύρεση τρόπου παρουσίασης των πληροφοριών αυτών, που να κάνει σαφή τα ευρήματα, είναι μία από τις προκλήσεις των Big Data.

Αξία (Value): Η εν δυνάμει αξία των Big Data, η οποία μπορεί να προκύψει από την ανάλυσή τους, είναι τεράστια. Μπορούν ωστόσο ταυτόχρονα, είτε να μειώσουν περιττά έξοδα, είτε να επιβαρύνουν με επιπρόσθετα κόστη. Για παράδειγμα, η αξιοποίηση των τεχνικών των Big Data για την ψηφιοποίηση γραφειοκρατικών διαδικασιών σε οποιαδήποτε δημόσια υπηρεσία, θα μπορούσε να μειώσει πολύ τα έξοδα λειτουργίας αυτής της υπηρεσίας. Ωστόσο, η ύπαρξη τεράστιου αχρείαστου όγκου Big Data που μένει αναξιοποίητος κοστίζει πολλά χρήματα κάθε χρόνο στις επιχειρήσεις.

2.4 Ιδιότητες ενός συστήματος Big Data

Σύμφωνα με τα χαρακτηριστικά που αναφέρθηκαν παραπάνω για τα Big Data, ένα σύστημα για να μπορεί να συλλέγει, να επεξεργάζεται, να αποθηκεύει και να αξιοποιεί Big Data, πρέπει να έχει τις εξής ιδιότητες[4]: 1) Ανθεκτικότητα και ανοχή σε σφάλματα, 2) Διάβασμα και Ανανέωση με μικρή καθυστέρηση, 3) Κλιμάκωση, 4) Γενίκευση, 5) Επεκτασιμότητα, 6) Εξειδικευμένα ερωτήματα (ad hoc queries), 7) Ελάχιστη συντήρηση και 8) Εύκολη αποσφαλμάτωση (debuggability).

*1) **Ανθεκτικότητα και ανοχή σε σφάλματα***

Μέρος της δημιουργίας ενός ανθεκτικού και αξιόπιστου συστήματος Big Data, είναι η αποφυγή περιπλοκών όπως η δυσκολία επίτευξης συνοχής σε κατανεμημένες βάσεις δεδομένων, τα διπλότυπα δεδομένων, ο συγχρονισμός, οι εξυπηρετητές (servers) που "πέφτουν" και άλλα. Ωστόσο, η ανθεκτικότητα του συστήματος, αφορά και μια άλλη περιπλοκή που συχνά παραβλέπεται και η οποία είναι το ανθρώπινο λάθος. Σε οποιοδήποτε πληροφοριακό σύστημα, είναι αναπόφευκτο το να γίνει κάποιο ανθρώπινο λάθος κάποια στιγμή, όπως για παράδειγμα το να εφαρμοστεί ένας λανθασμένος κώδικας ο οποίος αλλοιώνει τις τιμές μιας βάσης δεδομένων. Το χτίσιμο ωστόσο ενός σταθερού πυρήνα σε ένα σύστημα Big Data, συνδυαστικά με έναν μηχανισμό επαναφοράς από σφάλμα, συμβάλλουν στην ανθεκτικότητα του συστήματος σε ανθρώπινα λάθη.

*2) **Διάβασμα και ανανέωση με μικρή καθυστέρηση***

Η πλειονότητα των εφαρμογών που αναπτύσσονται, απαιτεί το διάβασμα (read) να ικανοποιείται με πολύ μικρή καθυστέρηση, συνήθως μεταξύ μερικών χιλιοστών του δευτερολέπτου και μερικών εκατοντάδων χιλιοστών του δευτερολέπτου (milliseconds).

Αντίθετα, οι απαιτήσεις σε χρονική καθυστέρηση, σε ότι αφορά την ανανέωση (update), ποικίλουν από εφαρμογή σε εφαρμογή. Κάποιες εφαρμογές απαιτούν οι ανανεώσεις να διαδίδονται αμέσως, ενώ σε κάποιες άλλες μια καθυστέρηση μερικών ωρών είναι αποδεκτή. Ανεξάρτητα από την εκάστοτε εφαρμογή ωστόσο, είναι απαραίτητη η επίτευξη χαμηλής καθυστέρησης διαβάσματος και ανανέωσης σε ένα σύστημα Big Data, όταν αυτή απαιτείται, και κυρίως χωρίς να επηρεάζεται η ανθεκτικότητα του συστήματος.

*3) **Κλιμάκωση***
Η κλιμάκωση είναι η δυνατότητα διατήρησης της επίδοσης του συστήματος, σε περίπτωση αυξημένου όγκου δεδομένων ή πλεονάζοντος φορτίου, μέσω της προσθήκης υπολογιστικών πόρων στο σύστημα. Αυτή η ιδιότητα των συστημάτων Big Data, είναι και ένα από τα αντικείμενα που θα εξετασθούν στα πλαίσια αυτής της διπλωματικής εργασίας.

*4) **Γενίκευση***
Ένα γενικευμένο σύστημα, μπορεί να υποστηρίξει μεγάλο εύρος εφαρμογών.

*5) **Επεκτασιμότητα***
Η ιδιότητα αυτή, επιτρέπει την προσθήκη λειτουργιών και χαρακτηριστικών στο σύστημα, με ελάχιστο κόστος ανάπτυξης. Συχνά, η προσθήκη ενός καινούριου χαρακτηριστικού, απαιτεί την μετατροπή δεδομένων από μια παλιά σε μία νέα μορφή. Όταν ένα σύστημα είναι επεκτάσιμο, μετατροπές τέτοιου είδους είναι εύκολο να γίνουν σε μεγάλη κλίμακα.

*6) **Εξειδικευμένα ερωτήματα (ad hoc queries)***
Η υποβολή εξειδικευμένων ερωτημάτων είναι πολύ σημαντική, αφού σχεδόν κάθε συλλογή δεδομένων κρύβει απρόσμενες τιμές. Η δυνατότητα εξειδικευμένης έρευνας μιας συλλογής δεδομένων δίνει την ευκαιρία για βελτιστοποίηση συστημάτων Big Data, σε επίπεδο επιχειρήσεων καθώς και την δυνατότητα ανάπτυξης νέων εφαρμογών.

*7) **Ελάχιστη συντήρηση***
Η συντήρηση αναφέρεται στις εργασίες που απαιτούνται για την διατήρηση της ομαλής λειτουργίας ενός συστήματος. Αυτές περιλαμβάνουν την πρόβλεψη της ανάγκης πρόσθετων μηχανημάτων, την διατήρηση των διαδικασιών σε λειτουργία και την αντιμετώπιση οποιουδήποτε σφάλματος μπορεί να εμφανιστεί.
Σημαντικός παράγοντας που επηρεάζει την συντήρηση ενός συστήματος, είναι η επιλογή των δομικών στοιχείων. Οι κατανεμημένες βάσεις δεδομένων τείνουν να έχουν πολύ πολύπλοκους εσωτερικούς μηχανισμούς. Όμως όσο πιο πολύπλοκο είναι ένα σύστημα, τόσο πιο πιθανό είναι να εμφανισθεί κάποιο σφάλμα και τόσο πιο βαθιά κατανόηση του συστήματος απαιτείται για τον εντοπισμό και την αντιμετώπισή του.
Η πολυπλοκότητα αυτή λοιπόν, μπορεί να καταπολεμηθεί, βασίζοντας το σύστημα σε απλούς αλγορίθμους και σε απλά δομικά στοιχεία, μεταφέροντας την πολυπλοκότητα του συστήματος μακριά από τα στοιχεία του πυρήνα, σε στοιχεία που βρίσκονται στα εξωτερικά στρώματα του συστήματος και των οποίων τα προϊόντα, απομακρύνονται μετά από κάποιες ώρες. Συνεπώς, μικρή πολυπλοκότητα του συστήματος, οδηγεί και σε ελαχιστοποίηση των απαιτήσεων για την συντήρησή του.

*8) **Αποσφαλμάτωση (debuggability)***

Ένα σύστημα Big Data πρέπει να παρέχει τις απαραίτητες πληροφορίες για την αποσφαλμάτωση του συστήματος όταν κάτι δεν λειτουργεί σωστά. Το κλειδί είναι η δυνατότητα εντοπισμού του λόγου για τον οποίο κάθε μεταβλητή του συστήματος, έχει αυτήν την τιμή. Η αποσφαλμάτωση μπορεί να επιτευχθεί σε ένα σύστημα Big Data, μέσω αλγορίθμων επανάληψης υπολογισμών, όπου αυτό είναι εφικτό.

2.5 Επεξεργασία Big Data - MapReduce

Τα χαρακτηριστικά των Big Data καθώς και οι ιδιότητες που πρέπει να έχουν τα συστήματα τα οποία τα διαχειρίζονται, οδήγησαν στην ανάγκη για ανάπτυξη καινούριων τεχνικών επεξεργασίας δεδομένων, αφού οι παραδοσιακές τεχνικές αποτυγχάνουν. Το 2004, δύο ερευνητές της Google δημοσίευσαν μια εργασία, η οποία εισήγαγε ένα καινούριο προγραμματιστικό μοντέλο επεξεργασίας δεδομένων, το οποίο χρησιμοποιείται έκτοτε ευρέως για την επεξεργασία Big Data.

Αυτό το προγραμματιστικό μοντέλο, ονομάζεται MapReduce και αφορά την επεξεργασία και την παραγωγή μεγάλων συλλογών δεδομένων, με τη χρήση ενός παράλληλου, κατανεμημένου αλγορίθμου καθώς και ενός κατανεμημένου cluster υπολογιστών. Το μοντέλο αυτό είναι εμπνευσμένο από δύο πολύ συνηθισμένες συναρτήσεις, που χρησιμοποιούνται στον συναρτησιακό προγραμματισμό και αναπτύχθηκε αρχικά από την Google[10], για την κατάταξη ιστοσελίδων σε πίνακα, αντικαθιστώντας τους προηγούμενους ανάλογους αλγορίθμους. Το MapReduce λειτουργεί χωρίζοντας την διαδικασία σε δύο φάσεις: την φάση map και την φάση reduce. Κάθε φάση έχει ζευγάρια κλειδιών-τιμών ως είσοδο και ως έξοδο, το είδος των οποίων μπορεί να καθοριστεί από τον προγραμματιστή. Ο προγραμματιστής προσδιορίζει επίσης, δύο λειτουργίες/συναρτήσεις:

- την Map, η οποία διαμοιράζει εργασίες σε διαφορετικούς κόμβους του κατανεμημένου cluster και
- την Reduce, η οποία συνδυάζει τις εργασίες και τα αποτελέσματα αυτών σε μία μοναδική τιμή.

Η πραγματική συμβολή ωστόσο του MapReduce, δεν είναι οι συναρτήσεις map και reduce, αλλά η κλιμάκωση (μέχρι και σε clusters με χιλιάδες υπολογιστές) και η ανέχεια σε σφάλματα.

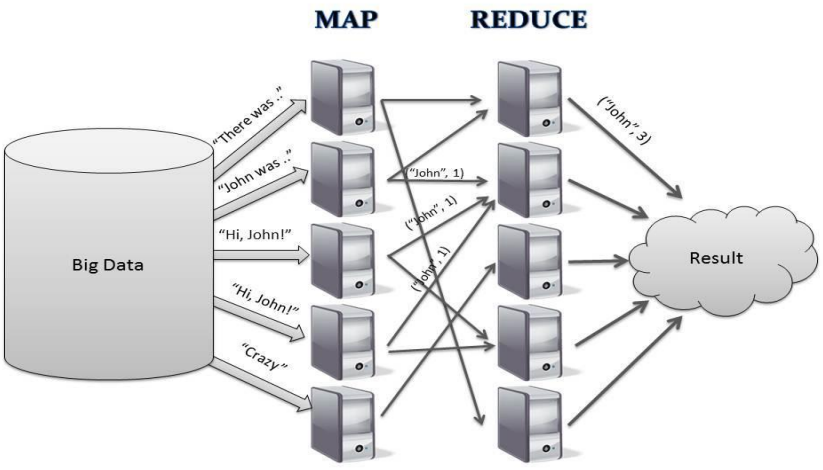
Για να γίνει πιο κατανοητός ο τρόπος χρήσης του MapReduce, δίνεται στη συνέχεια ένα απλό παράδειγμα χρήσης του[11]. Έστω ότι υπάρχουν πέντε αρχεία, καθένα από τα οποία, περιλαμβάνει δύο στήλες: μία με διάφορες πόλεις του κόσμου και μία με θερμοκρασίες. Έχουμε δηλαδή ζεύγη πόλης-θερμοκρασίας (κλειδιού-τιμής), που καταγράφηκαν σε διάφορες μέρες. Το κλειδί είναι η πόλη και η τιμή είναι η θερμοκρασία.

Έστω ότι ζητείται η μέγιστη θερμοκρασία για κάθε πόλη, λαμβάνοντας τις μετρήσεις και των πέντε αρχείων, όπου σε κάθε αρχείο μπορεί για μία πόλη να υπάρχουν περισσότερες από μία μετρήσεις θερμοκρασίας. Χρησιμοποιώντας το MapReduce, μπορούμε να χωρίσουμε την εργασία εύρεσης της μέγιστης θερμοκρασίας για κάθε πόλη, σε πέντε υπο-εργασίες (mappers), όπου κάθε μία από αυτές, βρίσκει την μέγιστη θερμοκρασία κάθε πόλης στο συγκεκριμένο αρχείο.

Τορόντο	*20*	*Παρίσι*	*20*	*Αθήνα*	*22*	*Παρίσι*	*16*	*Νέα Υόρκη*	*15*
Αθήνα	*25*	*Παρίσι*	*16*	*Ρώμη*	*26*	*Παρίσι*	*15*	*Ρώμη*	*18*
Ρώμη	*30*	*Αθήνα*	*32*	*Τορόντο*	*4*	*Ρώμη*	*35*	*Τορόντο*	*17*
Τορόντο	*10*	*Νέα Υόρκη*	*18*	*Νέα Υόρκη*	*14*	*Τορόντο*	*22*	*Τορόντο*	*7*
Ρώμη	*33*	*Ρώμη*	*29*	*Νέα Υόρκη*	*16*	*Αθήνα*	*26*	*Παρίσι*	*5*
Νέα Υόρκη	*22*	*Τορόντο*	*11*	*Ρώμη*	*27*	*Ρώμη*	*34*	*Παρίσι*	*12*
Αθήνα	*29*	*Τορόντο*	*5*	*Αθήνα*	*28*	*Νέα Υόρκη*	*20*	*Αθήνα*	*10*
Παρίσι	*18*	*Ρώμη*	*34*	*Παρίσι*	*24*	*Αθήνα*	*15*	*Τορόντο*	*8*

Σχήμα 8 :Τα πέντε αρχεία με τις μετρήσεις των θερμοκρασιών των πόλεων

Με βάση τον παραπάνω πίνακα και έπειτα από την εκτέλεση κάθε υπο-εργασίας προκύπτουν τα εξής ζεύγη:
(Τορόντο,20) (Αθήνα,29) (Ρώμη, 33) (Νέα Υόρκη,22) (Παρίσι, 18) (Τορόντο,11) (Αθήνα,32) (Ρώμη, 34) (Νέα Υόρκη,18) (Παρίσι, 20) (Τορόντο, 4) (Αθήνα,28) (Ρώμη, 27) (Νέα Υόρκη,16) (Παρίσι, 24) (Τορόντο,22) (Αθήνα,26) (Ρώμη, 35) (Νέα Υόρκη,20) (Παρίσι, 16) (Τορόντο,17) (Αθήνα,10) (Ρώμη, 18) (Νέα Υόρκη,15) (Παρίσι, 12).
Στη συνέχεια, μια άλλη υπο-εργασία(reduce) αναλαμβάνει να βρει την μέγιστη θερμοκρασία για κάθε πόλη, ανάμεσα στα παραπάνω αποτελέσματα των πέντε υπο-εργασιών. Με αυτόν τον τρόπο, προκύπτει ένα μοναδικό ζευγάρι-λύση για κάθε πόλη : (Τορόντο,22) (Αθήνα,32) (Ρώμη,35) (Νέα Υόρκη,22) (Παρίσι,24).
Ένα άλλο παράδειγμα, όπου χρησιμοποιείται συνεχώς το MapReduce είναι η μέτρηση της συχνότητας εμφάνισης μιας λέξης (για παράδειγμα John), μέσα σε ένα κείμενο. Στην περίπτωση αυτή, το κείμενο χωρίζεται σε μικρότερα τμήματα, σε καθένα από τα οποία υπολογίζεται η συχνότητα εμφάνισης της λέξης John και στη συνέχεια συνδυάζοντας τα επιμέρους αποτελέσματα, υπολογίζεται η συνολική συχνότητα εμφάνισης αυτής της λέξης σε ολόκληρο το κείμενο, όπως φαίνεται και στο παρακάτω σχήμα (Σχήμα 9).



Σχήμα 9: Παράδειγμα χρήσης του MapReduce για την αναζήτηση της συχνότητας της λέξης "John" μέσα σε κάποιο κείμενο.

Εκτός από το MapReduce, υπάρχουν και άλλα εργαλεία που χρησιμοποιούνται για την επεξεργασία Big Data. Τα πιο σημαντικά από αυτά είναι το Apache Hadoop, το οποίο είναι ανοιχτό λογισμικό και έχει βασιστεί στο MapReduce και στο Hadoop Distributed File System (HDFS), οι βάσεις δεδομένων NoSQL, το Hive, το PIG κτλ.

2.6 Τομείς που βρίσκουν εφαρμογή τα Big Data

Όπως αναφέρθηκε παραπάνω, τα Big Data συναντώνται σε πολλούς διαφορετικούς τομείς και απαιτούν την δημιουργία και χρήση συστημάτων ικανών να τα διαχειριστούν.

Ένας σημαντικός τομέας στον οποίο έχουν εφαρμογή τα Big Data[7], είναι αυτός της υγείας. Με τη χρήση των Big Data, μπορεί για παράδειγμα να δημιουργηθεί ένας ιατρικός φάκελος για κάθε ασθενή, ώστε να έχει όσο το δυνατόν πιο προσωποποιημένη φαρμακευτική αγωγή και γενικότερα περίθαλψη.

Άλλος σημαντικός τομέας στον οποίο χρησιμοποιούνται Big Data είναι αυτός του μάρκετινγκ και των διαφημιστικών εταιρειών, όπου μέσω της ανάλυσης των δεδομένων των κοινωνικών δικτύων, μελετάται η ανταπόκριση των ανθρώπων σε διαφημιστικές καμπάνιες, προσφορές και άλλα διαφημιστικά μέσα. Τα Big Data των κοινωνικών δικτύων χρησιμοποιούνται επίσης από εταιρείες καταναλωτικών προϊόντων, για την μελέτη των προτιμήσεων και της καταναλωτικής συμπεριφοράς των ανθρώπων.

Μεγάλο όγκο δεδομένων, διαχειρίζονται επίσης καθημερινά, οι δημόσιες υπηρεσίες ενός κράτους, όπως για παράδειγμα η εφορία, τα ΚΕΠ, τα ταμεία ασφαλίσεως κτλ.

Σε ό,τι αφορά τον τομέα της τεχνολογίας, πέρα από τις μηχανές αναζήτησης, επεξεργασία Big Data απαιτείται και στις διάφορες εφαρμογές του Internet of Things, το οποίο αποτελεί το δίκτυο διαφόρων φυσικών αντικειμένων, όπως συσκευές, οχήματα, κτίρια, κτλ, στα οποία υπάρχουν ενσωματωμένα ηλεκτρονικά κυκλώματα, λογισμικό, αισθητήρες και συνδεσιμότητα σε δίκτυο, τα οποία δίνουν στα αντικείμενα αυτά, τη δυνατότητα χειρισμού τους από απόσταση, καθώς και τη δυνατότητα συλλογής και ανταλλαγής δεδομένων. Μέσω αυτού του δικτύου αντικειμένων λοιπόν, παράγονται πολλά και διαφορετικά δεδομένα προς επεξεργασία.

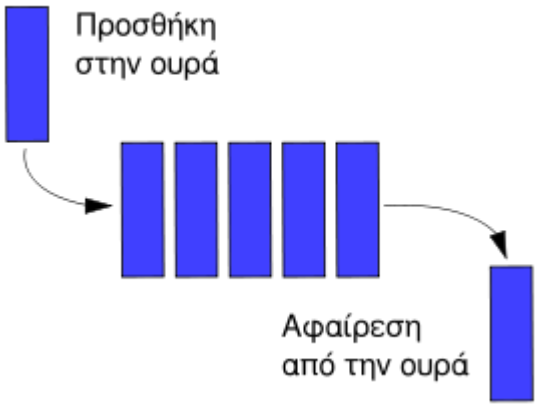
Τα Big Data αξιοποιούνται τέλος, σε επιστημονικά πειράματα και μελέτες. Ένα χαρακτηριστικό παράδειγμα επιστημονικής μελέτης Big Data, αποτελεί η επεξεργασία των δεδομένων που παράγονται από τους εκατομμύρια αισθητήρες του μεγάλου επιταχυντή αδρονίων, που βρίσκεται στο Cern της Γενεύης, τα οποία ξεπερνούν τα 500 exabytes την ημέρα και τα οποία είναι 200 φορές ο όγκος δεδομένων που παράγεται από όλες τις υπόλοιπες πηγές του κόσμου μαζί. Μεγάλος όγκος δεδομένων παράγεται επίσης μέσω αστρονομικών, βιολογικών και κάθε άλλου είδους επιστημονικών μελετών.

Κεφάλαιο 3:Ουρές Μηνυμάτων (Message Queues)

3.1 Ορισμός ουράς

Για να γίνει κατανοητός ο ορισμός της ουράς μηνυμάτων, πρέπει πρώτα να επεξηγηθεί η έννοια της ουράς, ως δομής δεδομένων[12].

Η ουρά είναι μια αφηρημένη δομή δεδομένων με ευρεία χρήση στα υπολογιστικά συστήματα, η οποία λειτουργεί σύμφωνα με την αρχή FIFO (First In First Out - Πρώτο Μέσα Πρώτο Έξω). Λειτουργεί επομένως, όπως και μια ουρά ανθρώπων που περιμένουν σε ένα ταμείο για παράδειγμα. Κάθε φορά εξυπηρετείται αυτός που βρίσκεται στην αρχή της ουράς, ενώ οποιοσδήποτε άλλος θέλει να προστεθεί στην ουρά, στέκεται στο τέλος αυτής και θα εξυπηρετηθεί, αφού εξυπηρετηθούν όλοι όσοι βρίσκονται μπροστά του. Υποστηρίζει δύο λειτουργίες : προσθήκη στην ουρά (enqueue), η οποία γίνεται πάντα στο τέλος της ουράς και αφαίρεση από την ουρά (dequeue), η οποία γίνεται πάντα από την αρχή της ουράς (Σχήμα 10).



Σχήμα 10: Σχηματική απεικόνιση της δομής της ουράς καθώς και των δύο βασικών λειτουργιών της: προσθήκη και αφαίρεση(πηγή: Wikipedia "Queue")

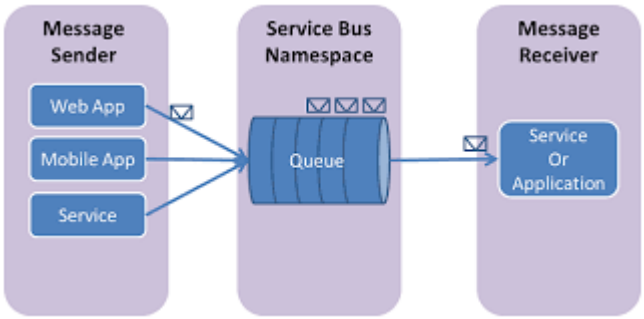
Υπάρχουν διάφορα είδη ουρών, καθένα από τα οποία είναι διαφορετικό και δημιουργήθηκε για την επίλυση συγκεκριμένων προβλημάτων. Ένα από αυτά τα είδη είναι και οι ουρές μηνυμάτων (Message Queues).

3.2 Ορισμός ουράς μηνυμάτων (message queue)

Οι ουρές μηνυμάτων χρησιμοποιούνται για την ανταλλαγή δεδομένων ανάμεσα σε διαφορετικές εφαρμογές ή σε διαφορετικά συστήματα ή σε διαφορετικές υπηρεσίες, μέσω ασύγχρονων μηνυμάτων[13]. Μέσω των ουρών αυτών, οι εφαρμογές είναι αποσυνδεδεμένες και οι αποδέκτες και οι παραλήπτες υπάρχουν, αγνοώντας ο ένας την ύπαρξη του άλλου. Είναι επομένως, υπεύθυνη η ουρά μηνυμάτων για την μεταφορά των μηνυμάτων ανάμεσα στις εφαρμογές, ενώ επιτρέπει την αποθήκευση δεδομένων που θα προσπελαστούν αργότερα. Τα μηνύματα που διαχειρίζονται, μπορούν να έχουν μορφή απλού κειμένου (.txt), μορφή xml ή οποιαδήποτε άλλη μορφή.

Κάποιες από τις συνηθισμένες περιπτώσεις χρήσης των ουρών μηνυμάτων είναι οι εξής:

- Αποστολή/Λήψη δεδομένων από τρίτες διεπαφές προγραμματισμού εφαρμογών (APIs)
- Ασύγχρονη επικοινωνία ανάμεσα σε ποικίλες εφαρμογές
- Αποστολή e-mail
- "Ανέβασμα" εγγράφων
- "Τρέξιμο" εντατικών διαδικασιών



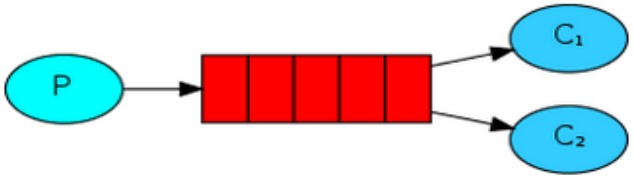
Σχήμα 11: Σχηματική απεικόνιση της λειτουργίας μιας ουράς μηνυμάτων (πηγή: https://azure.microsoft.com/en-us/documentation/articles/service-bus-dotnet-how-to-use-queues/)

3.3 Τρόποι χρήσης των ουρών μηνυμάτων

Οι ουρές μηνυμάτων υποστηρίζουν μπορούν να χρησιμοποιηθούν με αρκετούς διαφορετικούς τρόπους[14]. Ο πιο απλός από αυτούς είναι να χρησιμοποιηθεί ως μια απλή ουρά η οποία δέχεται και προωθεί μηνύματα, όπως ένα γραμματοκιβώτιο (Σχήμα 12).

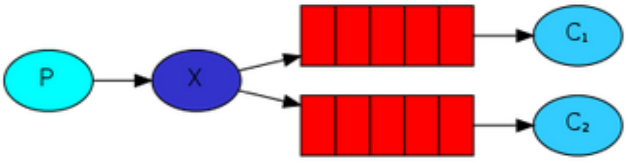
Σχήμα 12: Απλή ουρά μηνυμάτων(πηγή: RabbitMQ tutorials)

Ένας άλλος τρόπος χρήσης, επιτρέπει την κατανομή πολλών χρονοβόρων εργασιών σε πολλούς διαφορετικούς "εργάτες" (workers), το οποίο εξυπηρετεί την αποφυγή της άμεσης εκτέλεσης και της αναμονής λήξης μιας εντατικής εργασίας (Σχήμα 13).



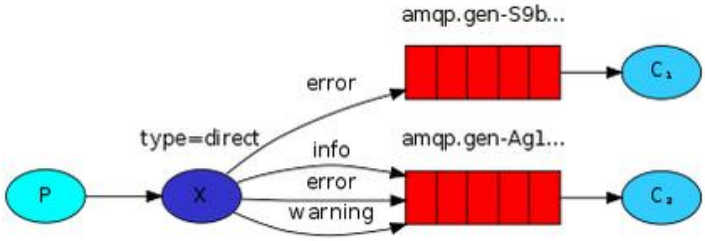
Σχήμα 13: Χρήση ουράς για την κατανομή μιας εργασίας (πηγή: RabbitMQ tutorials)

Ακόμη, υπάρχει η δυνατότητα χρήσης της ουράς για την αποστολή του ίδιου μηνύματος σε πολλούς παραλήπτες ταυτόχρονα. Αυτό το μοντέλο αρχιτεκτονικής της ουράς είναι γνωστό ως Publish/Subscribe (Σχήμα χχ).



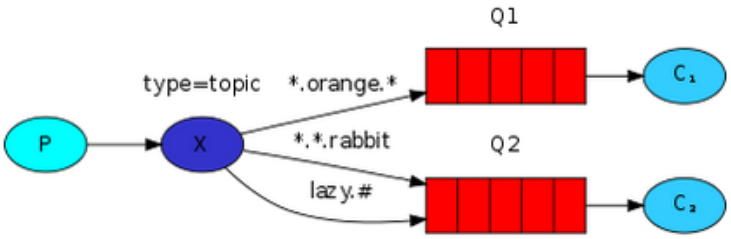
Σχήμα 14: Χρήση ουράς για την αποστολή του ίδιου μηνύματος σε πολλούς παραλήπτες (πηγή: RabbitMQ tutorials)

Υπάρχει και μία επέκταση αυτού του μοντέλου, κατά την οποία ένα μήνυμα μπορεί να σταλθεί ταυτόχρονα σε πολλούς παραλήπτες, αλλά ο κάθε παραλήπτης δεν έχει δικαίωμα να λάβει όλα τα μηνύματα, παρά μόνο ένα υποσύνολο αυτών (Σχήμα 15).



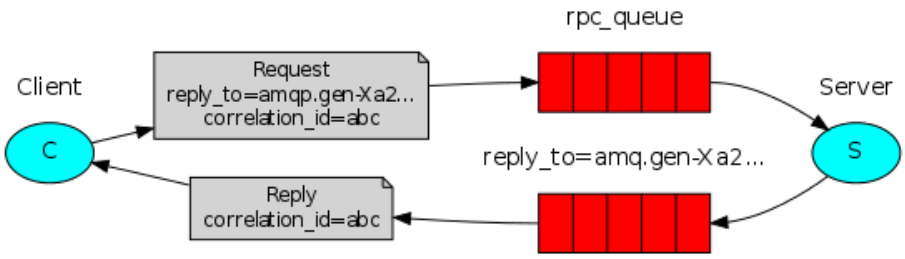
Σχήμα 15: Χρήση ουράς για την αποστολή του ίδιου μηνύματος σε πολλούς παραλήπτες, με διαφορετικά δικαιώματα λήψης ο καθένας (πηγή: RabbitMQ tutorials)

Πολύ σημαντικός τρόπος χρήσης της ουράς μηνυμάτων είναι επίσης η χρήση της ως "θέμα" (topic), κατά την οποία έχει την δυνατότητα να αποφασίζει εκείνη, με βάση κάποια κριτήρια ή κάποια "κλειδιά", σε ποιον/ποιους παραλήπτες θα στείλει το κάθε μήνυμα (Σχήμα 16).



Σχήμα 16: Χρήση ουράς η οποία αποφασίζει με βάση κάποια κλειδιά σε ποιους παραλήπτες θα στείλει κάθε μήνυμα (πηγή: RabbitMQ tutorials)

Τέλος, υπάρχει και η περίπτωση χρήσης της ουράς σε κατανεμημένα συστήματα, για την απομονωμένη κλήση μιας διαδικασίας (Remote Procedure Call-RPC), κατά την οποία ένα πρόγραμμα εκτελεί μια διαδικασία σε έναν άλλο υπολογιστή του ιδίου δικτύου (Σχήμα χχ).



Σχήμα 17: Χρήση ουράς για απομονωμένη κλήση διαδικασίας (πηγή: RabbitMQ tutorials)

3.4 Χαρακτηριστικά των ουρών μηνυμάτων - Πλεονεκτήματα και Μειονεκτήματα

Οι ουρές μηνυμάτων διαθέτουν κάποια χαρακτηριστικά τα οποία αποτελούν ταυτόχρονα και πλεονεκτήματα αυτών έναντι άλλων μορφών επικοινωνίας ανάμεσα σε διαδικασίες ή εφαρμογές[15]. Τα πλεονεκτήματα αυτά είναι τα εξής:

α)**Αποσύνδεση (Decoupling):** Στην αρχή της διαδικασίας ανάπτυξης μιας εφαρμογής είναι δύσκολο να προβλεφτούν οι μελλοντικές ανάγκες. Με την εισαγωγή μίας ουράς μηνυμάτων ανάμεσα σε δύο διαδικασίες, δημιουργείται ουσιαστικά, μία διεπιφάνεια, με την οποία επικοινωνούν και οι δύο. Με την εξασφάλιση επομένως, της ανταπόκρισης και των δύο διαδικασιών στις ίδιες απαιτήσεις διεπιφάνειας, αυτές ανεξαρτητοποιούνται η μία από την άλλη και καθίσταται δυνατή η ανεξάρτητη επέκταση και τροποποίηση των διαδικασιών αυτών.

β)**Πλεονασμός (Redudancy):** Μερικές φορές οι διαδικασίες μπορεί να αποτύχουν και να σταματήσουν να λειτουργούν κατά τη διάρκεια της επεξεργασίας δεδομένων. Αν τα δεδομένα αυτά δεν είναι "επίμονα" (persistent), τότε χάνονται για πάντα. Οι ουρές μηνυμάτων μπορούν να μετριάσουν το πρόβλημα αυτό, διατηρώντας τα δεδομένα μέχρι να έχει ολοκληρωθεί η επεξεργασία αυτών. Το παράδειγμα τοποθέτησε-πάρε-διάγραψε (put-get-delete), που πολλές ουρές χρησιμοποιούν, απαιτεί από μια διαδικασία να υποδείξει ότι έχει τελειώσει την επεξεργασία ενός μηνύματος, πριν αυτό αφαιρεθεί από την ουρά, διασφαλίζοντας την διατήρηση των δεδομένων, όσο αυτά είναι απαραίτητα.

γ)**Επεκτασιμότητα (Scalability**): Δεδομένου ότι οι ουρές μηνυμάτων ανεξαρτητοποιούν τις διαδικασίες μεταξύ τους, μπορεί εύκολα να αυξηθεί ο ρυθμός με τον οποίο προστίθενται μηνύματα στην ουρά ή τίθενται υπό επεξεργασία, απλά με την προσθήκη μίας ακόμη διαδικασίας. Δεν χρειάζεται να γίνει καμία αλλαγή στον κώδικα, ούτε να διαμορφωθούν οι ρυθμίσεις.

δ)**Ελαστικότητα (Elasticity):** Μια εφαρμογή είναι απαραίτητο να μπορεί να λειτουργήσει και με αυξημένο φορτίο δεδομένων. Ωστόσο η ύπαρξη επιπλέον πόρων σε αναμονή, προκειμένου να αντιμετωπιστούν τέτοια αυξημένα φορτία, είναι σπατάλη, η οποία μπορεί να αποφευχθεί με την χρήση ουρών μηνυμάτων. Οι ουρές θα βοηθήσουν τα στοιχεία που πολιορκούνται από πολλαπλές αιτήσεις, να ανταπεξέλθουν στην αυξημένη ζήτηση και να μην υπερφορτωθούν και τελικά αποτύχουν.

ε)**Ευελιξία:** Όταν μέρος της αρχιτεκτονικής αποτυγχάνει, δεν χρειάζεται να αποτυγχάνει ολόκληρο το σύστημα. Σε περίπτωση αποτυχίας κάποιας διαδικασίας που επικοινωνεί με την ουρά, μπορεί να συνεχιστεί η προσθήκη μηνυμάτων στην ουρά, τα οποία θα τεθούν υπό επεξεργασία, όταν επανέλθει το σύστημα.

στ)**Εγγύηση παράδοσης**: Ο πλεονασμός που επιτρέπουν οι ουρές μηνυμάτων, εγγυάται ότι ένα μήνυμα θα προσπελαστεί τελικά, εφόσον υπάρχει διαδικασία που να διαβάζει από την ουρά. Σε κάποιες ουρές μηνυμάτων μάλιστα, υπάρχει η εγγύηση μοναδικής παράδοσης του μηνύματος, ανεξάρτητα από τον αριθμό των διαδικασιών που διαβάζουν από την ουρά. Αυτό είναι δυνατό, διότι κάθε φορά που διαβάζεται ένα μήνυμα από την ουρά, αφαιρείται προσωρινά από αυτήν και αν δεν δηλωθεί από την διαδικασία που το διάβασε, ότι τελείωσε την επεξεργασία αυτού, το μήνυμα τοποθετείται και πάλι στην ουρά, προκειμένου να προσπελαστεί ξανά, μετά από κάποιο χρονικό διάστημα.

ζ)**Εγγύηση σειράς προσπέλασης**: Σε πολλές περιπτώσεις, η σειρά με την οποία θα γίνει η προσπέλαση των δεδομένων είναι σημαντική. Οι ουρές μηνυμάτων εξασφαλίζουν ότι τα δεδομένα θα τεθούν υπό επεξεργασία με την σειρά που προστέθηκαν σε αυτήν.

η)**Απομόνωση (Buffering)**: Σε κάθε μη τετριμμένο σύστημα, υπάρχουν διαδικασίες που χρειάζονται διαφορετικό χρόνο επεξεργασίας. Για παράδειγμα, το ανέβασμα μιας φωτογραφίας απαιτεί λιγότερο χρόνο από την εφαρμογή ενός φίλτρου σε αυτήν. Οι ουρές μηνυμάτων βοηθούν αυτές τις διαδικασίες να λειτουργούν με τη μέγιστη αποτελεσματικότητα, προσφέροντας την απομόνωση της κάθε διαδικασίας. Η διαδικασία που γράφει στην ουρά, μπορεί να γράψει όσο γρήγορα θέλει, χωρίς να περιορίζεται από την ετοιμότητα μιας άλλης διαδικασίας, η οποία διαβάζει από την ουρά. Αυτή η απομόνωση συνεπώς, συμβάλλει στον έλεγχο και την βελτιστοποίηση της ταχύτητας με την οποία ρέουν τα δεδομένα μέσα στο σύστημα.

θ)**Κατανόηση της ροής δεδομένων**: Σε ένα κατανεμημένο σύστημα, είναι δύσκολο να γίνει αντιληπτό το πόσο χρόνο χρειάζεται η εκτέλεση των διαφόρων διαδικασιών και γιατί. Οι ουρές μηνυμάτων, μέσω του ρυθμού με τον οποίο προσπελάζονται, βοηθούν στον εύκολο εντοπισμό των διαδικασιών που υπολειτουργούν καθώς και των περιοχών που η ροή δεδομένων δεν είναι η βέλτιστη.

ι)**Ασύγχρονη επικοινωνία**: Πολλές φορές, δεν είναι απαραίτητη η άμεση επεξεργασία ενός μηνύματος. Οι ουρές μηνυμάτων καθιστούν δυνατή την ασύγχρονη επεξεργασία, η οποία επιτρέπει την προσθήκη ενός μηνύματος στην ουρά, χωρίς να προσπελαστεί άμεσα.

Ωστόσο, οι ουρές μηνυμάτων παρά τα πολλά πλεονεκτήματά τους, έχουν και κάποια μειονεκτήματα. Για παράδειγμα, η ασύγχρονη επικοινωνία που προσφέρουν, μπορεί ταυτόχρονα να αποτελεί πρόβλημα, αφού μια διαδικασία για να συνεχίσει την λειτουργία της, πρέπει να περιμένει να τελειώσει μια άλλη διαδικασία τη δική της λειτουργία πρώτα. Επιπλέον, το γεγονός ότι κάθε μήνυμα πρέπει να προσπελαστεί με την σειρά με την οποία προστέθηκε στην ουρά, σημαίνει ότι, αν προκύψει καθυστέρηση στην επεξεργασία ενός μηνύματος, θα καθυστερήσει εξίσου και η επεξεργασία των επομένων μηνυμάτων. Τέλος, η χρήση των ουρών μηνυμάτων προϋποθέτει την προσαρμογή της αρχιτεκτονικής της εκάστοτε εφαρμογής, προκειμένου να συμπεριληφθούν οι ουρές μηνυμάτων.

3.5 Apache ActiveMQ

Υπάρχουν πολλές διαφορετικές εφαρμογές ουρών μηνυμάτων, όπου κάθε μία έχει και δικά της επιπλέον χαρακτηριστικά και δυνατότητες, πέρα από τα χαρακτηριστικά που έχουν όλες οι ουρές μηνυμάτων και χρησιμεύει σε διαφορετικές αρχιτεκτονικές εφαρμογών και συστημάτων. Κάποιες από τις πιο δημοφιλείς ουρές μηνυμάτων είναι: Apache Active MQ, RabbitMQ, ZeroMQ, IronMQ, RestMQ, κτλ.

Στην παρούσα διπλωματική επιλέχθηκε να χρησιμοποιηθεί η Apache ActiveMQ, διότι έχει κάποια χαρακτηριστικά που την καθιστούν καλύτερη σε σχέση με άλλες ουρές μηνυμάτων[16].
Αρχικά, είναι γρήγορη και φορητή (αφού είναι γραμμένη σε γλώσσα προγραμματισμού) και διαθέτει μεσάζοντα μηνυμάτων JMS (JMS message broker) ανοιχτού λογισμικού, κατάλληλο για ελαφρώς συνδεδεμένα περιβάλλοντα κατανεμημένων εφαρμογών. Βασίζεται σε Αρχιτεκτονική Προσανατολισμένη σε Υπηρεσίες, υποστηρίζει clustering, peer-to-peer federated δίκτυο και ποικιλία πρωτοκόλλων όπως τα TCP, SSL, UDP, Multicast.
Σε ό,τι αφορά τα τεχνικά χαρακτηριστικά της, παρέχει ανθεκτικότητα (persistence) μηνυμάτων και επιβεβαίωση παράδοσης, διαθέτει αποδεδειγμένη κλιμάκωση, διαθεσιμότητα και απόδοση που αυξάνονται ανάλογα με τις απαιτήσεις του πελάτη και υποστηρίζει δύο μοντέλα μηνυμάτων: αυτό της απλής ουράς από σημείο σε σημείο (point-to-point) και αυτό του εκδότη και των συνδρομητών (Publish/Subscribe), κατά το οποίο η ουρά ονομάζεται θέμα (topic).

Κεφάλαιο 4:Παροχή υπηρεσιών Cloud (μέσω ενός Marketplace)

Η συνεχής εξέλιξη των τεχνολογιών Cloud καθώς και τα πλεονεκτήματα που αυτές προσφέρουν (βλ. Κεφ.3) έχουν αυξήσει κατά πολύ την ζήτησή τους τα τελευταία χρόνια, με αποτέλεσμα την δημιουργία και ανάπτυξη μιας καινούριας αγοράς, η οποία αφορά την παροχή υπηρεσιών Cloud. Ωστόσο, καθώς οι τεχνολογίες Cloud ωριμάζουν και δημιουργούνται συνεχώς καινούριες υπηρεσίες, αυξάνονται συνεχώς και οι πάροχοί τους. Αυτή η διαρκής αύξηση του πλήθους των υπηρεσιών Cloud καθώς και ο μεγάλος αριθμός παρόχων και η πολυπλοκότητα των τιμολογιακών πολιτικών τους, συνεπάγονται την αστάθεια της αγοράς αυτής και καθιστούν ιδιαίτερα δύσκολη και περίπλοκη την επιλογή του καταλληλότερου συνδυασμού-πακέτου υπηρεσιών Cloud.

Μια πιθανή λύση στο πρόβλημα αυτό, είναι η δημιουργία μιας ηλεκτρονικής αγοράς (marketplace) υπηρεσιών Cloud, η οποία θα μπορεί για κάθε χρήστη να εντοπίσει τις υπηρεσίες αυτές που μπορούν να συνδυαστούν μεταξύ τους και ταυτόχρονα να ικανοποιήσουν τις απαιτήσεις του με τον καλύτερο δυνατό τρόπο, τόσο από άποψη τεχνική και ποιοτική, όσο και από άποψη οικονομική.

Έχουν γίνει προσπάθειες για την δημιουργία τέτοιων marketplaces, αλλά πρόκειται περισσότερο για καταστήματα παροχής υπηρεσιών Cloud μεμονωμένα ή για marketplaces που προτείνουν έτοιμους συνδυασμούς-πακέτα υπηρεσιών για την δημιουργία διαφόρων ειδών συστημάτων Cloud, χωρίς να δίνουν τη δυνατότητα προσωποποιημένης λύσης ανάλογα με τις επιθυμίες του κάθε χρήστη. Παραδείγματα τέτοιων καταστημάτων αποτελούν τα εξής: Windows Azure Marketplace, Google Apps Store, AppExchange, SuiteApp, Zoho, κτλ.

4.1 Περιγραφή του τρόπου λειτουργίας του marketplace

Μία τέτοια αγορά (marketplace) λοιπόν, για να μπορεί να είναι αποτελεσματική, θα πρέπει να μπορεί να λαμβάνει τις προδιαγραφές που επιθυμεί ο χρήστης να έχει το σύστημα που θέλει να υλοποιήσει, τόσο σε τεχνικό και ποιοτικό όσο και σε τιμολογιακό επίπεδο. Στη συνέχεια, να αναζητά τις υπηρεσίες που καλύπτουν τις τεχνικές προδιαγραφές του συστήματος, να εντοπίζει τους παρόχους αυτών και αφού υπολογίσει το κόστος τους, να προτείνει στον χρήστη την πιο συμφέρουσα για αυτόν λύση.

Ένα τέτοιο marketplace λοιπόν, που θα προτείνει προσωποποιημένες λύσεις σε κάθε χρήστη, θα πρέπει να αποτελείται από τα εξής συστατικά στοιχεία: ένα σύστημα που θα υπολογίζει με έξυπνο τρόπο την καλύτερη δυνατή λύση, μια διεπιφάνεια μέσω της οποίας θα επικοινωνεί το marketplace με τον χρήστη και μια βάση δεδομένων όπου θα υπάρχουν αποθηκευμένες όλες οι τεχνικές λεπτομέρειες των υπηρεσιών Cloud καθώς και όλοι οι πάροχοι αυτών, μαζί με τα ποιοτικά χαρακτηριστικά τους και τις τιμές που προσφέρουν.

Αρχικά ο χρήστης θα επιλέγει μέσω της διεπιφάνειας το σύστημα που θέλει να δημιουργήσει και στη συνέχεια θα προσδιορίζει τα χαρακτηριστικά αυτού. Ο αλγόριθμος θα παίρνει ως είσοδο τα χαρακτηριστικά αυτά, θα τα διαχωρίζει σε τεχνικά, ποιοτικά και οικονομικά και θα τα επεξεργάζεται χωριστά.

Στο πρώτο στάδιό του, ο αλγόριθμος θα αναζητά στην βάση δεδομένων και θα εντοπίζει όλους τους συνδυασμούς υπηρεσιών Cloud που πληρούν τις τεχνικές προδιαγραφές. Στο επόμενο στάδιο, για κάθε μία από τις υπηρεσίες του κάθε συνδυασμού, θα αναζητά και πάλι στην βάση δεδομένων όλους τους πιθανούς παρόχους και για κάθε πιθανό συνδυασμό υπηρεσιών και παρόχων, θα υπολογίζει την συνολική τιμή, μέσω μιας πολύπλοκης μαθηματικής συνάρτησης που θα λαμβάνει υπόψιν διαφορετικές μονάδες μέτρησης κόστους υπηρεσιών, αφού ανάλογα με την υπηρεσία και τον πάροχο, ο τρόπος χρέωσης αλλάζει. Μία υπηρεσία για παράδειγμα, μπορεί να χρεώνεται ανά kB ή ανά ώρα ή ανά μήνα ή να έχει κάποιο κόστος συνδρομής για συγκεκριμένο χρονικό διάστημα ή για απεριόριστη χρήση. Ακολούθως, θα απορρίπτει τους συνδυασμούς που ξεπερνούν την τιμή που έχει θέσει ως όριο ο χρήστης καθώς και αυτούς που δεν ικανοποιούν τα απαραίτητα ποιοτικά χαρακτηριστικά. Τέλος, θα συγκρίνει όλους τους συνδυασμούς που ικανοποιούν και τις τεχνικές και τις ποιοτικές προδιαγραφές και την επιθυμητή τιμή και θα καταλήγει στην καλύτερη δυνατή λύση την οποία θα προτείνει στον χρήστη ως συμβόλαιο έτοιμο προς υπογραφή.

4.2 Προβλήματα και επιπλοκές του προτεινόμενου μοντέλου marketplace

Το παραπάνω μοντέλο marketplace που προτείνεται ως λύση για την προσωποποιημένη εξυπηρέτηση του κάθε χρήστη έχει ωστόσο κάποια προβλήματα/μειονεκτήματα.

Αρχικά, το γεγονός ότι αφορά υπηρεσίες Cloud, του προσδίδουν αυτόματα τα μειονεκτήματα που έχει το Cloud ως τεχνολογία (βλ. Κεφ. 2). Διαθέτει επίσης, κάποια χαρακτηριστικά ως σύστημα, τα οποία μπορούν να το ανάγουν σε πρόβλημα Big Data.

Το πρώτο από τα χαρακτηριστικά αυτά είναι ο όγκος (Volume). Ο όγκος των δεδομένων που επεξεργάζεται και αποστέλλει το marketplace από το ένα στάδιο επεξεργασίας στο άλλο προκειμένου να εντοπίσει όλες τις διαθέσιμες υπηρεσίες Cloud και όλους τους πιθανούς παρόχους, να δημιουργήσει όλους τους πιθανούς συνδυασμούς αυτών, να λάβει υπόψιν του όλες τις παραμέτρους και τελικά να προσδιορίσει την καλύτερη δυνατή λύση, είναι πολύ μεγάλος, δεδομένου του πολύ μεγάλου αριθμού διαθέσιμων υπηρεσιών και παρόχων και αυξάνεται ανάλογα με την πολυπλοκότητα του επιθυμητού από τον χρήστη συστήματος.

Το δεύτερο χαρακτηριστικό του, που το καθιστά σύστημα Big Data, είναι η ταχύτητα (Velocity) με την οποία παράγονται συνεχώς καινούρια δεδομένα. Δεδομένου ότι στο marketplace θα έχουν πρόσβαση ταυτόχρονα πολλοί χρήστες, είναι εύκολα κατανοητό το γεγονός ότι όλος αυτός ο όγκος δεδομένων που παράγεται και τίθεται υπό επεξεργασία για την εξυπηρέτηση κάθε χρήστη, αφενός πολλαπλασιάζεται σε μέγεθος όταν εξυπηρετούνται πολλοί χρήστες ταυτόχρονα αλλά κυρίως έχει ως αποτέλεσμα την συνεχή δημιουργία καινούριων δεδομένων με πολύ μεγάλη ταχύτητα.

Το τρίτο χαρακτηριστικό των Big Data που συναντάται στο σύστημα αυτό, είναι η ποικιλία των δεδομένων που διαχειρίζεται. Κάθε στάδιο του συστήματος αυτού, διαθέτει διαφορετικό τρόπο προσέγγισης και διαφορετικό αλγόριθμο επεξεργασίας επειδή δέχεται ως είσοδο και επεξεργάζεται δεδομένα που αφορούν διαφορετικά χαρακτηριστικά (τεχνικά, ποιοτικά, οικονομικά).

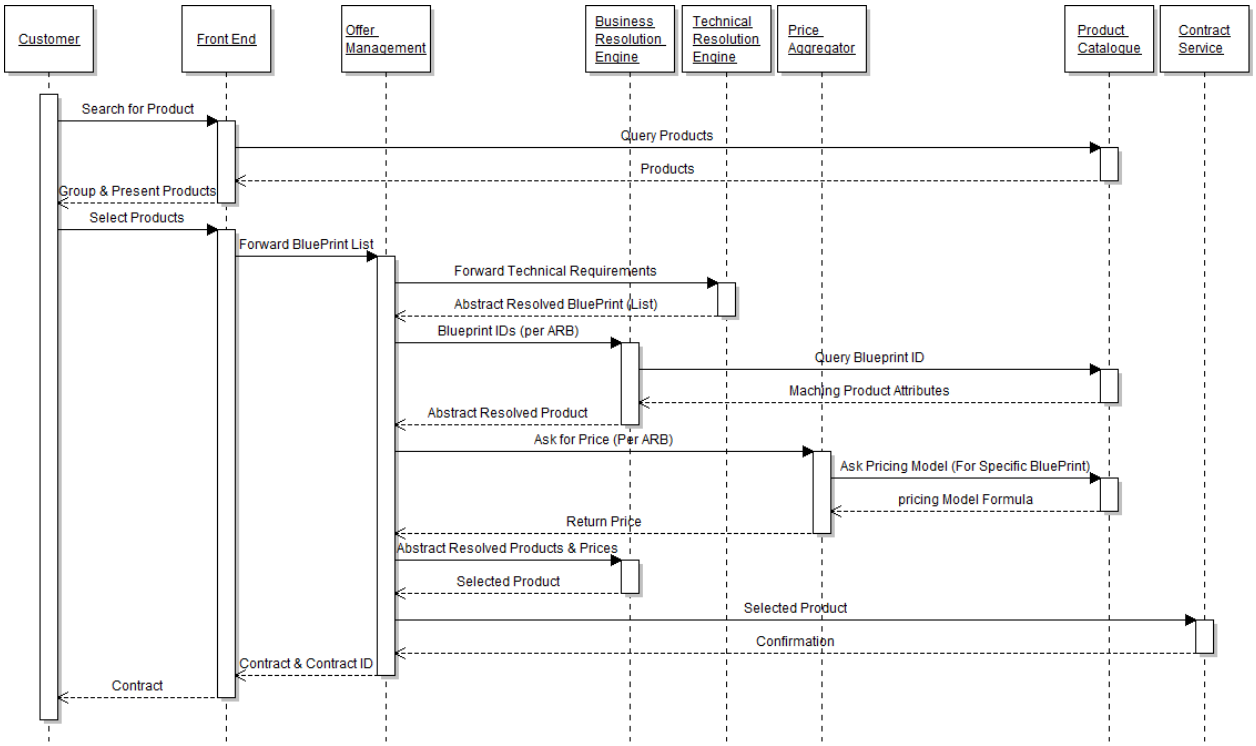
Το τελευταίο από τα χαρακτηριστικά των Big Data που διαθέτει το marketplace, είναι η μεταβλητότητα (Variability). Όπως αναφέρθηκε και παραπάνω, η διαρκής εξέλιξη των

τεχνολογιών Cloud και η συνεχής αλλαγή των δεδομένων της αγοράς αυτής σε ό,τι αφορά τις διαθέσιμες υπηρεσίες, τους παρόχους αυτών και τις τιμές τους, έχει ως αποτέλεσμα να μεταβάλλονται συνεχώς και τα δεδομένα που πρέπει να έχει αποθηκευμένα στην βάση δεδομένων του και κατ'επέκταση τα δεδομένα που λαμβάνει υπόψιν και διαχειρίζεται.

4.3 Παράδειγμα υλοποίησης ενός marketplace υπηρεσιών Cloud

Στα πλαίσια μιας παλαιότερης εργασίας, είχε υλοποιηθεί ένα marketplace υπηρεσιών Cloud, το οποίο λειτουργεί με τον τρόπο που περιγράφηκε παραπάνω και το οποίο ονομάζεται 4CaaSt[17]. Το 4CaaSt υποστηρίζει το εμπόριο υπηρεσιών Cloud όλων των ειδών (SaaS, PaaS, IaaS), ενώ έχει τη δυνατότητα να προτείνει προσωποποιημένες λύσεις ανάλογα με τις απαιτήσεις του χρήστη και να επεξεργαστεί διαφόρων ειδών τιμολογήσεις, προκειμένου να καταλήξει στην βέλτιστη λύση. Στο παρακάτω σχήμα (Σχήμα χχ), φαίνεται ο τρόπος λειτουργίας του 4CaaSt, ο οποίος συμφωνεί με τα όσα περιγράφηκαν παραπάνω, για τον τρόπο λειτουργίας ενός marketplace.

Αρχικά, ο χρήστης-πελάτης (Customer) αναζητά, τα διαθέσιμα προϊόντα, μέσω του front end, τα οποία το 4CaaSt αναζητά στον κατάλογο προϊόντων (Product Catalogue),ο οποίος είναι υπεύθυνος για τις αναζητήσεις στη βάση δεδομένων του συστήματος, τα ομαδοποιεί και τα παρουσιάζει στον χρήστη. Ο χρήστης επιλέγει το προϊόν που επιθυμεί και δίνει μέσω του front end στο σύστημα τις προδιαγραφές που θέλει να έχει αυτό το προϊόν(Blueprint List), τις οποίες το front end προωθεί στο Offer Management, το οποίο παίζει τον ρόλο του ελεγκτή και του συντονιστή της επικοινωνίας, ανάμεσα σε όλα τα επιμέρους στοιχεία του συστήματος. Αυτό με τη σειρά του, προωθεί τις τεχνικές προδιαγραφές (Technical Requirements) στην μηχανή του Technical Resolution, η οποία επιστρέφει μια λίστα με Abstract Resolved Blueprints (ARB) στο Offer Management. Αυτό με τη σειρά του, για κάθε ARB, στέλνει τα Blueprint IDs στη μηχανή του Business Resolution, η οποία αναζητά στο Product Catalogue, καθένα από αυτά τα IDs και του επιστρέφονται τα προϊόντα που πληρούν τις τεχνικές προδιαγραφές, μαζί με τα χαρακτηριστικά τους, τα οποία και προωθεί στο Offer Management. Έπειτα, για κάθε Arb, το Offer Management, ζητά από τον Price Aggregator να υπολογίσει την τιμή του κι αυτός με τη σειρά του, ζητά από το Product Catalogue το τιμολογιακό μοντέλο που ακολουθείται ανάλογα με το Blueprint. Του στέλνεται πίσω μια φόρμουλα τιμολόγησης και βάσει αυτής, υπολογίζει την τιμή και την επιστρέφει στο Offer Management. Στη συνέχεια, το Offer Management, στέλνει στην μηχανή του Business Resolution μια λίστα, με όλες τις πιθανές λύσεις μαζί με τις τιμές τους και αυτή του επιστρέφει την καλύτερη λύση από άποψη τεχνολογική, επιχειρηματική και τιμολογιακή. Τέλος, η επιλεγμένη λύση, στέλνεται στην υπηρεσία συμβολαίων (Contract Service) και αφού εγκριθεί στέλνεται ως συμβόλαιο στο front end και στον χρήστη.



Σχήμα 18: Σχηματική απεικόνιση του τρόπου λειτουργίας του marketplace 4CaaSt (πηγή:[17])

4.4 Τρόπος ενδοεπικοινωνίας του marketplace 4CaaSt

Παρατηρούμε ότι στο marketplace 4CaaSt, η επικοινωνία ανάμεσα στα διάφορα στάδια του συστήματος γίνεται με πακέτα δεδομένων. Για παράδειγμα, όταν το Offer Management στέλνει στην μηχανή του Technical Resolution τις τεχνικές προδιαγραφές του χρήστη, το Technical Resolution, πρώτα θα εντοπίσει όλα τα Blueprints που τις ικανοποιούν και στη συνέχεια θα τα στείλει πίσω όλα μαζί σε μία λίστα. Το ίδιο συμβαίνει ανάμεσα και στα υπόλοιπα στάδια επεξεργασίας του4CaaSt.

Αυτός ο τρόπος επικοινωνίας ωστόσο, δεν είναι αποδοτικός, διότι καθυστερεί την επεξεργασία των δεδομένων. Ο πρώτος λόγος που προκαλείται καθυστέρηση είναι ότι το επόμενο στάδιο του συστήματος, θα μπορούσε να αρχίσει να επεξεργάζεται τα αποτελέσματα του προηγούμενο σταδίου που είναι ήδη έτοιμα, ένα-ένα και να μην χρειάζεται να περιμένει να δημιουργηθούν όλα τα αποτελέσματα. Ο δεύτερος λόγος καθυστέρησης, προκύπτει αν το σύστημα είναι κατανεμημένο, το οποίο είναι αρκετά πιθανό λόγω της έκτασης και της φύσης του marketplace. Στην περίπτωση αυτή λοιπόν, η μετάδοση δεδομένων απαιτεί αρκετό χρόνο, ο οποίος θα μπορούσε να είναι μικρότερος, αν η ανταλλαγή δεδομένων-μηνυμάτων από το ένα στάδιο στο άλλο, δεν γινόταν σε πακέτα αλλά μεμονωμένα, κάθε φορά που κάποιο αποτέλεσμα ήταν έτοιμο.

Μια λύση στο πρόβλημα αυτό, είναι η χρήση ουρών μηνυμάτων (message queues) για την επικοινωνία ανάμεσα στα διάφορα στοιχεία του marketplace. Οι ουρές μηνυμάτων θα βελτιώσουν την απόδοση του συστήματος, θα συμβάλουν στην επίτευξη κάποιου παραλληλισμού κατά την διαδικασία επίλυσης του προβλήματος, ενώ θα εξασφαλίσουν την αδιάκοπη λειτουργία όλων των υπολοίπων σταδίων, όταν κάποιο από αυτά αντιμετωπίζει αυξημένο φόρτο εργασίας και δεν μπορεί να δεχθεί περαιτέρω μηνύματα εκείνη την στιγμή.

Η χρήση των ουρών μηνυμάτων μπορεί να γίνει με δύο τρόπους: είτε με ξεχωριστές ουρές, οι οποίες θα συνδέουν άμεσα κάθε στάδιο με το επόμενό του, είτε με μια κοινή ουρά, η οποία θα δέχεται τα μηνύματα όλων των σταδίων και θα τα προωθεί ανάλογα με τον εκάστοτε παραλήπτη. Στην παρούσα διπλωματική εργασία, θα θεωρηθεί δεδομένο ότι η ενσωμάτωση των ουρών μηνυμάτων επιφέρει βελτίωση στην απόδοση του συστήματος και θα μελετηθεί η συμπεριφορά του συστήματος κατά την χρήση ουρών μηνυμάτων με τους δύο παραπάνω τρόπους και μάλιστα σε διάφορες περιπτώσεις, ώστε να προκύψει ποιος τρόπος χρήσης υπερτερεί τελικά σε κάθε περίπτωση.

Κεφάλαιο 5:Υλοποίηση marketplace υπηρεσιών Cloud

Στην παρούσα διπλωματική χρησιμοποιήθηκε αρχιτεκτονική προσανατολισμένη στις υπηρεσίες (SOA) για την επέκταση και υλοποίηση του συστήματος marketplace 4CaaSt, το οποίο είχε αναπτυχθεί στα πλαίσια ευρωπαϊκού έργου. Το σύστημα χωρίστηκε σε 4 στάδια- web services, ενώ για την επικοινωνία μεταξύ των σταδίων, χρησιμοποιήθηκε η δομή δεδομένων της ουράς μηνυμάτων και συγκεκριμένα, η ουρά μηνυμάτων Apache ActiveMQ. Ακόμη, για την ενορχήστρωση του συστήματος και τον συντονισμό της ροής δεδομένων μέσα σε αυτό, χρησιμοποιήθηκε η μηχανή δρομολόγησης και μεσολάβησης Apache Camel, ενώ ο εξυπηρετητής (server) που επιλέχθηκε για τα web services ήταν ο Apache Tomcat Web Server. Το σύνολο του κώδικα είναι γραμμένο στη γλώσσα αντικειμενοστραφούς προγραμματισμού JAVA, λόγω της πολύ καλής διασύνδεσής της, τόσο με την ActiveMQ και την Apache Camel, καθώς και με τον server που επιλέχθηκε, τον Apache Tomcat. Τέλος, η είσοδος και η έξοδος του συστήματος δίνονται σε μορφή XML αρχείου.

5.1 Παρεχόμενες υπηρεσίες και πάροχοι αυτών

Στα πλαίσια της παρούσας εργασίας, θεωρήσαμε ότι το επιθυμητό σύστημα μπορεί να αποτελείται από των συνδυασμό 4 βασικών υπηρεσιών: μίας βάσης δεδομένων (Database), ενός λειτουργικού συστήματος (OperatingSystem), μιας πλατφόρμας διαχείρισης πελατειακών σχέσεων (CRM) και μιας σύνδεσης στο δίκτυο (Network).

Για να μπορέσει το σύστημα να λειτουργήσει, χρειάζεται να έχει πρόσβαση σε όλες τις πληροφορίες που αφορούν τις παρεχόμενες υπηρεσίες και τους πιθανούς παρόχους αυτών. Για τον σκοπό αυτό χρησιμοποιήθηκε ένας φάκελος μέσα στο file system του υπολογιστή, ο οποίος έπαιξε το ρόλο της βάσης δεδομένων του συστήματος και μέσα σε αυτόν αποθηκεύσαμε με τη μορφή αρχείων XML, τις απαραίτητες πληροφορίες. Στη συνέχεια, δίνονται τα χαρακτηριστικά που αποθηκεύσαμε χωριστά για κάθε υπηρεσία και για κάθε πάροχο, μέσα στον φάκελο αυτό.

Για κάθε μία βάση δεδομένων, θεωρήσαμε απαραίτητα για την περιγραφή της τα εξής χαρακτηριστικά: το όνομά της (Name), την αρχιτεκτονική της (OP) η οποία μπορεί να είναι είτε x64 είτε x86 και το είδος της (Type), αφού μπορεί να είναι σχεσιακή (SQL) ή μη σχεσιακή (NoSQL). Επιπλέον, για τη διευκόλυνση της επεξεργασίας από το σύστημά μας, δώσαμε και έναν αναγνωριστικό αριθμό σε κάθε βάση δεδομένων (ID). Παρακάτω δίνεται ένα παράδειγμα αρχείου XML, που περιγράφει μια βάση δεδομένων.

```
<Database>
        <ID>1002003205</ID>
        <Name>PostgreSQL</Name>
        <OP>x86</OP>
        <Type>SQL</Type>
</Database>
```

Αντίστοιχα, για κάθε λειτουργικό σύστημα, θεωρήσαμε απαραίτητα για την περιγραφή του, τα εξής χαρακτηριστικά: το είδος του (Type), δηλαδή αν πρόκειται για Linux, MacOS ή Windows, τη διανομή του (Distribution), για παράδειγμα 7 ή 8 ή 10 κτλ, αν πρόκειται για Windows καθώς και την αρχιτεκτονική τους (OP), δηλαδή αν υποστηρίζουν αρχιτεκτονική x64 ή x86. Και εδώ, για τη διευκόλυνση της επεξεργασίας από το σύστημά μας, δώσαμε και έναν αναγνωριστικό αριθμό σε κάθε λειτουργικό σύστημα (ID). Παρακάτω δίνεται ένα παράδειγμα αρχείου XML, που περιγράφει ένα λειτουργικό σύστημα.

```
<OperatingSystem>
        <ID>2002003205</ID>
        <Distribution>10</Distribution>
        <Type>Windows</Type>
        <OP>x64</OP>
</OperatingSystem>
```

Για κάθε πλατφόρμα διαχείρισης πελατειακών σχέσεων (CRM), ορίσαμε μόνο δυο χαρακτηριστικά ως απαραίτητα για την περιγραφή της: το όνομα (Name) και την αρχιτεκτονική που υποστηρίζει (OP). Ακόμη, χάριν διευκόλυνσης της επεξεργασίας από το σύστημά μας, δώσαμε και έναν αναγνωριστικό αριθμό (ID), σε κάθε μία από τις πλατφόρμες CRM. Παρακάτω δίνεται ένα παράδειγμα αρχείου XML, που περιγράφει μία πλατφόρμα CRM.

```
<CRM>
        <ID>3002003202</ID>
        <Name>Drupal 8 beta 1</Name>
        <OP>x86</OP>
</CRM>
```

Τέλος, με ανάλογο τρόπο δώσαμε την περιγραφή κάθε μίας πιθανής σύνδεσης σε δίκτυο, χρησιμοποιώντας τα εξής χαρακτηριστικά: το όνομα της σύνδεσης (Name), που αφορά το αν πρόκειται για ADSL ή VDSL, το είδος της σύνδεσης (Type), το οποίο αναφέρεται στο αν πρόκειται για σύνδεση με οπτικές ίνες (fiber) ή με καλώδια χαλκού (copper) και τέλος, έναν αναγνωριστικό αριθμό (ID), για την διευκόλυνση της επεξεργασίας. Παρακάτω δίνεται ένα παράδειγμα αρχείου XML, που περιγράφει μία σύνδεση δικτύου.

```
<Network>
        <ID>4002003202</ID>
        <Name>VDSL Connection</Name>
        <Type>copper</Type>
</Network>
```

Για κάθε μία από τις προαναφερθείσες υπηρεσίες, δώσαμε μία απλοποιημένη περιγραφή του κάθε παρόχου αυτής, κοινή για όλους τους παρόχους, ανεξαρτήτως υπηρεσίας, με τα

εξής χαρακτηριστικά: το όνομά του (ServiceProvider), το επίπεδο των ποιοτικών χαρακτηριστικών που προσφέρει, τα οποία αναλύονται στη συνέχεια και η τιμή (Price)στην οποία προσφέρει την υπηρεσία αυτή. Τα ποιοτικά χαρακτηριστικά είναι πέντε: η διαθεσιμότητα (Availability), εκφρασμένη ως ποσοστό επί τοις εκατό του χρόνου που εγγυάται ο πάροχος ότι η υπηρεσία θα είναι διαθέσιμη, η ασφάλεια (Security), η φήμη (Reputation) και η αξιοπιστία (Reliability), οι οποίες βαθμολογούνται από το 1 ως το 5 και η εμπιστευτικότητα (Confidentiality), η οποία είτε παρέχεται είτε όχι. Επιπλέον, διαθέτει και έναν αναγνωριστικό αριθμό (ID), ο οποίος ταυτίζεται με τον αναγνωριστικό αριθμό της παρεχόμενης υπηρεσίας, προκειμένου να διευκολυνθούν οι υπολογισμοί. Παρακάτω δίνεται ένα παράδειγμα παρόχου μιας βάσης δεδομένων, σε μορφή αρχείου XML.

```
<Database>
        <ID>1002003214</ID>
        <ServiceProvider>Amazon</ServiceProvider>
        <Availability>77</Availability>
        <Security>5</Security>
        <Reputation>3</Reputation>
        <Reliability>4</Reliability>
        <Confidentiality>NO</Confidentiality>
        <Price>7000</Price>
</Database>
```

5.2 Είσοδος του συστήματος - Προδιαγραφές χρήστη

Το σύστημά μας, δέχεται ως είσοδο ένα αρχείο XML, στο οποίο περιλαμβάνονται οι προδιαγραφές που επιθυμεί ο χρήστης να έχει η λύση που θα του προταθεί.
Το αρχείο αυτό έχει την εξής μορφή:

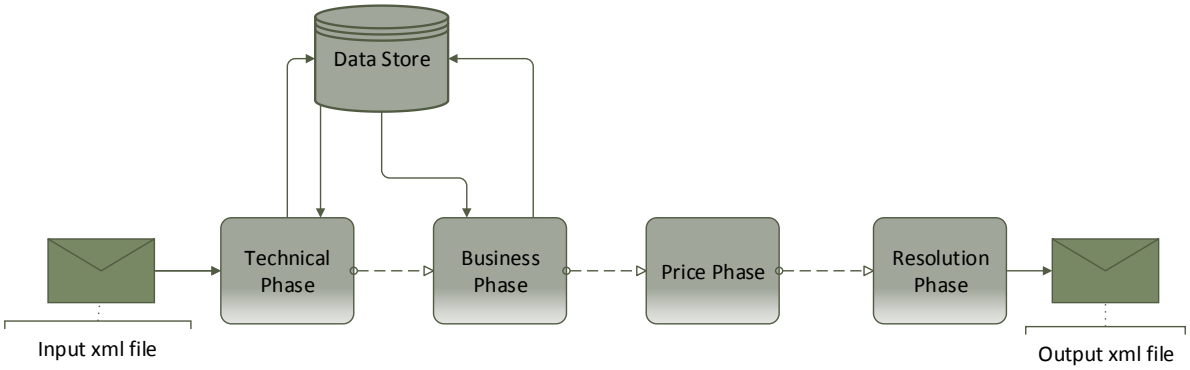
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<UserSystem>
<Id>1</Id>
<UserRequirements>
        <Optimization>
                <MAXPrice>12000</MAXPrice>
                <MINAvailability>87</MINAvailability>
                <MINSecurity>3</MINSecurity>
                <MINReputation>2</MINReputation>
                <MINReliability>3</MINReliability>
                <Confidentiality>YES</Confidentiality>
        </Optimization>
        <Service>
                <Database>
                        <OP>x64</OP>
                        <Type>SQL</Type>
```

```
            </Database>
            <OperatingSystem>
                  <Type>Linux</Type>
                  <OP>x64</OP>
            </OperatingSystem>
            <CRM>
                  <OP>x64</OP>
             </CRM>
            <Network>
                  <Type>copper</Type>
             </Network>
      </Service>
</UserRequirements>
<Technical></Technical>
<Business></Business>
</UserSystem>
```

Το Id αναφέρεται στο στάδιο το οποίο θα επεξεργαστεί το μήνυμα αυτό (1: Technical Phase, 2: Business Phase, 3:PricePhase, 4: Resolution, τα οποία επεξηγούνται παρακάτω). Το τμήμα <UserRequirements> αφορά τις απαιτήσεις του χρήστη και χωρίζεται σε δύο τμήματα: το <Service>, όπου δίνονται τα τεχνικά χαρακτηριστικά που επιθυμεί ο χρήστης και το <Optimisation>, όπου δίνεται το επίπεδο των επιθυμητών ποιοτικών χαρακτηριστικών. Στην παρούσα διπλωματική εργασία, μελετήσαμε την περίπτωση όπου μόνο ένας χρήστης χρησιμοποιεί το σύστημα, προκειμένου να γίνει πιο εύκολη η ανάλυση της επεξεργασίας που γίνεται για ένα αίτημα μεμονωμένο.

5.3 Περιγραφή του συστήματος των 4 σταδίων

Το κεντρικό τμήμα του συστήματός μας, αποτελείται από τέσσερα στάδια επεξεργασίας, τα οποία είναι web services και τρέχουν στον server που τρέχουμε τοπικά στον υπολογιστή, ενώ η ροή των δεδομένων γίνεται σειριακά από στάδιο σε στάδιο (Σχήμα 19).



Σχήμα 19: Σχηματική απεικόνιση της αρχιτεκτονικής του συστήματος

1ο στάδιο: Technical Phase

Το 1ο στάδιο, το οποίο ονομάσαμε Technical Phase (Τεχνική φάση), δέχεται ως είσοδο το αρχείο XML με τις προδιαγραφές του χρήστη και καλεί το web service που ονομάζεται Technical Controller, το οποίο με βάση τα τεχνικά χαρακτηριστικά της εισόδου, εντοπίζει όλες τις επιμέρους υπηρεσίες που τα ικανοποιούν και σχηματίζει όλους τους πιθανούς συνδυασμούς-πακέτα όλων αυτών, τους οποίους και προωθεί στο επόμενο στάδιο. Σημειώνεται εδώ, ότι το αρχείο εισόδου XML, μετατρέπεται σε αντικείμενο της Java προκειμένου να το επεξεργαστεί το 1ο στάδιο, ενώ αντικείμενα Java είναι και αυτά που αποστέλλονται από το 1ο στάδιο στο 2ο.

2ο στάδιο: Business Phase

Το 2ο στάδιο, το οποίο ονομάσαμε Business Phase (Επιχειρηματική φάση), δέχεται ως είσοδο τον κάθε πιθανό συνδυασμό-πακέτο υπηρεσιών που καλύπτει τις τεχνικές προδιαγραφές του χρήστη, όπως αυτό σχηματίστηκε από το 1ο στάδιο και στη συνέχεια, καλεί το web service που ονομάζεται Business Controller, το οποίο προσδιορίζει τους πιθανούς παρόχους της κάθε επιμέρους υπηρεσίας του πακέτου αυτού και σχηματίζει όλους τους πιθανούς συνδυασμούς παρόχων για το συγκεκριμένο πακέτο υπηρεσιών, τους οποίους στέλνει στο επόμενο στάδιο. Και εδώ, οι πληροφορίες που δέχεται και αποστέλλει το στάδιο αυτό, είναι σε μορφή αντικειμένου Java.

3ο στάδιο: Price Phase

Στο 3ο στάδιο, το οποίο πήρε το όνομα Price Phase (φάση Τιμής), δέχεται ως είσοδο κάθε πιθανό πακέτο υπηρεσιών και καλεί το web service που ονομάζεται Price Controller, το οποίο υπολογίζει το συνολικό κόστος του συγκεκριμένου πακέτου. Στο σημείο αυτό, επειδή η μαθηματική συνάρτηση υπολογισμού της συνολικής τιμής, είναι αρκετά πολύπλοκη και λαμβάνει υπόψιν της τιμές με διαφορετικό τρόπο κοστολόγησης και δεδομένου ότι ο στόχος της διπλωματικής είναι η μελέτη και η σύγκριση των διαφορετικών μοντέλων επικοινωνίας της αρχιτεκτονικής αυτής, έγινε η παραδοχή ότι η κάθε υπηρεσία έχει μια σταθερή τιμή και μονάδα κοστολόγησης και ότι η συνολική τιμή υπολογίζεται ως απλό άθροισμα των επιμέρους τιμών. Αυτή η συνολική τιμή, μαζί με τα υπόλοιπα χαρακτηριστικά του εκάστοτε πακέτου, περνάει ως πληροφορία στο επόμενο στάδιο, όπου θα ολοκληρωθεί η επεξεργασία.

4ο στάδιο: Resolution

Το 4ο και τελευταίο στάδιο επεξεργασίας του συστήματος, δέχεται ως είσοδο (σε μορφή αντικειμένου Java) το κάθε πιθανό πακέτο και καλεί το web service που ονομάζεται Resolution Controller, το οποίο λαμβάνοντας υπόψιν τα ποιοτικά χαρακτηριστικά της κάθε υπηρεσίας, υπολογίζει το επίπεδο των ποιοτικών χαρακτηριστικών που έχει τελικά το συγκεκριμένο πακέτο και στη συνέχεια, προσδιορίζει έναν γενικό δείκτη (TotalTotal), ο οποίος αντιπροσωπεύει το συνολικό επίπεδο του πακέτου, τόσο από ποιοτική όσο και από οικονομική άποψη. Τέλος, επιλέγει ανάμεσα σε όλα τα πακέτα που πληρούν τις ποιοτικές προδιαγραφές του χρήστη, αυτό με τον καλύτερο γενικό δείκτη και αφού το μετατρέψει σε αρχείο XML, το δίνει ως έξοδο.

Έξοδος συστήματος

Η έξοδος του συστήματος, είναι ένα αρχείο XML, στο οποίο δίνεται πλήρης περιγραφή των προδιαγραφών του χρήστη, των επιμέρους υπηρεσιών που τελικά επιλέχθηκαν, μαζί με τα χαρακτηριστικά αυτών και των παρόχων τους, της συνολικής τιμής της προτεινόμενης λύσης καθώς και του επιπέδου των ποιοτικών χαρακτηριστικών της. Παράδειγμα ενός τέτοιου αρχείου εξόδου σε μορφή XML, δίνεται παρακάτω.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<UserSystem>
   <Id>3</Id>
   <UserRequirements>
      <Optimization>
         <MAXPrice>10500</MAXPrice>
         <MINAvailability>80</MINAvailability>
         <MINSecurity>3</MINSecurity>
         <MINReputation>3</MINReputation>
         <MINReliability>3</MINReliability>
         <Confidentiality>YES</Confidentiality>
      </Optimization>
      <Service>
         <Database>
            <ID>0</ID>
            <Name></Name>
            <OP>x86</OP>
            <Type>NoSQL</Type>
            <ServiceProvider></ServiceProvider>
            <Availability>0</Availability>
            <Security>0</Security>
            <Reputation>0</Reputation>
            <Reliability>0</Reliability>
            <Confidentiality>NO</Confidentiality>
            <Price>0</Price>
         </Database>
         <OperatingSystem>
            <ID>0</ID>
            <Distribution></Distribution>
            <Type>Linux</Type>
            <OP>x86</OP>
            <ServiceProvider></ServiceProvider>
            <Availability>0</Availability>
            <Security>0</Security>
            <Reputation>0</Reputation>
            <Reliability>0</Reliability>
            <Confidentiality>NO</Confidentiality>
            <Price>0</Price>
         </OperatingSystem>
         <CRM>
            <ID>0</ID>
```

```
            <Name></Name>
            <OP>x86</OP>
            <ServiceProvider></ServiceProvider>
            <Availability>0</Availability>
            <Security>0</Security>
            <Reputation>0</Reputation>
            <Reliability>0</Reliability>
            <Confidentiality>NO</Confidentiality>
            <Price>0</Price>
        </CRM>
        <Network>
            <ID>0</ID>
            <Name></Name>
            <Type>copper</Type>
            <ServiceProvider></ServiceProvider>
            <Availability>0</Availability>
            <Security>0</Security>
            <Reputation>0</Reputation>
            <Reliability>0</Reliability>
            <Confidentiality>NO</Confidentiality>
            <Price>0</Price>
        </Network>
    </Service>
</UserRequirements>
<Technical>
    <database>
        <ID>1002003230</ID>
        <Name>Neo4J</Name>
        <OP>x86</OP>
        <Type>NoSQL</Type>
        <ServiceProvider></ServiceProvider>
        <Availability>0</Availability>
        <Security>0</Security>
        <Reputation>0</Reputation>
        <Reliability>0</Reliability>
        <Confidentiality>NO</Confidentiality>
        <Price>0</Price>
    </database>
    <operatingSystem>
        <ID>2002003201</ID>
        <Distribution>Debian</Distribution>
        <Type>Linux</Type>
        <OP>x86</OP>
        <ServiceProvider></ServiceProvider>
        <Availability>0</Availability>
        <Security>0</Security>
        <Reputation>0</Reputation>
        <Reliability>0</Reliability>
```

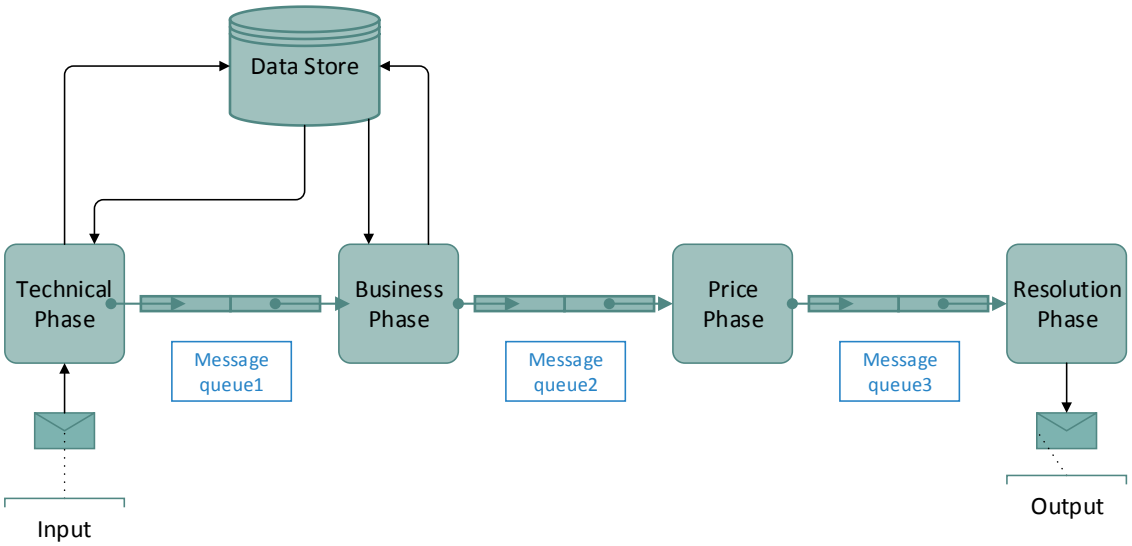
```
      <Confidentiality>NO</Confidentiality>
      <Price>0</Price>
    </operatingSystem>
    <crm>
      <ID>3002003202</ID>
      <Name>Drupal 8 beta 1</Name>
      <OP>x86</OP>
      <ServiceProvider></ServiceProvider>
      <Availability>0</Availability>
      <Security>0</Security>
      <Reputation>0</Reputation>
      <Reliability>0</Reliability>
      <Confidentiality>NO</Confidentiality>
      <Price>0</Price>
    </crm>
    <network>
      <ID>4002003201</ID>
      <Name>ADSL Connection</Name>
      <Type>copper</Type>
      <ServiceProvider></ServiceProvider>
      <Availability>0</Availability>
      <Security>0</Security>
      <Reputation>0</Reputation>
      <Reliability>0</Reliability>
      <Confidentiality>NO</Confidentiality>
      <Price>0</Price>
    </network>
  </Technical>
  <Business>
    <database>
      <ID>1002003230</ID>
      <Name>Neo4J</Name>
      <OP>x86</OP>
      <Type>NoSQL</Type>
      <ServiceProvider>RedHat</ServiceProvider>
      <Availability>52</Availability>
      <Security>4</Security>
      <Reputation>4</Reputation>
      <Reliability>5</Reliability>
      <Confidentiality>YES</Confidentiality>
      <Price>2800</Price>
    </database>
    <operatingSystem>
      <ID>2002003201</ID>
      <Distribution>Debian</Distribution>
      <Type>Linux</Type>
      <OP>x86</OP>
      <ServiceProvider>Amazon</ServiceProvider>
```

```
            <Availability>99</Availability>
            <Security>5</Security>
            <Reputation>5</Reputation>
            <Reliability>4</Reliability>
            <Confidentiality>YES</Confidentiality>
            <Price>100</Price>
        </operatingSystem>
        <crm>
            <ID>3002003202</ID>
            <Name>Drupal 8 beta 1</Name>
            <OP>x86</OP>
            <ServiceProvider>Drupal Inc</ServiceProvider>
            <Availability>92</Availability>
            <Security>5</Security>
            <Reputation>3</Reputation>
            <Reliability>4</Reliability>
            <Confidentiality>YES</Confidentiality>
            <Price>90</Price>
        </crm>
        <network>
            <ID>4002003201</ID>
            <Name>ADSL Connection</Name>
            <Type>copper</Type>
            <ServiceProvider>CosmOTE</ServiceProvider>
            <Availability>80</Availability>
            <Security>5</Security>
            <Reputation>5</Reputation>
            <Reliability>4</Reliability>
            <Confidentiality>YES</Confidentiality>
            <Price>110</Price>
        </network>
    </Business>
    <TotalAvailability>80</TotalAvailability>
    <TotalSecurity>4</TotalSecurity>
    <TotalConfidentiality>YES</TotalConfidentiality>
    <TotalReliability>3</TotalReliability>
    <TotalReputation>4</TotalReputation>
    <TotalPrice>3100</TotalPrice>
    <TotalTotal>-190</TotalTotal>
</UserSystem>
```

5.4 Περιγραφή των μοντέλων της αρχιτεκτονικής

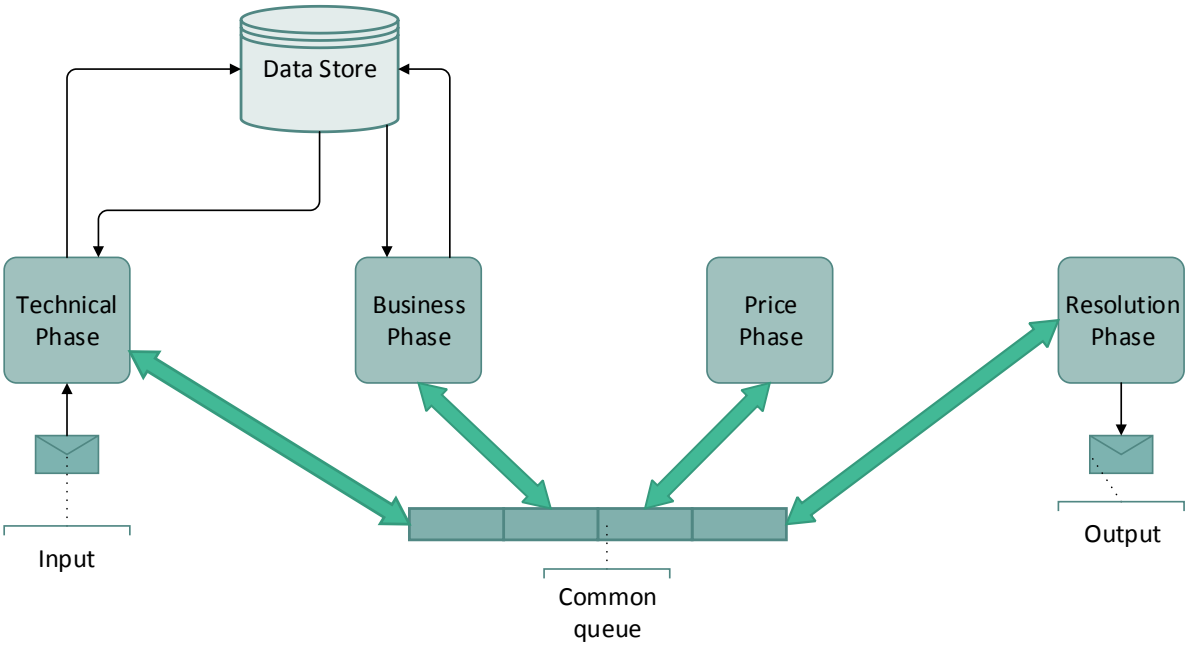
Στην παρούσα διπλωματική, μελετήθηκε η συμπεριφορά του marketplace σε δύο διαφορετικά μοντέλα επικοινωνίας της παραπάνω αρχιτεκτονικής των 4 σταδίων. Στο πρώτο μοντέλο, χρησιμοποιήθηκαν τρεις ξεχωριστές ουρές για την επικοινωνία και την

ανταλλαγή μηνυμάτων μεταξύ του 1ου με το 2ο στάδιο, του 2ου με το 3ο και του 3ου με το 4ο στάδιο. Η υλοποίησή τους έγινε με τη χρήση των queues της ActiveMQ, ενώ στο παρακάτω σχήμα, φαίνεται το μοντέλο που περιγράφεται (Σχήμα 20).



Σχήμα 20: Σχηματική απεικόνιση του πρώτου μοντέλου της αρχιτεκτονικής του marketplace που υλοποιήθηκε

Στο δεύτερο μοντέλο που μελετήθηκε, χρησιμοποιήθηκε μια κοινή ουρά για την επικοινωνία όλων των σταδίων με τα υπόλοιπα, με αποτέλεσμα η ροή των δεδομένων να πρέπει να καθοριστεί με άλλο τρόπο, αφού δεν είναι σαφής όπως στην περίπτωση του πρώτου μοντέλου. Για την υλοποίηση της κοινής ουράς, χρησιμοποιήθηκε πάλι μια ουρά της ActiveMQ, ενώ την διαχείριση των μηνυμάτων και την αποστολή των σωστών μηνυμάτων σε κάθε παραλήπτη, ανέλαβε η μηχανή δρομολόγησης και μεσολάβησης Apache Camel. Στο παρακάτω σχήμα (Σχήμα 21), φαίνεται το δεύτερο μοντέλο της αρχιτεκτονικής αυτής.



Σχήμα 21: Σχηματική απεικόνιση του δευτέρου μοντέλου της αρχιτεκτονικής του marketplace που υλοποιήθηκε

Κεφάλαιο 6:Υλοποίηση της αρχιτεκτονικής marketplace και αποτελέσματα

6.1 Τρόποι μελέτης της συμπεριφοράς των δύο μοντέλων της αρχιτεκτονικής marketplace

Για κάθε ένα από τα δύο διαφορετικά μοντέλα επικοινωνίας που περιγράφηκαν παραπάνω, μελετήθηκε η χρονική τους απόκριση, για πέντε διαφορετικές εισόδους.

Ως είσοδοι, επιλέχθηκαν 5 διαφορετικοί συνδυασμοί προδιαγραφών του χρήστη (Input 1-5), καθένας από τους οποίους οδηγεί στην παραγωγή, επεξεργασία και ανταλλαγή διαφορετικού συνολικού αριθμού μηνυμάτων. Αρχικά, δοκιμάστηκε ένας συνδυασμός που παράγει λίγα σχετικά μηνύματα και στον τελευταίο συνδυασμό ο αριθμός μηνυμάτων ήταν αρκετά μεγάλος. Από τον πρώτο συνδυασμό προδιαγραφών χρήστη προκύπτουν 1464 μηνύματα, από τον δεύτερο συνδυασμό 3258 μηνύματα, από τον τρίτο συνδυασμό 5323 μηνύματα, από τον τέταρτο συνδυασμό 10975 και από τον πέμπτο συνδυασμό 38010 μηνύματα.

Το μεγαλύτερο ποσοστό των μηνυμάτων αυτών, παράγεται κάθε φορά από το 2ο στάδιο (Business Phase), το δεύτερο μεγαλύτερο ποσοστό παράγεται από το 3ο στάδιο (Price Phase) και το μικρότερο ποσοστό παράγεται από το 1ο στάδιο (Technical Phase). Αυτό είναι λογικό, αφού για κάθε μήνυμα που παράγει το 1ο στάδιο (Technical Phase), το 2ο στάδιο (Business Phase) παράγει πολλαπλάσιο αριθμό μηνυμάτων, ενώ το 3ο στάδιο (Price Phase) δεν παράγει καινούρια μηνύματα, απλώς ανανεώνει το περιεχόμενο αυτών που κατανάλωσε και άλλα από αυτά τα προωθεί στο στάδιο 4 (Resolution Phase), άλλα τα απορρίπτει.

Δεδομένου ότι το μεγαλύτερο ποσοστό μηνυμάτων παράγεται από το 2ο στάδιο (Business Phase), προκύπτει ότι το μεγαλύτερο φορτίο στην επικοινωνία μεταξύ των διαφόρων σταδίων δημιουργείται πάντα, ανάμεσα στο 2ο (Business Phase) και το 3ο στάδιο (Price Phase), αφού ο όγκος μηνυμάτων που παράγεται από το 2ο στάδιο (Business Phase), είναι πολύ μεγάλος (Big Data) και δεν μπορεί να τον επεξεργαστεί γρήγορα το 3ο στάδιο (Price Phase). Για τον λόγο αυτό, μελετήθηκε η συμπεριφορά των δύο διαφορετικών μοντέλων, σε τρεις διαφορετικές περιπτώσεις. Στην πρώτη περίπτωση, αφέθηκε το κάθε μοντέλο να τρέξει ως έχει, χωρίς να αλλάξει κάποια παράμετρός της, ενώ στις επόμενες δύο περιπτώσεις έγινε προσπάθεια να περιορίσουμε το φορτίο που δημιουργείται ανάμεσα στα στάδια 2 (Business Phase) και 3 (Price Phase), με δύο διαφορετικούς τρόπους. Στην δεύτερη περίπτωση, προστέθηκε χρονοκαθυστέρηση πριν την παραγωγή των μηνυμάτων από το 1ο (Technical Phase) και το 2ο στάδιο (Business Phase), προκειμένου να δοκιμαστεί η περίπτωση όπου τα στάδια αυτά είχαν πιο πολύπλοκο αλγόριθμο και ήθελαν περισσότερο χρόνο για να εκτελεστούν. Η μοντελοποίηση αυτής της περίπτωσης έγινε με τη χρήση μίας ρουτίνας που προσφέρει τη δυνατότητα προσομοίωσης της χρονικής απόκρισης του συγκεκριμένου αλγορίθμου. Μέσω της ρουτίνας αυτής προστέθηκε χρονική καθυστέρηση 700 msec πριν από την παραγωγή μηνυμάτων του 1ου σταδίου(Technical Phase) και 250 msec πριν την παραγωγή μηνυμάτων του 2ου σταδίου (Business Phase). Στην τρίτη περίπτωση, έγινε δοκιμή αντιμετώπισης του μεγάλου φορτίου, μέσω της παραλληλοποίησης της επεξεργασίας των μηνυμάτων. Χρησιμοποιήθηκαν ουσιαστικά περισσότερα του ενός

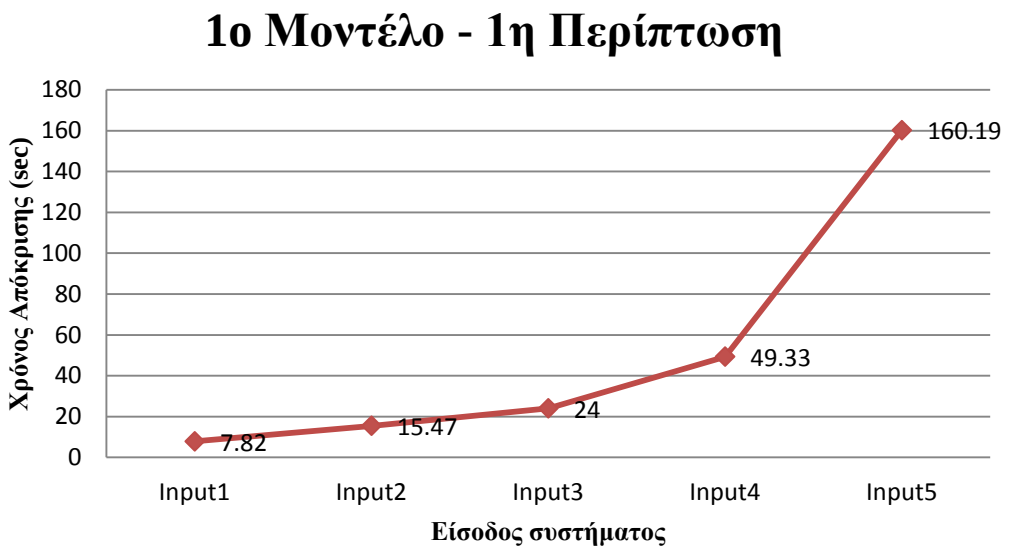
νήματα, τα οποία κατανάλωναν παράλληλα τα μηνύματα και μείωναν το φορτίο, διαμοιράζοντάς το ουσιαστικά σε περισσότερους του ενός καταναλωτές. Η ActiveMQ δίνει τη δυνατότητα χρήσης περισσοτέρων του ενός καταναλωτών για μια ουρά, μέσω μιας παραμέτρου που ονομάζεται "Consumers". Για να εξεταστεί λοιπόν η τρίτη αυτή περίπτωση, χρησιμοποιήθηκαν 10 Consumers αντί για 1, που χρησιμοποιήθηκαν στις άλλες δύο περιπτώσεις.

6.2 Αποτελέσματα μελέτης του πρώτου μοντέλου

Στο μοντέλο αυτό, χρησιμοποιήθηκαν τρεις ουρές (queues): η queue1, η οποία συνδέει το 1ο (Technical Phase) με το 2ο στάδιο (Business Phase), η queue2, η οποία συνδέει το 2ο (Business Phase) με το 3ο στάδιο(Price Phase) και η queue3, η οποία συνδέει το 3ο (Price Phase) με το 4ο και τελευταίο στάδιο (Resolution Phase).

Περίπτωση 1η: Απλή εκτέλεση

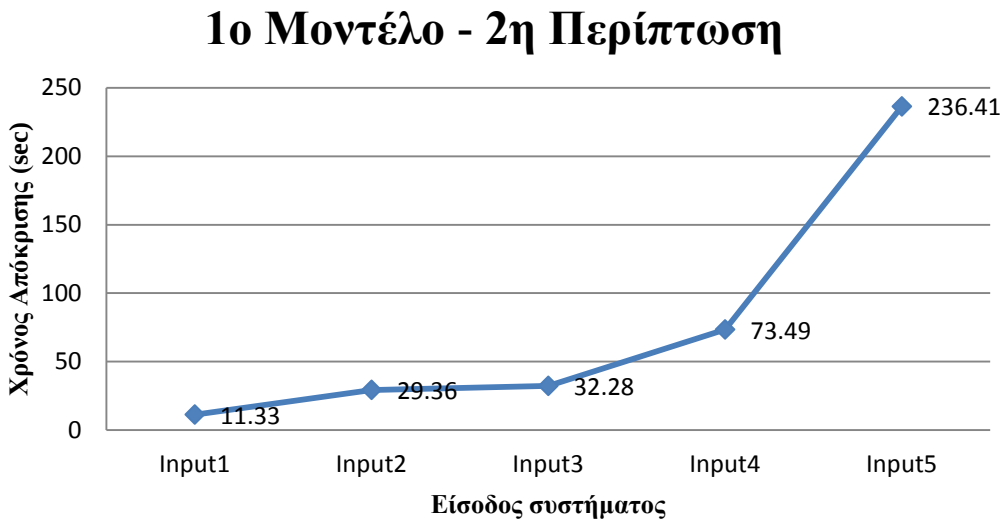
Κατά την απλή εκτέλεση, όπως ήταν αναμενόμενο και με βάση τα όσα ειπώθηκαν παραπάνω, παρατηρήθηκε αυξημένο φορτίο στην queue2, στην οποία εισάγονται τα μηνύματα που παράγονται από το 2ο στάδιο (Business Phase). Ακόμη, καθώς ο συνολικός αριθμός μηνυμάτων αυξανόταν (δηλαδή, από το Input 1 προς το Input 5), αυξανόταν τόσο το φορτίο στην queue2 όσο και ο συνολικός χρόνος εκτέλεσης.



Διάγραμμα 1: Χρονική απόκριση του πρώτου μοντέλου για κάθε είσοδο, στην πρώτη περίπτωση

Περίπτωση 2η: Εκτέλεση με χρήση της ρουτίνας χρονοκαθυστέρησης

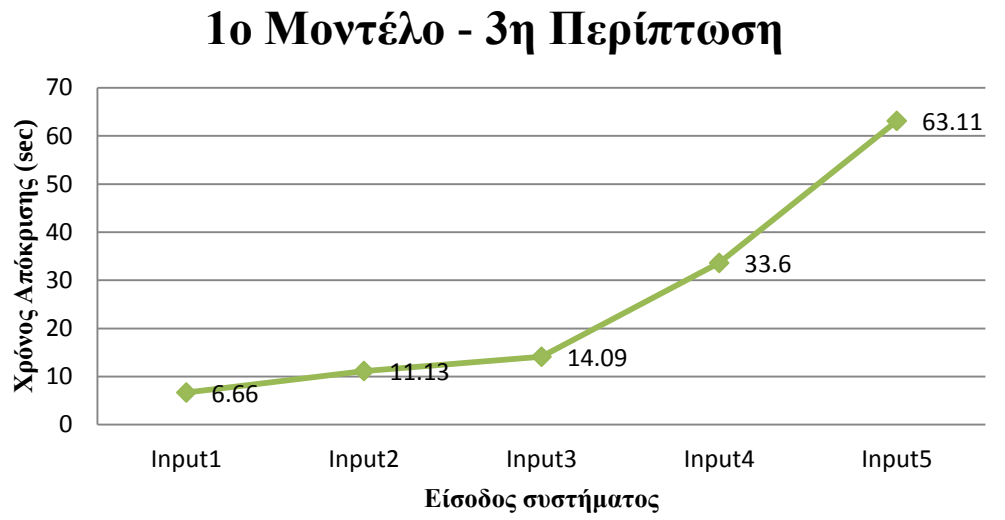
Σε αυτήν την περίπτωση, προστέθηκε με χρήση της μεθόδου delay της Java, χρονοκαθυστέρηση 700 msec πριν την εκτέλεση του 1ου σταδίου(Technical Phase) και 250 msec πριν την εκτέλεση του 2ου σταδίου (Business Phase), με στόχο την ελάττωση του ρυθμού παραγωγής μηνυμάτων από το 2ο στάδιο (Business Phase) και την ελάττωση τελικά του φορτίου που επωμίζεται η queue2.



Διάγραμμα 2: Χρονική απόκριση του πρώτου μοντέλου για κάθε είσοδο, στην δεύτερη περίπτωση

Περίπτωση 3η: Εκτέλεση με παραπάνω καταναλωτές (Consumers)

Στην περίπτωση αυτή, χρησιμοποιήθηκαν 10 καταναλωτές (Consumers) στην queue2, προκειμένου να αντιμετωπιστεί το παραπάνω φορτίο ανάμεσα στα στάδια 2 (Business Phase) και 3 (Price Phase). Πράγματι παρατηρείται ότι ο συνολικός χρόνος εκτέλεσης, μειώθηκε αισθητά χάρη στην χρήση των επιπλέον καταναλωτών (Consumers).

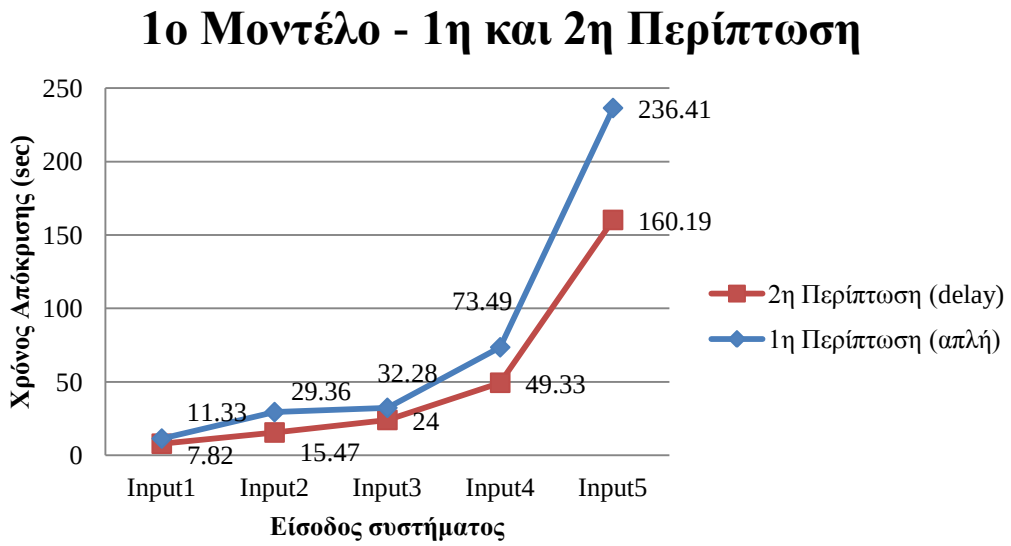


Διάγραμμα 3: Χρονική απόκριση του πρώτου μοντέλου για κάθε είσοδο, στην τρίτη περίπτωση

6.3 Σύγκριση 2ης και 3ης περίπτωσης με την 1η περίπτωση του πρώτου μοντέλου

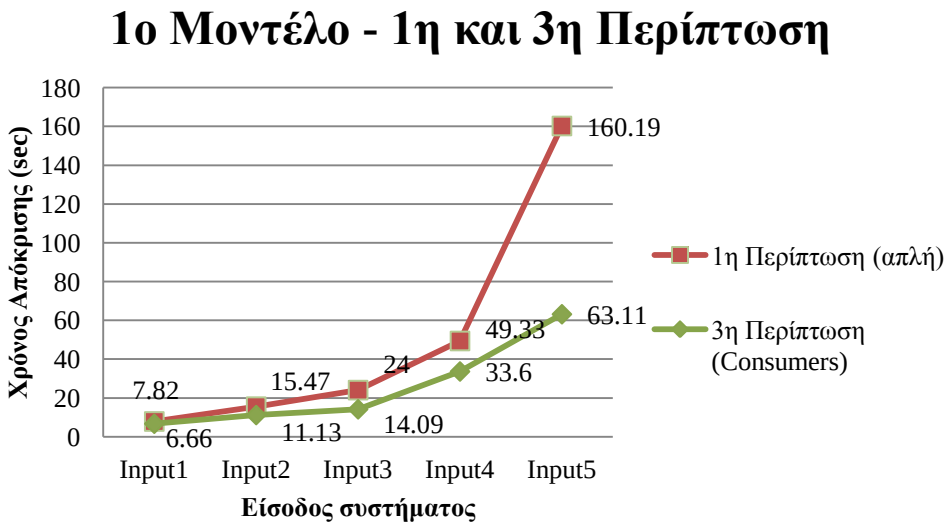
Στο παρακάτω διάγραμμα (Διάγραμμα 4), δίνονται μαζί τα αποτελέσματα του πρώτου μοντέλου για την πρώτη και την δεύτερη περίπτωση. Κατά την εκτέλεση της δεύτερης

περίπτωσης, παρατηρήθηκε ότι είχε αυξημένο φορτίο αρχικά η queue1, γεγονός το οποίο ελάττωσε αρχικά το φορτίο της queue2, αλλά μόλις άδειασε η queue1, η queue2 απέκτησε και πάλι αυξημένο φορτίο.



Διάγραμμα 4: Χρονική απόκριση του πρώτου μοντέλου για κάθε είσοδο, στην πρώτη και στην δεύτερη περίπτωση

Στη συνέχεια, δίνονται στο ίδιο διάγραμμα (Διάγραμμα 5), τα αποτελέσματα της πρώτης και της τρίτης περίπτωσης. Είναι εμφανές, ότι η προσθήκη επιπλέον καταναλωτών (Consumers), μείωσε αισθητά τον χρόνο λειτουργίας, ειδικά για μεγάλο όγκο μηνυμάτων, αφού στην είσοδο με τον μεγαλύτερο παραγόμενο αριθμό μηνυμάτων, ο χρόνος μειώθηκε παραπάνω από το μισό.



Διάγραμμα 5: Χρονική απόκριση του δευτέρου μοντέλου για κάθε είσοδο, στην πρώτη και την τρίτη περίπτωση

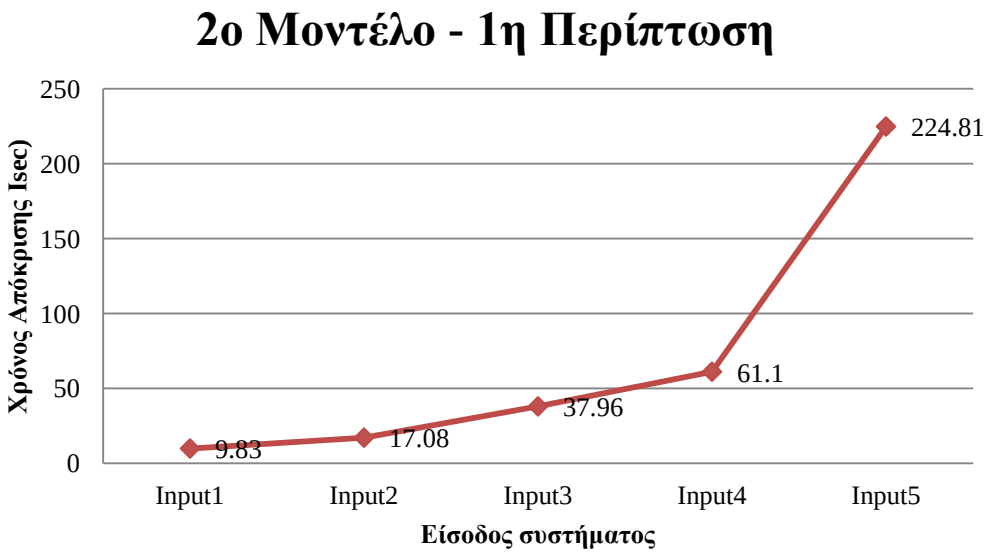
Προκύπτει λοιπόν το συμπέρασμα ότι για την αντιμετώπιση του μεγάλου φορτίου που δημιουργείται λόγω του μεγάλου όγκου δεδομένων που παράγεται, άρα και κατ'επέκταση για την αντιμετώπιση του προβλήματος που δημιουργείται στην αρχιτεκτονική του marketplace λόγω των Big Data, πιο αποτελεσματικός τρόπος προσέγγισης είναι η αύξηση των καταναλωτών (Consumers) των ουρών μηνυμάτων, δηλαδή η παραλληλοποίηση της επεξεργασίας των μηνυμάτων.

6.4 Αποτελέσματα μελέτης του δευτέρου μοντέλου

Στο μοντέλο αυτό, χρησιμοποιήθηκε μια ουρά (common) κοινή για όλα τα στάδια του συστήματος. Για να μπορέσει ωστόσο να γίνει ο κατάλληλος διαχωρισμός των μηνυμάτων στα επιμέρους στάδια και να γίνεται αποστολή των μηνυμάτων στον σωστό κάθε φορά παραλήπτη-στάδιο, ανέλαβε τον ρόλο του διανομέα των μηνυμάτων, το Camel, το οποίο ανάλογα με το ID του κάθε μηνύματος, με τιμές από 1 έως 4, έστελνε το μήνυμα στο αντίστοιχο στάδιο.

Περίπτωση 1η: Απλή εκτέλεση

Στην περίπτωση αυτή, υπήρχε αυξημένο φορτίο στην ουρά, αλλά επειδή η ουρά ήταν κοινή για όλα τα στάδια, δεν φαίνεται μέσω της κονσόλας του ActiveMQ (ActiveMQ Console), πώς κατανέμεται το σύνολο των μηνυμάτων στα επιμέρους στάδια και που τελικά δημιουργείται το μεγαλύτερο φορτίο.

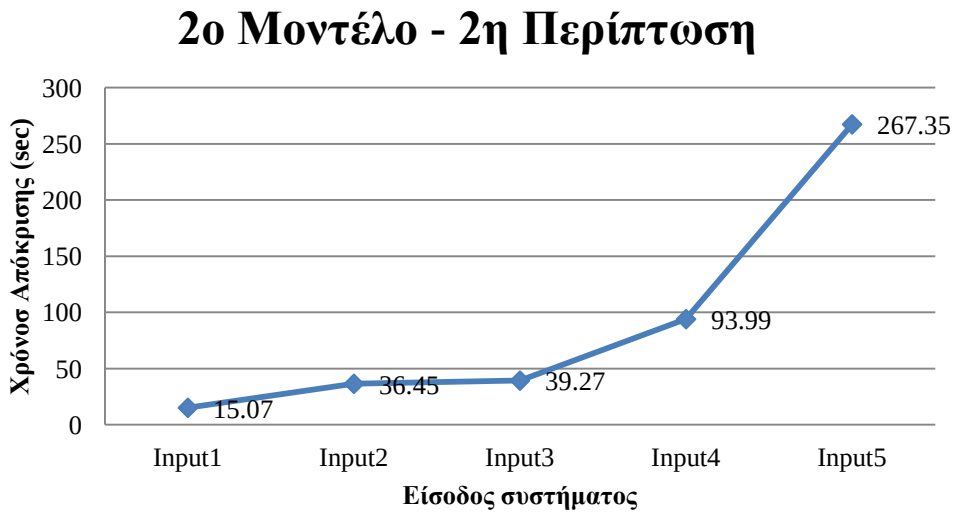


Διάγραμμα 6: Χρονική απόκριση του δευτέρου μοντέλου για κάθε είσοδο, στην πρώτη περίπτωση

Παρατηρείται ότι και εδώ, όπως είναι λογικό, όσο αυξάνεται ο αριθμός των συνολικών μηνυμάτων, τόσο αυξάνεται και ο χρόνος που απαιτείται, προκειμένου να ολοκληρωθεί η εκτέλεση του προγράμματος.

Περίπτωση 2η: Εκτέλεση με χρονοκαθυστέρηση

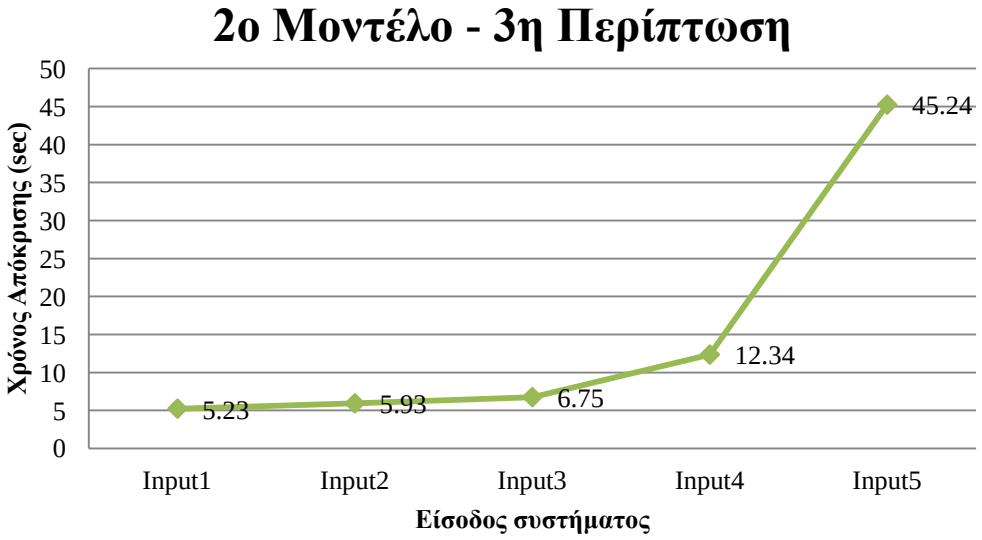
Σε αυτήν την περίπτωση, προστέθηκε με χρήση της μεθόδου sleep της Java, χρονοκαθυστέρηση 700 msec πριν την εκτέλεση του 1ου σταδίου(Technical Phase) και 250 msec πριν την εκτέλεση του 2ου σταδίου (Business Phase), με στόχο την ελάττωση του ρυθμού παραγωγής μηνυμάτων από το 2ο στάδιο (Business Phase) και την ελάττωση τελικά του μεγάλου φορτίου που αντιμετωπίζει η κοινή ουρά.



Διάγραμμα 7: Χρονική απόκριση του δευτέρου μοντέλου για κάθε είσοδο, στην δεύτερη περίπτωση

Περίπτωση 3η: Εκτέλεση με παραπάνω καταναλωτές (Consumers)

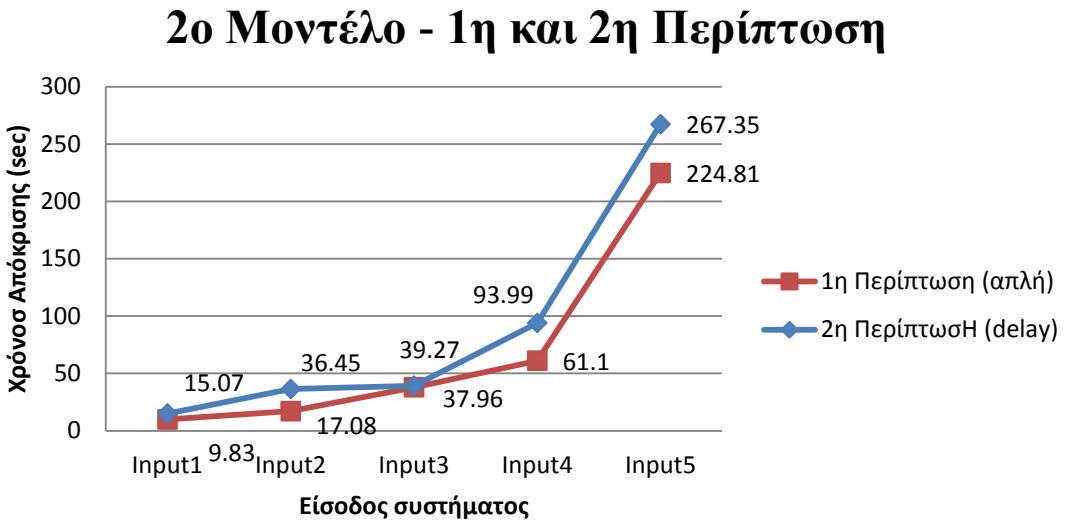
Στην περίπτωση αυτή, αυξήθηκε ο αριθμός των καταναλωτών (Consumers) της ουράς (common) προκειμένου να αντιμετωπιστεί το μεγάλο φορτίο. Ωστόσο, στην περίπτωση αυτή, σε αντίθεση με την περίπτωση του πρώτου μοντέλου, η παραλληλοποίηση που γίνεται χάρη στους παραπάνω καταναλωτές, αφορά τα μηνύματα ανάμεσα σε όλα τα στάδια και όχι μόνο τα μηνύματα ανάμεσα στα στάδια 2 (Business Phase) και 3 (Price Phase).



Διάγραμμα 8: Χρονική απόκριση του δευτέρου μοντέλου για κάθε είσοδο, στην τρίτη περίπτωση

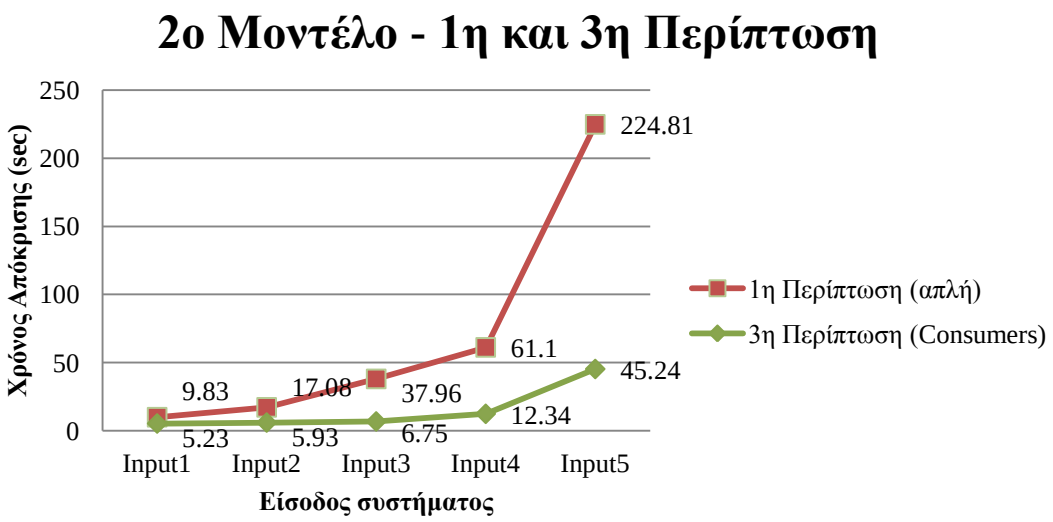
6.5 Σύγκριση 2ης και 3ης περίπτωσης με την 1η περίπτωση του δευτέρου μοντέλου

Στη συνέχεια παρουσιάζονται στο ίδιο διάγραμμα (Διάγραμμα 9), τα αποτελέσματα του δευτέρου μοντέλου, στην πρώτη και την δεύτερη περίπτωση. Παρατηρείται ότι η προσθήκη χρονοκαθυστέρησης πριν από την εκτέλεση των σταδίων που παράγουν τα περισσότερα μηνύματα, δεν βελτιώνει ιδιαίτερα το πρόβλημα, αφού απλά αυξήθηκε ο χρόνος εκτέλεσης, χωρίς όμως να ελαφρύνει ουσιαστικά το φορτίο.



Διάγραμμα 9: Χρονική απόκριση του δευτέρου μοντέλου για κάθε είσοδο, στην πρώτη και την δεύτερη περίπτωση

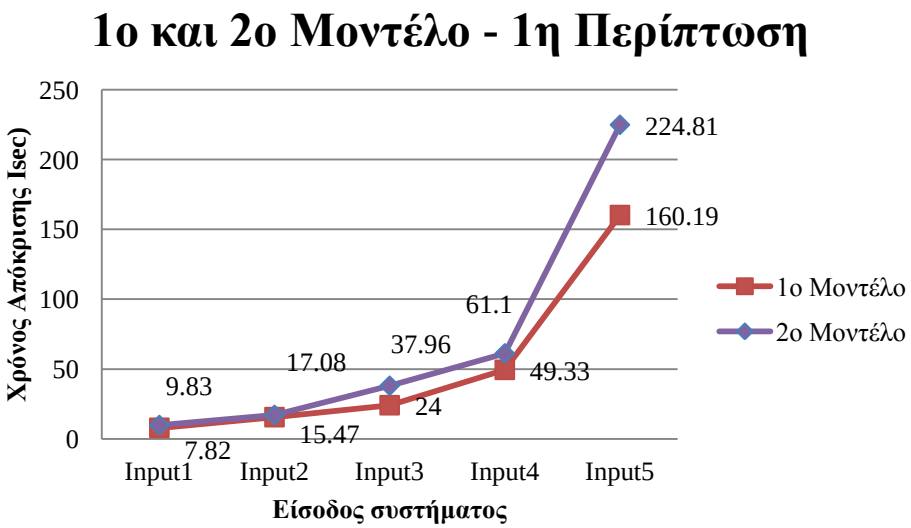
Στη συνέχεια, στο Διάγραμμα 10 παρουσιάζονται μαζί τα αποτελέσματα της πρώτης και της τρίτης περίπτωσης, όπου παρατηρείται ότι ο χρόνος απόκρισης του συστήματος μειώθηκε σε πολύ μεγάλο βαθμό και μάλιστα όσο μεγαλύτερος ήταν ο συνολικός αριθμός μηνυμάτων, τόσο πιο μεγάλη ήταν η μείωση του χρόνου επεξεργασίας. Συγκεκριμένα, στην περίπτωση του Input 5, που παράγει τον μεγαλύτερο αριθμό μηνυμάτων, ο χρόνος μειώθηκε στο ένα πέμπτο περίπου από τον αντίστοιχο χρόνο της απλής εκτέλεσης. Συνεπώς, όπως και στην περίπτωση του πρώτου μοντέλου, η παραλληλοποίηση της επεξεργασίας, μέσω της αύξησης των καταναλωτών, προκύπτει να είναι καλύτερος τρόπος αντιμετώπισης του αυξημένου φορτίου μηνυμάτων κατά την επικοινωνία των σταδίων μεταξύ τους, συγκριτικά με την προσθήκη χρονοκαθυστέρησης.



Διάγραμμα 10: Χρονική απόκριση του δευτέρου μοντέλου για κάθε είσοδο, στην πρώτη και την τρίτη περίπτωση

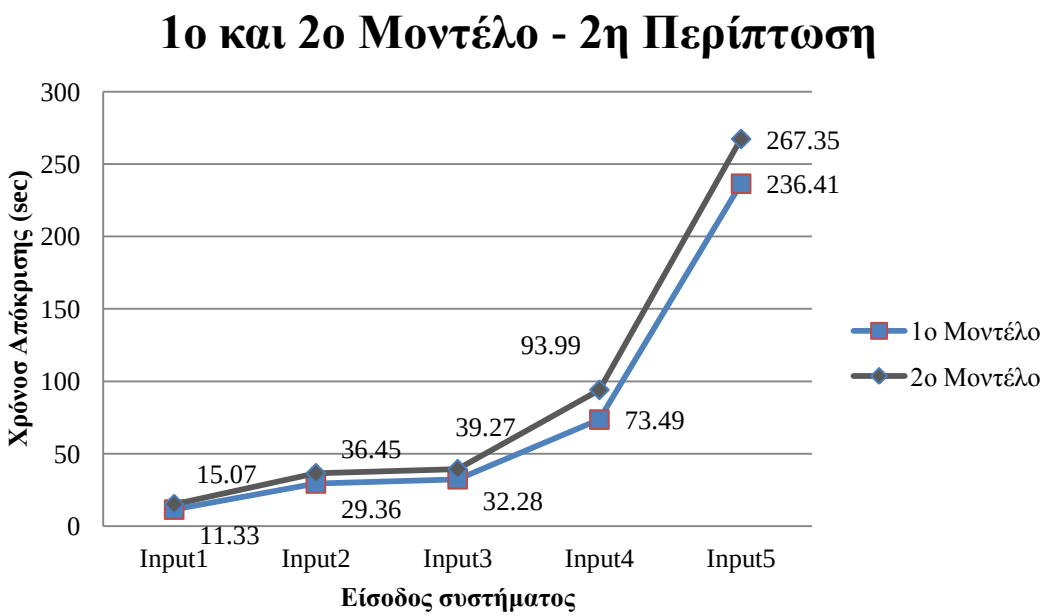
6.6 Σύγκριση πρώτου και δευτέρου μοντέλου σε όλες τις περιπτώσεις

Στην περίπτωση της απλής εκτέλεσης, συμπεραίνεται από το παρακάτω διάγραμμα (Διάγραμμα 11), ότι η χρήση των ξεχωριστών ουρών έχει σταθερά αρκετά μικρότερο χρόνο εκτέλεσης, συγκριτικά με το δεύτερο μοντέλο, το οποίο χρησιμοποιεί την μία κοινή ουρά.



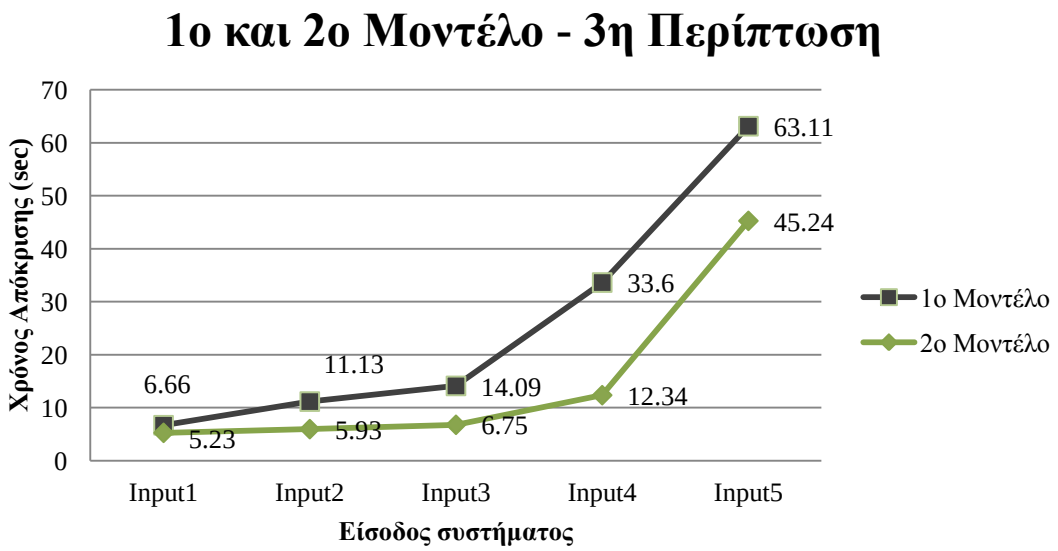
Διάγραμμα 11: Χρονική απόκριση του πρώτου και του δευτέρου μοντέλου για κάθε είσοδο, στην πρώτη περίπτωση

Ακολουθεί το διάγραμμα με τα αποτελέσματα και των δύο μοντέλων στην δεύτερη περίπτωση (Διάγραμμα 12), αυτήν της χρονοκαθυστέρησης. Είναι εμφανές ότι τα δύο μοντέλα, δεν έχουν μεγάλη διαφορά στην χρονική τους απόκριση. Ωστόσο, παρατηρείται ότι έστω και με μικρή χρονική διαφορά, υπερτερεί και πάλι το πρώτο μοντέλο, με τις ξεχωριστές ουρές.



Διάγραμμα 12: Χρονική απόκριση του πρώτου και του δευτέρου μοντέλου για κάθε είσοδο, στην δεύτερη περίπτωση

Τέλος, στο διάγραμμα που ακολουθεί (Διάγραμμα 13), παρουσιάζονται μαζί τα αποτελέσματα και των δύο μοντέλων στην τρίτη περίπτωση, δηλαδή στην περίπτωση του αυξημένου αριθμού καταναλωτών. Παρατηρείται ωστόσο, ότι σε αυτήν την περίπτωση, το δεύτερο μοντέλο, προκύπτει ως καλύτερη λύση χρονικά, από αυτήν του πρώτου μοντέλου. Το αποτέλεσμα αυτό, είναι λογικό, δεδομένου ότι μέσω της αύξησης των καταναλωτών στην κοινή ουρά (common), γίνεται παραλληλοποίηση της επεξεργασίας των μηνυμάτων, σε όλα τα στάδια του συστήματος και όχι μόνο στην επικοινωνία ανάμεσα στο 2ο (Business Phase) και το 3ο (Price Phase). Επιπλέον, το γεγονός ότι όλα τα μηνύματα περνούν από την ίδια ουρά, σημαίνει ότι η σειρά επεξεργασίας των μηνυμάτων ανακατεύεται περισσότερο απ'ότι στο πρώτο μοντέλο, και το ανακάτεμα αυτό, δημιουργεί τα απαραίτητα χρονικά κενά ανάμεσα στην παραγωγή των μηνυμάτων του 2ου σταδίου (Business Phase), τα οποία έγινε προσπάθεια να δημιουργηθούν μέσω της χρονοκαθυστέρησης, και τα οποία είναι απαραίτητα για την αποφυγή δημιουργίας μεγάλου φορτίου μηνυμάτων σε οποιοδήποτε σημείο των καναλιών επικοινωνίας που δημιουργούν οι ουρές μηνυμάτων.



Διάγραμμα 13: Χρονική απόκριση του πρώτου και του δευτέρου μοντέλου για κάθε είσοδο, στην πρώτη περίπτωση

Κεφάλαιο 7:Συμπεράσματα

Στην παρούσα διπλωματική εργασία, υλοποιήθηκε ένα σύστημα που προσομειώνει μια αρχιτεκτονική για πλατφόρμες-αγορές (marketplaces) υπηρεσιών Cloud και μελετήθηκε η συμπεριφορά της κατά την χρήση ουρών μηνυμάτων (message queues), με δύο διαφορετικούς τρόπους-μοντέλα, για την επικοινωνία των ενδιαμέσων σταδίων του. Στο πρώτο μοντέλο χρησιμοποιήθηκαν ξεχωριστές ουρές ανάμεσα στα στάδια του συστήματος, ενώ στο δεύτερο μοντέλο η επικοινωνία όλων των σταδίων μεταξύ τους, έγινε μέσω μιας κοινής ουράς. Ακόμη, εξετάστηκε η συμπεριφορά καθενός από τα δύο αυτά μοντέλα, με πέντε διαφορετικές εισόδους και σε τρεις διαφορετικές περιπτώσεις συμπεριφοράς του συστήματος. Η πρώτη περίπτωση αφορούσε την απλή λειτουργία του συστήματος, η δεύτερη περίπτωση αφορούσε την λειτουργία του συστήματος κάτω από πιο πολύπλοκο αλγόριθμο και η τρίτη περίπτωση αφορούσε την λειτουργία του συστήματος με πολλούς καταναλωτές.

Σύμφωνα με τα αποτελέσματα που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, προκύπτουν τα εξής συμπεράσματα: σε περιπτώσεις μικρού ή μετρίου όγκου μηνυμάτων προς επεξεργασία και τα δύο μοντέλα λειτουργούν αρκετά καλά, με το πρώτο μοντέλο να υπερτερεί χρονικά, χωρίς ωστόσο σημαντική διαφορά. Σε μεγάλο όγκο μηνυμάτων όμως, υπάρχουν σημαντικές αλλαγές στον τελικό απαιτούμενο χρόνο του συστήματος marketplace, με το πρώτο μοντέλο να είναι καλύτερο με διαφορά.

Ταυτόχρονα, από τους δύο τρόπους που χρησιμοποιήθηκαν ως μέσο περιορισμού του αυξημένου φορτίου μηνυμάτων που παρεμποδίζει την επικοινωνία των σταδίων του συστήματος και που οδηγεί στην αύξηση του τελικού χρόνου απόκρισης του συστήματος, δηλαδή ανάμεσα στην χρήση χρονοκαθυστέρησης και στην αύξηση των καταναλωτών (Consumers) των ουρών μηνυμάτων, καλύτερος αποδείχτηκε ο δεύτερος, ο οποίος μείωσε και στα δύο μοντέλα πολύ τον τελικό χρόνο επεξεργασίας όλων των μηνυμάτων, μέσω της παραλληλοποίησής της.

Συνδυάζοντας τα παραπάνω συμπεράσματα, προκύπτει ότι για την υλοποίηση ενός αποδοτικού marketplace υπηρεσιών Cloud, το οποίο θα αντιμετωπίζει αποτελεσματικά τα τεράστια φορτία μηνυμάτων που δημιουργούνται λόγω των Big Data που επεξεργάζεται σαν σύστημα, βέλτιστη είναι η χρήση κοινής ουράς με παραπάνω καταναλωτές (Consumers) για την επικοινωνία των επιμέρους στοιχείων της αρχιτεκτονικής του συστήματος, αφού επιτυγχάνεται η απαραίτητη παραλληλοποίηση, που επιλύει τα προβλήματα που δημιουργούνται λόγω Big Data.

7.1 Πιθανές μελλοντικές προσθήκες

Το σύστημα που μελετήθηκε επιδέχεται περαιτέρω μελλοντικές προσθήκες και βελτιώσεις, προκειμένου να γίνει ακόμη πιο αποδοτικό και γρήγορο. Καταρχάς, μπορεί να προστεθεί μία βάση δεδομένων, στην οποία θα είναι αποθηκευμένες όλες οι απαραίτητες πληροφορίες για τις επιμέρους υπηρεσίες και του παρόχους αυτών, προκειμένου να ελαχιστοποιηθεί ο χρόνος προσπέλασης αυτών. Επιπλέον, μπορεί να βελτιωθεί ο τρόπος με τον οποίο δίνεται η είσοδος στο σύστημα, μέσω μιας πλατφόρμας GUI, την οποία θα μπορεί να

χρησιμοποιήει ο χρήστης για να δώσει τις προδιαγραφές που επιθυμεί. Τέλος, μέσω της ίδιας πλατφόρμας, μπορεί να δίνεται μετά το πέρας της επεξεργασίας, η έξοδος τους συστήματος, δηλαδή ο τελικός συνδυασμός που προτείνεται ως βέλτιστη λύση με βάση τις προδιαγραφές που έδωσε ο χρήστης.

Βιβλιογραφία

[1] Peter Mell, Timothy Grance (2011 Σεπτέμβριος). *The NIST Definition of Cloud Computing*. NIST Special Publication 800-145, National Institute of Standards and Technology

[2] Sonsinsky, B. (2011). *Cloud Computing Bible.* Indianapolis, Indiana: Wiley Publishing, Inc.

[3] *Wikipedia*. (2016, Φεβρουάριος). Από: https://en.wikipedia.org/wiki/Cloud_computing_architecture

[4]Nathan Marz, J. W. (2012). *Big Data, Principles and best paractices of scalable realtime data systems,* Manning Pubilcations.

[5] Gil Press. *"12 Big Data Definitions: What's Yours?"*. Forbes. Από: http://www.forbes.com/sites/gilpress/2014/09/03/12-big-data-definitions-whats-yours/#42479e6821a9

[6] *Oxford English Dictionary.* Oxford University Press.

[7] *Wikipedia*. (2016, Φεβρουάριος). Από: https://en.wikipedia.org/wiki/Big_data

[8] McKinsey&Co. James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Angela Hung Byers. (2011). *Big data: The next frontier for innovation, competition, and productivity.* Μάιος.

[9] Dataconomy. *Understanding Big Data: The Seven V's*. Από: *http://dataconomy.com/seven-vs-big-data/*

[10] Google Inc. Jeffrey Dean, Sanjay Ghemawat (2004). *MapReduce: Simplified Data Processing on Large Clusters.* Από: http://static.googleusercontent.com/media/research.google.com/el//archive/mapreduce-osdi04.pdf

[11] IBM Map Reduce. Από: https://www-01.ibm.com/software/data/infosphere/hadoop/mapreduce/

[12] Wikipedia "Queue" and "Message Queue". Από: https://en.wikipedia.org/wiki/Queue_%28abstract_data_type%29 https://en.wikipedia.org/wiki/Message_queue

[13] Bruce SNyder, Dejan Bosanac, Rob Davies (2009). *ActiveMQ in Action*, Manning Publications

[14] RabbitMQ Tutorials. Από: https://www.rabbitmq.com/getstarted.html

[15] Iron.io. Από: http://www.iron.io/top-10-uses-for-message-queue/

[16] Data Science. Από: http://www.kuntalganguly.com/2014/08/message-queue-comparison.html

[17] Andreas Menychtas, Jürgen Vogel, Andrea Giessmann, Anna Gatzioura, Sergio Garcia Gomez, Vrettos Moulos, Frederic Junker, Mathias Müller, Dimosthenis Kyriazis, Katarina Stanoevska-Slabeva, Theodora Varvarigou (Μάρτιος 2014). *4CaaSt marketplace: An advanced business environment for trading cloud services,Future Generation Computer Systems*

Παράρτημα Ι

(https://bitbucket.org/dkmsgroup/ashlynn)

Η κλάση main μέσω της οποίας τρέχουμε το πρόγραμμα (Main.java)

```
package edu.gr.ntua.sylvia;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

	public static void main(String[] args) {
		ApplicationContext appContext = new
ClassPathXmlApplicationContext("META-INF/applicationContext.xml");
	}
}
```

Κώδικας Camel για το πρώτο μοντέλο με τις ξεχωριστές ουρές (Architecture1.java)

```
package edu.gr.ntua.sylvia.camel;
import javax.annotation.Resource;
import org.apache.camel.spring.SpringRouteBuilder;
import org.springframework.stereotype.Component;
import edu.gr.ntua.sylvia.process.ResolutionPhase;
import edu.gr.ntua.sylvia.process.BusinessPhase;
import edu.gr.ntua.sylvia.process.TechnicalPhase;
import edu.gr.ntua.sylvia.process.PricePhase;
import edu.gr.ntua.sylvia.process.CreateObject;

@Component
public class Architecture1 extends SpringRouteBuilder {

	private static final String START_PATH =
"C:/Users/Sylvie/Desktop/camelRouteStart";

	@Resource
	private CreateObject createObject;

	@Resource
	private TechnicalPhase technicalPhase;

	@Resource
	private BusinessPhase businessPhase;

	@Resource
	private PricePhase pricePhase;

	@Resource
	private ResolutionPhase resolP;

	@Override
	public void configure() throws Exception {

		from("file://" + START_PATH + "?include=.*.xml")
			.bean(createObject, "unmarshalXml")
			.to("direct:technicalPhase");

		from ("direct:technicalPhase")
```

```
//                  .delay(700)
                    .bean(technicalPhase, "TechnicalRequest");

              from("activemq:queue1?concurrentConsumers=1")
//                  .delay(250)
                    .bean(businessPhase, "BusinessRequest");

              from("activemq:queue2?concurrentConsumers=1")
//            from("activemq:queue2?concurrentConsumers=10")
                    .bean(pricePhase, "PriceRequest");

               from("activemq:queue3?concurrentConsumers=1")
                    .bean(resolP, "Resolution");
       }
}
```

Κώδικας Camel για το δεύτερο μοντέλο με την κοινή ουρά (Architecture2.java)

```
package edu.gr.ntua.sylvia.camel;
import javax.annotation.Resource;
import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.camel.spring.SpringRouteBuilder;
import org.springframework.stereotype.Component;
import edu.gr.ntua.sylvia.process.ResolutionPhase;
import edu.gr.ntua.sylvia.process.BusinessPhase;
import edu.gr.ntua.sylvia.process.TechnicalPhase;
import edu.gr.ntua.sylvia.process.PricePhase;
import edu.gr.ntua.sylvia.process.CreateObject;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Component
public class Architecture2 extends SpringRouteBuilder {

      private static final String START_PATH_COMMON =
"C:/Users/Sylvie/Desktop/camelRouteStartArchitecture2";

      @Resource
      private CreateObject createObjectCommon;
      @Resource
      private TechnicalPhase technicalPhaseCommon;
      @Resource
      private BusinessPhase businessPhaseCommon;
      @Resource
      private PricePhase pricePhaseCommon;
      @Resource
      private ResolutionPhase resolPCommon;
      @Override
      public void configure() throws Exception {

            from("file://" + START_PATH_COMMON + "?include=.*.xml")
            .bean(createObjectCommon, "unmarshalXml")
            .to("activemq:queue:common");

            from("activemq:queue:common?concurrentConsumers=1")
                  .process(new Processor() {

                  @Override
                  public void process(Exchange ex) throws Exception {
```

```
                        UserSystem userSystem =
ex.getIn().getBody(UserSystem.class);
                        Integer msgId = userSystem.getId();
                        switch (msgId) {
                        case 1:
//                              Thread.sleep(700);

      technicalPhaseCommon.CommonTechnicalRequest(userSystem);
                              break;
                        case 2:
//                              Thread.sleep(250);
                        businessPhaseCommon.CommonBusinessRequest(userSystem);
                              break;
                        case 3:
                              pricePhaseCommon.CommonPriceRequest(userSystem);
                              break;
                        case 4:
                              resolPCommon.CommonResolution(userSystem);
                              break;
                        default:
                              System.out.println("ERROR!!");
                        }
                  }
            });
      }
}
```

Η διαδικασία (process)μέσω της οποίας το Camel μετατρέπει το αρχείο εισόδου από XML σε Java Object (CreateObject.java)

```
package edu.gr.ntua.sylvia.process;
import java.io.File;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import org.springframework.stereotype.Component;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Component
public class CreateObject {

      public UserSystem unmarshalXml(File xmlFile) throws JAXBException {
            UserSystem system = new UserSystem();
            JAXBContext jc = JAXBContext.newInstance(UserSystem.class);
            Unmarshaller u = jc.createUnmarshaller();
            system = (UserSystem) u.unmarshal(xmlFile);
            return system;
      }
}
```

Το μοντέλο μέσω του οποίου γίνεται η κατασκευή του Java Object από το αρχείο εισόδου XML αποτελείται από τις παρακάτω κλάσεις:

1)Η βασική κλάση στην οποία βασίζεται η κατασκευή (UserSystem.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
```

```java
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlType
@XmlRootElement(name = "UserSystem")
@XmlAccessorType(XmlAccessType.FIELD)
public class UserSystem implements Serializable, Cloneable {
	private static final long serialVersionUID = -3557946853704538043L;

	@XmlElement(name = "Id")
	private Integer id;
	@XmlElement(name = "UserRequirements")
	private UserReqClass userRequirements;
	@XmlElement(name = "Technical")
	private TechnicalClass technical;{
	if (technical==null) {
		technical= new TechnicalClass();
		}
	}

	@XmlElement(name = "Business")
	private BusinessClass business;{
		if (business==null){
			business= new BusinessClass();
		}
	}

	@XmlElement(name = "TotalAvailability")
	private Integer totalAvailability;{
		if (totalAvailability==null){
			totalAvailability=0;
		}
	}

	@XmlElement(name = "TotalSecurity")
	private Integer totalSecurity;{
		if (totalSecurity==null){
			totalSecurity=0;
		}
	}
	@XmlElement(name = "TotalConfidentiality")
	private String totalConfidentiality;{
		if (totalConfidentiality==null){
			totalConfidentiality="NO";
		}
	}
	@XmlElement(name = "TotalReliability")
	private Integer totalReliability;{
		if (totalReliability==null){
			totalReliability=0;
		}
	}
	@XmlElement(name = "TotalReputation")
	private Integer totalReputation;{
		if (totalReputation==null){
			totalReputation=0;
		}
	}
```

```
	@XmlElement(name = "TotalPrice")
	private Integer totalPrice;{
		if (totalPrice==null){
			totalPrice=0;
		}
	}
	@XmlElement(name = "TotalTotal")
	private Integer totalTotal;{
		if (totalTotal==null){
			totalTotal=0;
		}
	}

	public Integer getId() {
		return id;
	}

	public void setId(Integer id) {
		this.id = id;
	}

	public BusinessClass getBusiness() {
		return business;
	}

	public void setBusiness(BusinessClass business) {
		this.business = business;
	}

	public TechnicalClass getTechnical() {
		return technical;
	}

	public void setTechnical(TechnicalClass technical) {
		this.technical = technical;
	}

	public UserReqClass getUserRequirements() {
		return userRequirements;
	}

	public void setUserRequirements(UserReqClass userRequirements) {
		this.userRequirements = userRequirements;
	}

	public Integer getTotalAvailability() {
		return totalAvailability;
	}

	public void setTotalAvailability(Integer totalAvailability) {
		this.totalAvailability = totalAvailability;
	}

	public Integer getTotalSecurity() {
		return totalSecurity;
	}

	public void setTotalSecurity(Integer totalSecurity) {
		this.totalSecurity = totalSecurity;
```

```
	}

	public String getTotalConfidentiality() {
		return totalConfidentiality;
	}

	public void setTotalConfidentiality(String totalConf) {
		this.totalConfidentiality = totalConf;
	}

	public Integer getTotalReliability() {
		return totalReliability;
	}

	public void setTotalReliability(Integer totalReliability) {
		this.totalReliability = totalReliability;
	}

	public Integer getTotalReputation() {
		return totalReputation;
	}

	public void setTotalReputation(Integer totalReputation) {
		this.totalReputation = totalReputation;
	}

	public Integer getTotalPrice() {
		return totalPrice;
	}

	public void setTotalPrice(Integer totalPrice) {
		this.totalPrice = totalPrice;
	}

	public Integer getTotalTotal() {
		return totalTotal;
	}

	public void setTotalTotal(Integer totalTotal) {
		this.totalTotal = totalTotal;
	}

	@Override
	public UserSystem clone() {
		UserSystem theClone = new UserSystem();
		theClone.setId(this.getId());
		if (this.getUserRequirements() != null) {

	theClone.setUserRequirements(this.getUserRequirements().clone());
		} else {
			UserReqClass userReq = new UserReqClass();
			theClone.setUserRequirements(userReq.empty());
		}
		if (this.getTechnical() != null) {
			theClone.setTechnical(this.getTechnical().clone());
		}else {
			theClone.setTechnical(this.getTechnical().empty());
		}
		if (this.getBusiness() != null) {
```

```
                    theClone.setBusiness(this.getBusiness().clone());
              }else {
                    theClone.setBusiness(this.getBusiness().empty());
              }
              theClone.setTotalAvailability(this.getTotalAvailability());
              theClone.setTotalSecurity(this.getTotalSecurity());
              theClone.setTotalConfidentiality(this.getTotalConfidentiality());
              theClone.setTotalReliability(this.getTotalReliability());
              theClone.setTotalReputation(this.getTotalReputation());
              theClone.setTotalPrice(this.getTotalPrice());
              theClone.setTotalTotal(this.getTotalTotal());
              return theClone;
       }

       public UserSystem empty() {
              UserSystem emptySystem = new UserSystem();
              emptySystem.setId(this.getId());
              if (this.getUserRequirements() != null) {

       emptySystem.setUserRequirements(this.getUserRequirements().empty());
              } else {
                    UserReqClass userReq = new UserReqClass();
                    emptySystem.setUserRequirements(userReq.empty());
              }
              if (this.getTechnical() != null) {
                    emptySystem.setTechnical(this.getTechnical().empty());
              } else {
                    TechnicalClass technical = new TechnicalClass();
                    emptySystem.setTechnical(technical.empty());
              }
              if (this.getBusiness() != null) {
                    emptySystem.setBusiness(this.getBusiness().empty());
              } else {
                    BusinessClass business = new BusinessClass();
                    emptySystem.setBusiness(business.empty());
              }

              emptySystem.setTotalAvailability(0);
              emptySystem.setTotalSecurity(0);
              emptySystem.setTotalConfidentiality("NO");
              emptySystem.setTotalReliability(0);
              emptySystem.setTotalReputation(0);
              emptySystem.setTotalPrice(0);
              emptySystem.setTotalTotal(0);
              return emptySystem;
       }
}
```

2)Η κλάση για τις προδιαγραφές του χρήστη (UserReqClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlType
@XmlRootElement(name = "UserRequirements")
```

```
@XmlAccessorType(XmlAccessType.FIELD)
public class UserReqClass implements Serializable, Cloneable {
	private static final long serialVersionUID = -3229148266600054676L;

	@XmlElement (name="Optimization")
	private OptClass optimization;
	@XmlElement (name="Service")
	private ServiceClass service;

	public OptClass getOptimization() {
		return optimization;
	}

	public void setOptimization(OptClass optimization) {
		this.optimization = optimization;
	}

	public ServiceClass getService() {
		return service;
	}

	public void setService(ServiceClass service) {
		this.service = service;
	}

	@Override
	public UserReqClass clone() {
		UserReqClass theClone = new UserReqClass();
		if (this.getOptimization() != null) {
			theClone.setOptimization(this.getOptimization().clone());
		} else {
			OptClass opt = new OptClass();
			theClone.setOptimization(opt.empty());
		}
		if (this.getService() != null) {
			theClone.setService(this.getService().clone());
		} else {
			ServiceClass service = new ServiceClass();
			theClone.setService(service.empty());
		}
		return theClone;
	}

	public UserReqClass empty() {
		UserReqClass theEmpty = new UserReqClass();
		if (this.getOptimization() != null) {
			theEmpty.setOptimization(this.getOptimization().empty());
		} else {
			OptClass optimization = new OptClass();
			theEmpty.setOptimization(optimization.empty());
		}
		if (this.getService() != null) {
			theEmpty.setService(this.getService().empty());
		}else {
			ServiceClass service = new ServiceClass();
			theEmpty.setService(service.empty());
		}
		return theEmpty;
	}
```

```
}
```

3) Η κλάση με τις ποιοτικές προδιαγραφές του χρήστη (OptClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;
import com.fasterxml.jackson.annotation.JsonIgnore;
@XmlType
@XmlRootElement(name = "Optimization")
@XmlAccessorType(XmlAccessType.FIELD)
public class OptClass implements Serializable, Cloneable {
	private static final long serialVersionUID = -322482143599160536L;

	@XmlElement (name = "MAXPrice")
	private Integer maxPrice;
	@XmlElement (name = "MINAvailability")
	private Integer minAvailability;
	@XmlElement (name = "MINSecurity")
	private Integer minSecurity;
	@XmlElement (name = "MINReputation")
	private Integer minReputation;
	@XmlElement (name = "MINReliability")
	private Integer minReliability;
	@XmlElement (name = "Confidentiality")
	private String confidentiality;
	@JsonIgnore
	@XmlTransient
	private Boolean conf;{
		if (confidentiality=="YES"){
			conf=true;
		} else
			conf=false;
	}

	public Boolean getConf() {
		return conf;
	}

	public void setConf(Boolean conf) {
		this.conf = conf;
	}

	public Integer getMinSecurity() {
		return minSecurity;
	}

	public void setMinSecurity(Integer minSecurity) {
		this.minSecurity = minSecurity;
	}

	public Integer getMinReputation() {
		return minReputation;
	}
```

```
	public void setMinReputation(Integer minReputation) {
		this.minReputation = minReputation;
	}

	public Integer getMinReliability() {
		return minReliability;
	}

	public void setMinReliability(Integer minReliability) {
		this.minReliability = minReliability;
	}

	public String getConfidentiality() {
		return confidentiality;
	}

	public void setConfidentiality(String confidentiality) {
		this.confidentiality = confidentiality;
	}

	public Integer getMaxPrice() {
		return maxPrice;
	}

	public void setMaxPrice(Integer maxPrice) {
		this.maxPrice = maxPrice;
	}

	public Integer getMinAvailability() {
		return minAvailability;
	}

	public void setMinAvailability(Integer minAvailability) {
		this.minAvailability = minAvailability;
	}

	@Override
	public OptClass clone() {
		OptClass theClone = new OptClass();
		theClone.setMaxPrice(this.getMaxPrice());
		theClone.setMinAvailability(this.getMinAvailability());
		theClone.setMinSecurity(this.getMinSecurity());
		theClone.setMinReliability(this.getMinReliability());
		theClone.setConfidentiality(this.getConfidentiality());
		theClone.setMinReputation(this.getMinReputation());
		theClone.setConf(this.getConf());
		return theClone;
	}
		public OptClass empty() {
		OptClass theEmpty = new OptClass();
		theEmpty.setMaxPrice(0);
		theEmpty.setMinAvailability(0);
		theEmpty.setMinSecurity(0);
		theEmpty.setMinReliability(0);
		theEmpty.setConfidentiality("NO");
		theEmpty.setMinReputation(0);
		theEmpty.setConf(false);
		return theEmpty;
	}
```

```
}
```

4) Η κλάση με τις τεχνικές προδιαγραφές του χρήστη οι οποίες αφορούν 4 διαφορετικά στοιχεία (Database, Operating System, CRM και Network (ServiceClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlType
@XmlRootElement(name = "Service")
@XmlAccessorType(XmlAccessType.FIELD)
public class ServiceClass implements Serializable, Cloneable {
	private static final long serialVersionUID = 6597221974556642678L;

	@XmlElement (name = "Database")
	private DatabaseClass database;
	@XmlElement (name = "OperatingSystem")
	private OperatingSysClass operatingSystem;
	@XmlElement (name = "CRM")
	private CRMClass crm;
	@XmlElement (name = "Network")
	private NetworkClass network;

	public DatabaseClass getDatabase() {
		return database;
	}
	public void setDatabase(DatabaseClass database) {
		this.database = database;
	}
	public OperatingSysClass getOperatingSystem() {
		return operatingSystem;
	}
	public void setOperatingSystem(OperatingSysClass operatingSystem) {
		this.operatingSystem = operatingSystem;
	}
	public CRMClass getCrm() {
		return crm;
	}
	public void setCrm(CRMClass crm) {
		this.crm = crm;
	}
	public NetworkClass getNetwork() {
		return network;
	}
	public void setNetwork(NetworkClass network) {
		this.network = network;
	}

	@Override
	public ServiceClass clone() {
		ServiceClass theClone = new ServiceClass();
		if (this.getDatabase() != null) {
			theClone.setDatabase(this.getDatabase().clone());
		} else {
			DatabaseClass database = new DatabaseClass();
```

```
                theClone.setDatabase(database.empty());
            }
            if (this.getOperatingSystem() != null) {

      theClone.setOperatingSystem(this.getOperatingSystem().clone());
            } else {
                OperatingSysClass operating = new OperatingSysClass();
                theClone.setOperatingSystem(operating.empty());
            }
            if (this.getCrm() != null) {
                theClone.setCrm(this.getCrm().clone());
            } else {
                CRMClass crm = new CRMClass();
                theClone.setCrm(crm.empty());
            }
            if (this.getNetwork() != null) {
                theClone.setNetwork(this.getNetwork().clone());
            } else {
                NetworkClass network = new NetworkClass();
                theClone.setNetwork(network.empty());
            }
            return theClone;
      }

      public ServiceClass empty() {
            ServiceClass theEmpty = new ServiceClass();
            if (this.getDatabase() != null) {
                theEmpty.setDatabase(this.getDatabase().empty());
            } else {
                DatabaseClass database = new DatabaseClass();
                theEmpty.setDatabase(database.empty());
            }
            if (this.getOperatingSystem() != null) {

      theEmpty.setOperatingSystem(this.getOperatingSystem().empty());
            } else {
                OperatingSysClass operating = new OperatingSysClass();
                theEmpty.setOperatingSystem(operating.empty());
            }
            if (this.getCrm() != null) {
                theEmpty.setCrm(this.getCrm().empty());
            } else {
                CRMClass crm = new CRMClass();
                theEmpty.setCrm(crm.empty());
            }
            if (this.getNetwork() != null) {
                theEmpty.setNetwork(this.getNetwork().empty());
            } else {
                NetworkClass network = new NetworkClass();
                theEmpty.setNetwork(network.empty());
            }
            return theEmpty;
      }
}
```

4α) Η κλάση που περιγράφει την βάση δεδομένων και τα χαρακτηριστικά της (DatabaseClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
```

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;
import com.fasterxml.jackson.annotation.JsonIgnore;

@XmlType
@XmlRootElement(name = "Database")
@XmlAccessorType(XmlAccessType.FIELD)
public class DatabaseClass implements Serializable, Cloneable{
	private static final long serialVersionUID = -649523521513696274L;

	@XmlElement (name = "ID")
	private String id; {
		if (id==null){
			id="0";
		}
	}
	@XmlElement (name = "Name")
	private String name; {
		if (name==null){
			name="";
		}
	}
	@XmlElement (name = "OP")
	private String op; {
		if (op==null){
			op="";
		}
	}
	@XmlElement (name = "Type")
	private String type;{
		if (type==null){
			type="";
		}
	}

	@XmlElement (name = "ServiceProvider")
	private String serviceProvider;{
		if (serviceProvider==null){
			serviceProvider="";
		}
	}

	@XmlElement (name = "Availability")
	private Integer availability;{
		if (availability==null){
			availability=0;
		}
	}
	@XmlElement (name = "Security")
	private Integer security;{
		if (security==null){
			security=0;
		}
	}
	@XmlElement (name = "Reputation")
```

```
	private Integer reputation;{
		if (reputation==null){
			reputation=0;
		}
	}
	@XmlElement (name = "Reliability")
	private Integer reliability;{
		if (reliability==null){
			reliability=0;
		}
	}
	@XmlElement (name = "Confidentiality")
	private String confidentiality;
	@XmlTransient
	@JsonIgnore
	private Boolean conf;{
		if (confidentiality==null){
			confidentiality="NO";
		}
		if (confidentiality=="NO"){
			conf=false;
		}
		if (confidentiality=="YES"){
			conf=true;
		}
	}
	@XmlElement (name = "Price")
	private Integer price;{
		if (price==null){
			price=0;
		}
	}

	public Boolean getConf() {
		return conf;
	}

	public void setConf(Boolean conf) {
		this.conf = conf;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public Integer getAvailability() {
		return availability;
	}

	public void setAvailability(Integer availability) {
		this.availability = availability;
	}

	public Integer getSecurity() {
		return security;
```

```
	}

	public void setSecurity(Integer security) {
		this.security = security;
	}

	public String getConfidentiality() {
		return confidentiality;
	}

	public void setConfidentiality(String confidentiality) {
		this.confidentiality = confidentiality;
	}

	public Integer getReliability() {
		return reliability;
	}

	public void setReliability(Integer reliability) {
		this.reliability = reliability;
	}

	public Integer getReputation() {
		return reputation;
	}

	public void setReputation(Integer reputation) {
		this.reputation = reputation;
	}

	public Integer getPrice() {
		return price;
	}

	public void setPrice(Integer price) {
		this.price = price;
	}

	public String getOp() {
		return op;
	}

	public void setOp(String op) {
		this.op = op;
	}

	public String getType() {
		return type;
	}

	public void setType(String type) {
		this.type = type;
	}

	public String getServiceProvider() {
		return serviceProvider;
	}

	public void setServiceProvider(String serviceProvider) {
```

```
		this.serviceProvider = serviceProvider;
	}


	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

	@Override
	public DatabaseClass clone() {

		DatabaseClass theClone = new DatabaseClass();
		theClone.setConf(this.getConf());
		theClone.setId(this.getId());
		theClone.setName(this.getName());
		theClone.setOp(this.getOp());
		theClone.setType(this.getType());
		theClone.setServiceProvider(this.getServiceProvider());
		theClone.setAvailability(this.getAvailability());
		theClone.setSecurity(this.getSecurity());
		theClone.setConfidentiality(this.getConfidentiality());
		theClone.setReliability(this.getReliability());
		theClone.setReputation(this.getReputation());
		theClone.setPrice(this.getPrice());
		return theClone;
	}

	public DatabaseClass empty() {
		DatabaseClass theEmpty = new DatabaseClass();
		theEmpty.setConf(false);
		theEmpty.setId("0");
		theEmpty.setName("");
		theEmpty.setOp("");
		theEmpty.setType("");
		theEmpty.setServiceProvider("");
		theEmpty.setAvailability(0);
		theEmpty.setSecurity(0);
		theEmpty.setConfidentiality("NO");
		theEmpty.setReliability(0);
		theEmpty.setReputation(0);
		theEmpty.setPrice(0);
		return theEmpty;
	}

}
```

4b)Η κλάση που περιγράφει το λειτουργικό σύστημα και τα χαρακτηριστικά του (OperatingSysClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
```

```
import javax.xml.bind.annotation.XmlType;
import com.fasterxml.jackson.annotation.JsonIgnore;

@XmlType
@XmlRootElement(name = "OperatingSystem")
@XmlAccessorType(XmlAccessType.FIELD)
public class OperatingSysClass implements Serializable, Cloneable {
	private static final long serialVersionUID = 6787497596842610368L;

	@XmlTransient
	@JsonIgnore
	private Boolean conf;
	@XmlElement (name = "ID")
	private String id; {
		if (id==null){
			id="0";
		}
	}
	@XmlElement (name = "Distribution")
	private String distribution; {
		if (distribution==null) {
			distribution="";
		}
	}
	@XmlElement (name = "Type")
	private String type;{
		if (type==null){
			type="";
		}
	}
	@XmlElement (name = "OP")
	private String op; {
		if (op==null){
			op="";
		}
	}

	@XmlElement (name = "ServiceProvider")
	private String serviceProvider;{
		if (serviceProvider==null){
			serviceProvider="";
		}
	}

	@XmlElement (name = "Availability")
	private Integer availability;{
		if (availability==null){
			availability=0;
		}
	}
	@XmlElement (name = "Security")
	private Integer security;{
		if (security==null){
			security=0;
		}
	}
	@XmlElement (name = "Reputation")
	private Integer reputation;{
		if (reputation==null){
```

```
                reputation=0;
        }
    }
    @XmlElement (name = "Reliability")
    private Integer reliability;{
        if (reliability==null){
                reliability=0;
        }
    }
    @XmlElement (name = "Confidentiality")
    private String confidentiality;{
        if (confidentiality==null){
                confidentiality="NO";
        }
        if (confidentiality=="NO"){
                conf=false;
        }
        if (confidentiality=="YES"){
                conf=true;
        }
    }
    @XmlElement (name = "Price")
    private Integer price;{
        if (price==null){
                price=0;
        }
    }

    public Boolean getConf() {
        return conf;
    }

    public void setConf(Boolean conf) {
        this.conf = conf;
    }

    public void setConfidentiality(String confidentiality) {
        this.confidentiality = confidentiality;
    }

    public String getConfidentiality() {
        return confidentiality;
    }

    public String getDistribution() {
        return distribution;
    }

    public void setDistribution(String distribution) {
        this.distribution = distribution;
    }

    public String getType() {
        return type;
    }

    public void setType(String type) {
        this.type = type;
    }
```

```
	public String getOp() {
		return op;
	}

	public void setOp(String op) {
		this.op = op;
	}

	public String getServiceProvider() {
		return serviceProvider;
	}

	public void setServiceProvider(String serviceProvider) {
		this.serviceProvider = serviceProvider;
	}

	public Integer getAvailability() {
		return availability;
	}

	public void setAvailability(Integer availability) {
		this.availability = availability;
	}

	public Integer getSecurity() {
		return security;
	}

	public void setSecurity(Integer security) {
		this.security = security;
	}

	public Integer getReliability() {
		return reliability;
	}

	public void setReliability(Integer reliability) {
		this.reliability = reliability;
	}

	public Integer getReputation() {
		return reputation;
	}

	public void setReputation(Integer reputation) {
		this.reputation = reputation;
	}

	public Integer getPrice() {
		return price;
	}

	public void setPrice(Integer price) {
		this.price = price;
	}

	public String getId() {
		return id;
```

```
	}

	public void setId(String id) {
		this.id = id;
	}

	@Override
	public OperatingSysClass clone() {

		OperatingSysClass theClone = new OperatingSysClass();
		theClone.setConf(this.getConf());
		theClone.setId(this.getId());
		theClone.setDistribution(this.getDistribution());
		theClone.setOp(this.getOp());
		theClone.setType(this.getType());
		theClone.setServiceProvider(this.getServiceProvider());
		theClone.setAvailability(this.getAvailability());
		theClone.setSecurity(this.getSecurity());
		theClone.setConfidentiality(this.getConfidentiality());
		theClone.setReliability(this.getReliability());
		theClone.setReputation(this.getReputation());
		theClone.setPrice(this.getPrice());
		return theClone;
	}

	public OperatingSysClass empty() {
		OperatingSysClass theEmpty = new OperatingSysClass();
		theEmpty.setConf(false);
		theEmpty.setId("0");
		theEmpty.setDistribution("");
		theEmpty.setOp("");
		theEmpty.setType("");
		theEmpty.setServiceProvider("");
		theEmpty.setAvailability(0);
		theEmpty.setSecurity(0);
		theEmpty.setConfidentiality("NO");
		theEmpty.setReliability(0);
		theEmpty.setReputation(0);
		theEmpty.setPrice(0);
		return theEmpty;
	}
}
```

4c)Η κλάση που περιγράφει το CRM και τα χαρακτηριστικά του (CRMClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;
import com.fasterxml.jackson.annotation.JsonIgnore;

@XmlType
@XmlRootElement(name = "CRM")
@XmlAccessorType(XmlAccessType.FIELD)
public class CRMClass implements Serializable, Cloneable {
	private static final long serialVersionUID = 3292269591176781380L;
```

```
@XmlTransient
@JsonIgnore
private Boolean conf;
@XmlElement (name = "ID")
private String id; {
	if (id==null){
		id="0";
	}
}
@XmlElement (name = "Name")
private String name; {
	if (name==null) {
		name="";
	}
}
@XmlElement (name = "OP")
private String op;

@XmlElement (name = "ServiceProvider")
private String serviceProvider;{
	if (serviceProvider==null){
		serviceProvider="";
	}
}

@XmlElement (name = "Availability")
private Integer availability;{
	if (availability==null){
		availability=0;
	}
}
@XmlElement (name = "Security")
private Integer security;{
	if (security==null){
		security=0;
	}
}
@XmlElement (name = "Reputation")
private Integer reputation;{
	if (reputation==null){
		reputation=0;
	}
}
@XmlElement (name = "Reliability")
private Integer reliability;{
	if (reliability==null){
		reliability=0;
	}
}
@XmlElement (name = "Confidentiality")
private String confidentiality;{
	if (confidentiality==null){
		confidentiality="NO";
	}
	if (confidentiality=="NO") {
		conf=false;
	}
	if (confidentiality =="YES"){
```

```
                conf=true;
        }
    }
    @XmlElement (name = "Price")
    private Integer price;{
        if (price==null){
                price=0;
        }
    }

    public Boolean getConf() {
        return conf;
    }

    public void setConf(Boolean conf) {
        this.conf = conf;
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getOp() {
        return op;
    }

    public void setOp(String op) {
        this.op = op;
    }

    public String getServiceProvider() {
        return serviceProvider;
    }

    public void setServiceProvider(String serviceProvider) {
        this.serviceProvider = serviceProvider;
    }

    public Integer getAvailability() {
        return availability;
    }

    public void setAvailability(Integer availability) {
        this.availability = availability;
    }

    public Integer getSecurity() {
```

```
		return security;
	}

	public void setSecurity(Integer security) {
		this.security = security;
	}

	public String getConfidentiality() {
		return confidentiality;
	}

	public void setConfidentiality(String confidentiality) {
		this.confidentiality = confidentiality;
	}

	public Integer getReliability() {
		return reliability;
	}

	public void setReliability(Integer reliability) {
		this.reliability = reliability;
	}

	public Integer getReputation() {
		return reputation;
	}

	public void setReputation(Integer reputation) {
		this.reputation = reputation;
	}

	public Integer getPrice() {
		return price;
	}

	public void setPrice(Integer price) {
		this.price = price;
	}

	@Override
	public CRMClass clone() {

		CRMClass theClone = new CRMClass();
		theClone.setConf(this.getConf());
		theClone.setId(this.getId());
		theClone.setName(this.getName());
		theClone.setOp(this.getOp());
		theClone.setServiceProvider(this.getServiceProvider());
		theClone.setAvailability(this.getAvailability());
		theClone.setSecurity(this.getSecurity());
		theClone.setConfidentiality(this.getConfidentiality());
		theClone.setReliability(this.getReliability());
		theClone.setReputation(this.getReputation());
		theClone.setPrice(this.getPrice());
		return theClone;
	}

	public CRMClass empty() {
		CRMClass theEmpty = new CRMClass();
```

```
            theEmpty.setConf(false);
            theEmpty.setId("0");
            theEmpty.setName("");
            theEmpty.setOp("");
            theEmpty.setServiceProvider("");
            theEmpty.setAvailability(0);
            theEmpty.setSecurity(0);
            theEmpty.setConfidentiality("NO");
            theEmpty.setReliability(0);
            theEmpty.setReputation(0);
            theEmpty.setPrice(0);
            return theEmpty;
      }
}
```

4d)Η κλάση που περιγράφει το δίκτυο και τα χαρακτηριστικά του (NetworkClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlTransient;
import javax.xml.bind.annotation.XmlType;
import com.fasterxml.jackson.annotation.JsonIgnore;

@XmlType
@XmlRootElement(name = "Network")
@XmlAccessorType(XmlAccessType.FIELD)
public class NetworkClass implements Serializable, Cloneable {
      private static final long serialVersionUID = 3288434547800216964L;

      @XmlTransient
      @JsonIgnore
      private Boolean conf;
      @XmlElement (name = "ID")
      private String id; {
            if (id==null){
                  id="0";
            }
      }
      @XmlElement (name ="Name")
      private String name; {
            if (name==null){
                  name="";
            }
      }
      @XmlElement (name = "Type")
      private String type;

      @XmlElement (name = "ServiceProvider")
      private String serviceProvider;{
            if (serviceProvider==null){
                  serviceProvider="";
            }
      }

      @XmlElement (name = "Availability")
      private Integer availability;{
```

```
		if (availability==null){
			availability=0;
		}
	}
	@XmlElement (name = "Security")
	private Integer security;{
		if (security==null){
			security=0;
		}
	}
	@XmlElement (name = "Reputation")
	private Integer reputation;{
		if (reputation==null){
			reputation=0;
		}
	}
	@XmlElement (name = "Reliability")
	private Integer reliability;{
		if (reliability==null){
			reliability=0;
		}
	}
	@XmlElement (name = "Confidentiality")
	private String confidentiality;{
		if (confidentiality==null){
			confidentiality="NO";
		}
		if (confidentiality=="NO"){
			conf=false;
		}
		if (confidentiality=="YES"){
			conf=true;
		}
	}
	@XmlElement (name = "Price")
	private Integer price;{
		if (price==null){
			price=0;
		}
	}

	public Boolean getConf() {
		return conf;
	}

	public void setConf(Boolean conf) {
		this.conf = conf;
	}

	public String getId() {
		return id;
	}

	public void setId(String id) {
		this.id = id;
	}

	public String getName() {
		return name;
```

```
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}

	public String getServiceProvider() {
		return serviceProvider;
	}
	public void setServiceProvider(String serviceProvider) {
		this.serviceProvider = serviceProvider;
	}
	public Integer getAvailability() {
		return availability;
	}
	public void setAvailability(Integer availability) {
		this.availability = availability;
	}
	public Integer getSecurity() {
		return security;
	}
	public void setSecurity(Integer security) {
		this.security = security;
	}
	public String getConfidentiality() {
		return confidentiality;
	}
	public void setConfidentiality(String confidentiality) {
		this.confidentiality = confidentiality;
	}
	public Integer getReliability() {
		return reliability;
	}
	public void setReliability(Integer reliability) {
		this.reliability = reliability;
	}
	public Integer getReputation() {
		return reputation;
	}
	public void setReputation(Integer reputation) {
		this.reputation = reputation;
	}
	public Integer getPrice() {
		return price;
	}
	public void setPrice(Integer price) {
		this.price = price;
	}
	@Override
	public NetworkClass clone() {

		NetworkClass theClone = new NetworkClass();
		theClone.setConf(this.getConf());
		theClone.setId(this.getId());
```

```
                theClone.setName(this.getName());
                theClone.setType(this.getType());
                theClone.setServiceProvider(this.getServiceProvider());
                theClone.setAvailability(this.getAvailability());
                theClone.setSecurity(this.getSecurity());
                theClone.setConfidentiality(this.getConfidentiality());
                theClone.setReliability(this.getReliability());
                theClone.setReputation(this.getReputation());
                theClone.setPrice(this.getPrice());

                return theClone;
        }

public NetworkClass empty() {

                NetworkClass theEmpty = new NetworkClass();
                theEmpty.setConf(false);
                theEmpty.setId("0");
                theEmpty.setName("");
                theEmpty.setType("");
                theEmpty.setServiceProvider("");
                theEmpty.setAvailability(0);
                theEmpty.setSecurity(0);
                theEmpty.setConfidentiality("NO");
                theEmpty.setReliability(0);
                theEmpty.setReputation(0);
                theEmpty.setPrice(0);
                return theEmpty;
        }
}
```

5) Η κλάση που χρησιμοποιείται για να περιγραφούν όλα τα τεχνικά στοιχεία κάθε πιθανής απάντησης στις προδιαγραφές που έθεσε ο χρήστης (TechnicalClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlType
@XmlRootElement(name = "Technical")
@XmlAccessorType(XmlAccessType.FIELD)
public class TechnicalClass implements Serializable, Cloneable {
        private static final long serialVersionUID = -679369808642816453L;

        private DatabaseClass database= new DatabaseClass();
        private OperatingSysClass operatingSystem = new OperatingSysClass();
        private CRMClass crm = new CRMClass();
        private NetworkClass network = new NetworkClass();

        public DatabaseClass getDatabase() {
                return database;
        }
        public void setDatabase(DatabaseClass Database) {
                database = Database;
        }
        public OperatingSysClass getOperatingSystem() {
```

```
		return operatingSystem;
	}
	public void setOperatingSystem(OperatingSysClass operatingSystem) {
		this.operatingSystem = operatingSystem;
	}
	public CRMClass getCrm() {
		return crm;
	}
	public void setCrm(CRMClass crm) {
		this.crm = crm;
	}

	public NetworkClass getNetwork() {
		return network;
	}
	public void setNetwork(NetworkClass network) {
		this.network = network;
	}

	@Override
	public TechnicalClass clone() {
		TechnicalClass theClone = new TechnicalClass();
		if (this.getDatabase() != null) {
			theClone.setDatabase(this.getDatabase().clone());
		} else {
			DatabaseClass database = new DatabaseClass();
			theClone.setDatabase(database.empty());
		}
		if (this.getOperatingSystem() != null) {

	theClone.setOperatingSystem(this.getOperatingSystem().clone());
		} else {
			OperatingSysClass operating = new OperatingSysClass();
			theClone.setOperatingSystem(operating.empty());
		}
		if (this.getCrm() != null) {
			theClone.setCrm(this.getCrm().clone());
		} else {
			CRMClass crm = new CRMClass();
			theClone.setCrm(crm.empty());
		}
		if (this.getNetwork() != null) {
			theClone.setNetwork(this.getNetwork().clone());
		} else {
			NetworkClass network = new NetworkClass();
			theClone.setNetwork(network.empty());
		}
		return theClone;
	}

	public TechnicalClass empty() {
		TechnicalClass theEmpty = new TechnicalClass();

		if (this.getDatabase() != null) {
			theEmpty.setDatabase(this.getDatabase().empty());
		} else {
			DatabaseClass database = new DatabaseClass();
			theEmpty.setDatabase(database.empty());
		}
```

```
		if (this.getOperatingSystem() != null) {

	theEmpty.setOperatingSystem(this.getOperatingSystem().empty());
		} else {
			OperatingSysClass operating = new OperatingSysClass();
			theEmpty.setOperatingSystem(operating.empty());
		}
		if (this.getCrm() != null) {
			theEmpty.setCrm(this.getCrm().empty());
		} else {
			CRMClass crm = new CRMClass();
			theEmpty.setCrm(crm.empty());
		}
		if (this.getNetwork() != null) {
			theEmpty.setNetwork(this.getNetwork().empty());
		} else {
			NetworkClass network = new NetworkClass();
			theEmpty.setNetwork(network.empty());
		}
		return theEmpty;
	}
}
```

6) Η κλάση που χρησιμοποιείται για να περιγραφούν όλα τα τεχνικά στοιχεία κάθε πιθανής απάντησης στις προδιαγραφές που έθεσε ο χρήστης μαζί με τον εκάστοτε προμηθευτή αυτής και τα ποιοτικά χαρακτηριστικά που προσφέρει (BusinessClass.java)

```
package edu.gr.ntua.sylvia.web.model;
import java.io.Serializable;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlType
@XmlRootElement(name = "Business")
@XmlAccessorType(XmlAccessType.FIELD)
public class BusinessClass implements Serializable, Cloneable {
	private static final long serialVersionUID = 7419780890260718493L;

	private DatabaseClass database = new DatabaseClass();
	private OperatingSysClass operatingSystem = new OperatingSysClass();
	private CRMClass crm = new CRMClass();
	private NetworkClass network = new NetworkClass();

	public DatabaseClass getDatabase() {
		return database;
	}

	public void setDatabase(DatabaseClass database) {
		this.database = database;
	}

	public OperatingSysClass getOperatingSystem() {
		return operatingSystem;
	}

	public void setOperatingSystem(OperatingSysClass operatingSystem) {
		this.operatingSystem = operatingSystem;
```

```
	}

	public CRMClass getCrm() {
		return crm;
	}

	public void setCrm(CRMClass crm) {
		this.crm = crm;
	}

	public NetworkClass getNetwork() {
		return network;
	}

	public void setNetwork(NetworkClass network) {
		this.network = network;
	}

	@Override
	public BusinessClass clone() {

		BusinessClass theClone = new BusinessClass();
		if (this.getDatabase() != null) {
			theClone.setDatabase(this.getDatabase().clone());
		} else {
			DatabaseClass database = new DatabaseClass();
			theClone.setDatabase(database.empty());
		}
		if (this.getOperatingSystem() != null) {

	theClone.setOperatingSystem(this.getOperatingSystem().clone());
		} else {
			OperatingSysClass operating = new OperatingSysClass();
			theClone.setOperatingSystem(operating.empty());
		}
		if (this.getCrm() != null) {
			theClone.setCrm(this.getCrm().clone());
		} else {
			CRMClass crm = new CRMClass();
			theClone.setCrm(crm.empty());
		}
		if (this.getNetwork() != null){
			theClone.setNetwork(this.getNetwork().clone());
		} else {
			NetworkClass network = new NetworkClass();
			theClone.setNetwork(network.empty());
		}
		return theClone;
	}

	public BusinessClass empty() {
		BusinessClass theEmpty = new BusinessClass();
		if (this.getDatabase() != null) {
			theEmpty.setDatabase(this.getDatabase().empty());
		} else {
			DatabaseClass database = new DatabaseClass();
			theEmpty.setDatabase(database.empty());
		}
		if (this.getOperatingSystem() != null) {
```

```
        theEmpty.setOperatingSystem(this.getOperatingSystem().empty());
                } else {
                        OperatingSysClass operating = new OperatingSysClass();
                        theEmpty.setOperatingSystem(operating.empty());
                }
                if (this.getCrm() != null) {
                        theEmpty.setCrm(this.getCrm().empty());
                } else {
                        CRMClass crm = new CRMClass();
                        theEmpty.setCrm(crm.empty());
                }
                if (this.getNetwork() != null){
                        theEmpty.setNetwork(this.getNetwork().empty());
                } else {
                        NetworkClass network = new NetworkClass();
                        theEmpty.setNetwork(network.empty());
                }
                return theEmpty;
        }
}
```

Η διαδικασία (process)μέσω της οποίας το Camel καλεί το web service που εκτελεί την Τεχνική Φάση (TechnicalPhase.java)

```
package edu.gr.ntua.sylvia.process;

import javax.annotation.Resource;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Component
public class TechnicalPhase {

        @Resource
        private RestTemplate restTemplate;

        public void TechnicalRequest(UserSystem system) {

        restTemplate.postForObject("http://localhost:8080/web/technical/createTech
Solutions", system, UserSystem.class);
        }

        public void CommonTechnicalRequest(UserSystem system) {

        restTemplate.postForObject("http://localhost:8080/web/technical/createTech
SolutionsCommon", system, UserSystem.class);
        }
}
```

Ο controller που διαχειρίζεται το web service και εκτελεί τα αιτήματα του Camel (TechnicalController.java)

```
package edu.gr.ntua.sylvia.web.controller;
import java.io.File;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import edu.gr.ntua.sylvia.web.model.CRMClass;
```

```
import edu.gr.ntua.sylvia.web.model.DatabaseClass;
import edu.gr.ntua.sylvia.web.model.NetworkClass;
import edu.gr.ntua.sylvia.web.model.OperatingSysClass;
import edu.gr.ntua.sylvia.web.model.UserSystem;
import java.util.ArrayList;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.annotation.Resource;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.jms.Session;
import org.apache.commons.io.FileUtils;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;

@Controller
@RequestMapping("/technical")
public class TechnicalController {

	private String filePath="C:/Users/Sylvie/Desktop/ThesisDb/Technical/";
	private String databasePath;
	private String operatingSysPath;
	private String CrmPath;
	private String NetworkPath;

	@Resource
	private ConnectionFactory connectionFactory;

	@Resource(name = "myJmsTemplate")
	private JmsTemplate jmsTemplate;

	@Resource(name = "myJmsTemplateCommon")
	private JmsTemplate jmsTemplateCommon;

	private Connection jmsConnection;
	private Session jmsSession;
	private Connection jmsConnectionCommon;
	private Session jmsSessionCommon;
	private MessageProducer jmsQueueProducer;
	private MessageProducer jmsCommonProducer;

	//Creation of connection between the web service of Technical Controller
and the queue1
	@PostConstruct
	public void prepareJmsStuff() throws JMSException {
		jmsConnection = connectionFactory.createConnection();
		jmsConnection.start();
		jmsSession = jmsConnection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
```

```
            Destination queue = jmsSession.createQueue("queue1");
            jmsQueueProducer = jmsSession.createProducer(queue);
            jmsQueueProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
      }

      @PreDestroy
      public void cleanUpJmsStuff() throws JMSException {
            if (jmsConnection != null) {
                  jmsConnection.close();
            }
      }
      //Creation of connection between the web service of Technical Controller
and the common queue
      @PostConstruct
      public void prepareJmsStuffCommon() throws JMSException {
            jmsConnectionCommon = connectionFactory.createConnection();
            jmsConnectionCommon.start();
            jmsSessionCommon = jmsConnectionCommon.createSession(false,
Session.AUTO_ACKNOWLEDGE);
            Destination common = jmsSessionCommon.createQueue("common");
            jmsCommonProducer = jmsSessionCommon.createProducer(common);
            jmsCommonProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
      }

      @PreDestroy
      public void cleanUpJmsStuffCommon() throws JMSException {
            if (jmsConnectionCommon != null) {
                  jmsConnectionCommon.close();
            }
      }

      //Controller for architecture 1 and seperate queues
      @ResponseBody
      @RequestMapping(value = "/createTechSolutions", method =
{RequestMethod.POST}, consumes = {"application/json", "application/xml"})
      public void createSolutions(@RequestBody final UserSystem userSystem)
throws JMSException,JAXBException {

            MyContainer myContainer = prepareItems(userSystem);

            //Create final technical objects and send them to queue1
            for (final DatabaseClass database : myContainer.getDatabaseList())
{
                  userSystem.getTechnical().setDatabase(database);
                  for (final OperatingSysClass os :
myContainer.getOperatingSysList()) {
                        userSystem.getTechnical().setOperatingSystem(os);
                        for (final CRMClass crm : myContainer.getCrmList()) {
                              userSystem.getTechnical().setCrm(crm);

                              for (final NetworkClass network :
myContainer.getNetworkList()) {

      userSystem.getTechnical().setNetwork(network);
                                    final UserSystem messageObject =
userSystem;
                                    try {
                                          Message jmsMessage =
jmsSession.createObjectMessage(messageObject);
```

```
                                        jmsQueueProducer.send(jmsMessage);
                                  } catch (JMSException e) {
                                        e.printStackTrace();
                                  }
                            }
                      }
                }
          }
    }

    //Controller for architecture 2 and common queue
    @ResponseBody
    @RequestMapping(value = "/createTechSolutionsCommon", method =
{RequestMethod.POST}, consumes = {"application/json", "application/xml"})
    public void createSolutionsCommon(@RequestBody final UserSystem
userSystem) throws JMSException,JAXBException {

          MyContainer myContainer = prepareItems(userSystem);

          //Create final technical objects and send them to common
          for (final DatabaseClass database : myContainer.getDatabaseList())
{
                userSystem.getTechnical().setDatabase(database);
                for (final OperatingSysClass os :
myContainer.getOperatingSysList()) {
                      userSystem.getTechnical().setOperatingSystem(os);

                      for (final CRMClass crm : myContainer.getCrmList()) {
                            userSystem.getTechnical().setCrm(crm);

                            for (final NetworkClass network :
myContainer.getNetworkList()) {

    userSystem.getTechnical().setNetwork(network);
                                  userSystem.setId(2);
                                  final UserSystem messageObject =
userSystem;
                                  try {
                                        Message jmsMessageCommon =
jmsSessionCommon.createObjectMessage(messageObject);

    jmsCommonProducer.send(jmsMessageCommon);
                                  } catch (JMSException e) {
                                        e.printStackTrace();
                                  }
                            }
                      }
                }
          }
    }

    private MyContainer prepareItems(UserSystem userSystem) throws
JAXBException {
          MyContainer response = new MyContainer();

          File f1 = null;
          File f2 = null;
          File f3 = null;
          File f4 = null;
```

```
            String[] extensions = {"xml"};

            //Database list
            List<DatabaseClass> databaseList = new ArrayList<>();
            databasePath=
filePath+"Database/"+userSystem.getUserRequirements().getService().getDatabase()
.getOp()+"/"+userSystem.getUserRequirements().getService().getDatabase().getType
(); //+ "?include=.*.xml";
            f1 = new File(databasePath);

            JAXBContext dbjc = JAXBContext.newInstance(DatabaseClass.class);
            Unmarshaller dbu = dbjc.createUnmarshaller();

            // for each pathname in pathname array
          for (File dbPath : FileUtils.listFiles(f1, extensions, false)) {
            final String help1 = dbPath.getAbsolutePath();
                  DatabaseClass database = (DatabaseClass) dbu.unmarshal(new
File (help1));
                  databaseList.add(database);
          }

        response.setDatabaseList(databaseList);

         //Operating Systems' list
            List<OperatingSysClass> operatingSysList = new ArrayList<>();
            operatingSysPath =
filePath+"OperatingSystem/"+userSystem.getUserRequirements().getService().getOpe
ratingSystem().getOp()+"/"+userSystem.getUserRequirements().getService().getOper
atingSystem().getType();
            f2 = new File(operatingSysPath);

            JAXBContext osjc =
JAXBContext.newInstance(OperatingSysClass.class);
            Unmarshaller osu = osjc.createUnmarshaller();

            // for each pathname in pathname array
        for (File osPath : FileUtils.listFiles(f2, extensions, false)) {
            final String help2 = osPath.getAbsolutePath();
                  OperatingSysClass operatingSys = (OperatingSysClass)
osu.unmarshal(new File (help2));
                  operatingSysList.add(operatingSys);
        }

        response.setOperatingSysList(operatingSysList);

        // CRM List
            List<CRMClass> crmList = new ArrayList<>();
            CrmPath =
filePath+"CRM/"+userSystem.getUserRequirements().getService().getCrm().getOp();
            f3 = new File(CrmPath);

            JAXBContext crmjc = JAXBContext.newInstance(CRMClass.class);
            Unmarshaller crmu = crmjc.createUnmarshaller();

            // for each pathname in pathname array
          for (File crmPath : FileUtils.listFiles(f3, extensions, false)) {
              final String help3 = crmPath.getAbsolutePath();
                  CRMClass crm =(CRMClass) crmu.unmarshal(new File (help3));
                  crmList.add(crm);
```

```
            }

            response.setCrmList(crmList);

            // Network List
               List<NetworkClass> networkList = new ArrayList<>();
               NetworkPath =
filePath+"Network/"+userSystem.getUserRequirements().getService().getNetwork().g
etType();
               f4 = new File(NetworkPath);

               JAXBContext njc = JAXBContext.newInstance(NetworkClass.class);
               Unmarshaller nu = njc.createUnmarshaller();

               // for each pathname in pathname array
               for (File networkPath : FileUtils.listFiles(f4, extensions, false))
{
                     final String help4 = networkPath.getAbsolutePath();
                     NetworkClass network = (NetworkClass) nu.unmarshal(new File
(help4));
                     networkList.add(network);
               }

            response.setNetworkList(networkList);

               return response;
      }

      private class MyContainer {
            private List<DatabaseClass> databaseList;
            private List<OperatingSysClass> operatingSysList;
            private List<CRMClass> crmList;
            private List<NetworkClass> networkList;
            public List<DatabaseClass> getDatabaseList() {
                  return databaseList;
            }
            public void setDatabaseList(List<DatabaseClass> databaseList) {
                  this.databaseList = databaseList;
            }
            public List<OperatingSysClass> getOperatingSysList() {
                  return operatingSysList;
            }
            public void setOperatingSysList(List<OperatingSysClass>
operatingSysList) {
                  this.operatingSysList = operatingSysList;
            }
            public List<CRMClass> getCrmList() {
                  return crmList;
            }
            public void setCrmList(List<CRMClass> crmList) {
                  this.crmList = crmList;
            }
            public List<NetworkClass> getNetworkList() {
                  return networkList;
            }
            public void setNetworkList(List<NetworkClass> networkList) {
                  this.networkList = networkList;
            }
      }
```

```
}
```

Η διαδικασία (process)μέσω της οποίας το Camel καλεί το web service που εκτελεί την Επιχειρηματική Φάση (BusinessPhase.java)

```
package edu.gr.ntua.sylvia.process;
import javax.annotation.Resource;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Component
public class BusinessPhase {

	@Resource
	private RestTemplate restTemplate;

	public void BusinessRequest(UserSystem system) {

	restTemplate.postForObject("http://localhost:8080/web/business/CheckBusine
ss", system, UserSystem.class);
	}

	public void CommonBusinessRequest(UserSystem system) {

	restTemplate.postForObject("http://localhost:8080/web/business/CheckBusine
ssCommon", system, UserSystem.class);
	}

}
```

Ο controller που διαχειρίζεται το web service και εκτελεί τα αιτήματα του Camel στην Επιχειρηματική φάση (BusinessController.java)

```
package edu.gr.ntua.sylvia.web.controller;
import java.io.File;
import java.util.ArrayList;
import java.util.List;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.annotation.Resource;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import org.apache.commons.io.FileUtils;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import edu.gr.ntua.sylvia.web.model.CRMClass;
import edu.gr.ntua.sylvia.web.model.DatabaseClass;
import edu.gr.ntua.sylvia.web.model.NetworkClass;
import edu.gr.ntua.sylvia.web.model.OperatingSysClass;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Controller
@RequestMapping("/business")
public class BusinessController {

	private String filePath="C:/Users/Sylvie/Desktop/ThesisDb/Business/";
	private String databasePath;
	private String operatingSysPath;
	private String CrmPath;
	private String NetworkPath;

	@Resource
	private ConnectionFactory connectionFactory;

	@Resource(name = "myJmsTemplate")
	private JmsTemplate jmsTemplate;

	@Resource(name = "myJmsTemplateCommon")
	private JmsTemplate jmsTemplateCommon;

	private Connection jmsConnection;
	private Session jmsSession;
	private Connection jmsConnectionCommon;
	private Session jmsSessionCommon;
	private MessageProducer jmsQueueProducer;
	private MessageProducer jmsCommonProducer;

	//Creation of connection between the web service of Business Controller
and the queue2
	@PostConstruct
	public void prepareJmsStuff() throws JMSException {
		jmsConnection = connectionFactory.createConnection();
		jmsConnection.start();
		jmsSession = jmsConnection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
		Destination queue = jmsSession.createQueue("queue2");
		jmsQueueProducer = jmsSession.createProducer(queue);
		jmsQueueProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
	}

	@PreDestroy
	public void cleanUpJmsStuff() throws JMSException {
		if (jmsConnection != null) {
			jmsConnection.close();
		}
	}

	//Creation of connection between the web service of Business Controller
and the common queue
	@PostConstruct
	public void prepareJmsStuffCommon() throws JMSException {
		jmsConnectionCommon = connectionFactory.createConnection();
```

```
            jmsConnectionCommon.start();
            jmsSessionCommon = jmsConnectionCommon.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            Destination common = jmsSessionCommon.createQueue("common");
            jmsCommonProducer = jmsSessionCommon.createProducer(common);
            jmsCommonProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
      }

      @PreDestroy
      public void cleanUpJmsStuffCommon() throws JMSException {
            if (jmsConnectionCommon != null) {
                  jmsConnectionCommon.close();
            }
      }

      @ResponseBody
      @RequestMapping(value = "/CheckBusiness", method = {RequestMethod.POST},
consumes = {"application/json", "application/xml"})
      public void businessSolutions(@RequestBody final UserSystem userSystem)
throws JMSException, JAXBException {

            BusinessContainer businessContainer = prepareItems(userSystem);
            sendMessages(userSystem, businessContainer, jmsTemplate, "queue2");
      }

      @ResponseBody
      @RequestMapping(value = "/CheckBusinessCommon", method =
{RequestMethod.POST}, consumes = {"application/json", "application/xml"})
      public void businessSolutionsCommon(@RequestBody final UserSystem
userSystem) throws JMSException, JAXBException {

            BusinessContainer businessContainer = prepareItems(userSystem);
            sendMessages(userSystem, businessContainer, jmsTemplateCommon,
"common");
      }

      private BusinessContainer prepareItems(UserSystem userSystem) throws
JAXBException {
            BusinessContainer response = new BusinessContainer();

            File f1 = null;
            File f2 = null;
            File f3 = null;
            File f4 = null;
            String[] extensions = {"xml"};

            //Database list
            List<DatabaseClass> databaseList = new ArrayList<>();
            databasePath=
filePath+"Database/"+userSystem.getTechnical().getDatabase().getOp()+"/"+userSys
tem.getTechnical().getDatabase().getType()+"/"+userSystem.getTechnical().getData
base().getId();
            f1 = new File(databasePath);

            JAXBContext dbjc = JAXBContext.newInstance(DatabaseClass.class);
            Unmarshaller dbu = dbjc.createUnmarshaller();

            // for each pathname in pathname array
```

```
          for (File dbPath : FileUtils.listFiles(f1, extensions, false)) {
            final String help1 = dbPath.getAbsolutePath();
                  DatabaseClass database = (DatabaseClass) dbu.unmarshal(new
File (help1));
                  databaseList.add(database);
          }

          response.setDatabaseList(databaseList);

          //Operating Systems' list
            List<OperatingSysClass> operatingSysList = new ArrayList<>();
            operatingSysPath =
filePath+"OperatingSystem/"+userSystem.getTechnical().getOperatingSystem().getOp
()+"/"+userSystem.getTechnical().getOperatingSystem().getType()+"/"+userSystem.g
etTechnical().getOperatingSystem().getId();
            f2 = new File(operatingSysPath);

      JAXBContext osjc = JAXBContext.newInstance(OperatingSysClass.class);
            Unmarshaller osu = osjc.createUnmarshaller();

            // for each pathname in pathname array
       for (File osPath : FileUtils.listFiles(f2, extensions, false)) {
            final String help2 = osPath.getAbsolutePath();
                  OperatingSysClass operatingSys = (OperatingSysClass)
osu.unmarshal(new File (help2));
                  operatingSysList.add(operatingSys);
       }

       response.setOperatingSysList(operatingSysList);

       // CRM List
            List<CRMClass> crmList = new ArrayList<>();
            CrmPath =
filePath+"CRM/"+userSystem.getTechnical().getCrm().getOp()+"/"+userSystem.getTec
hnical().getCrm().getId();
            f3 = new File(CrmPath);

       JAXBContext crmjc = JAXBContext.newInstance(CRMClass.class);
            Unmarshaller crmu = crmjc.createUnmarshaller();

            // for each pathname in pathname array
          for (File crmPath : FileUtils.listFiles(f3, extensions, false)) {
              final String help3 = crmPath.getAbsolutePath();
                  CRMClass crm =(CRMClass) crmu.unmarshal(new File (help3));
                  crmList.add(crm);
          }

          response.setCrmList(crmList);

          // Network List
            List<NetworkClass> networkList = new ArrayList<>();
            NetworkPath =
filePath+"Network/"+userSystem.getTechnical().getNetwork().getType()+"/"+userSys
tem.getTechnical().getNetwork().getId();
            f4 = new File(NetworkPath);

       JAXBContext njc = JAXBContext.newInstance(NetworkClass.class);
            Unmarshaller nu = njc.createUnmarshaller();
```

```
            // for each pathname in pathname array
          for (File networkPath : FileUtils.listFiles(f4, extensions, false)) {
              final String help4 = networkPath.getAbsolutePath();
                    NetworkClass network = (NetworkClass) nu.unmarshal(new File
(help4));
                    networkList.add(network);
          }

          response.setNetworkList(networkList);
//        System.out.println("prepareItems finished in " +
(System.currentTimeMillis()-startTimestamp) + "ms");
             return response;
      }

      private void sendMessages(UserSystem userSystem, BusinessContainer
businessContainer, JmsTemplate targetJmsTemplate, String targetMessageRecipient)
{
             for (final DatabaseClass database :
businessContainer.getDatabaseList()) {
                    userSystem.getBusiness().setDatabase(database);

      userSystem.getBusiness().getDatabase().setName(userSystem.getTechnical().g
etDatabase().getName());

      userSystem.getBusiness().getDatabase().setOp(userSystem.getTechnical().get
Database().getOp());

      userSystem.getBusiness().getDatabase().setType(userSystem.getTechnical().g
etDatabase().getType());

                    for (final OperatingSysClass os :
businessContainer.getOperatingSysList()) {
                           userSystem.getBusiness().setOperatingSystem(os);

      userSystem.getBusiness().getOperatingSystem().setDistribution(userSystem.g
etTechnical().getOperatingSystem().getDistribution());

      userSystem.getBusiness().getOperatingSystem().setOp(userSystem.getTechnica
l().getOperatingSystem().getOp());

      userSystem.getBusiness().getOperatingSystem().setType(userSystem.getTechni
cal().getOperatingSystem().getType());

                           for (final CRMClass crm :
businessContainer.getCrmList()) {
                                  userSystem.getBusiness().setCrm(crm);

      userSystem.getBusiness().getCrm().setName(userSystem.getTechnical().getCrm
().getName());

      userSystem.getBusiness().getCrm().setOp(userSystem.getTechnical().getCrm()
.getOp());

                                  for (final NetworkClass network :
businessContainer.getNetworkList()) {

      userSystem.getBusiness().setNetwork(network);

      userSystem.getBusiness().getNetwork().setName(userSystem.getTechnical().ge
```

```
tNetwork().getName());

      userSystem.getBusiness().getNetwork().setType(userSystem.getTechnical().ge
tNetwork().getType());

                                          userSystem.setId(3);
                                          final UserSystem messageObject =
userSystem;
                                          sendMessage(messageObject,
targetJmsTemplate, targetMessageRecipient);
                                    }
                              }
                        }
                  }
            }

      private void sendMessage(final UserSystem messageObject, JmsTemplate
targetJmsTemplate, String targetMessageRecipient) {
            if (targetMessageRecipient=="common"){
                  try {
                        Message jmsMessageCommon =
jmsSessionCommon.createObjectMessage(messageObject);
                        jmsCommonProducer.send(jmsMessageCommon);
                  } catch (JMSException e) {
                        e.printStackTrace();
                  }
            } else {
                  try {
                        Message jmsMessage =
jmsSession.createObjectMessage(messageObject);
                        jmsQueueProducer.send(jmsMessage);
                  } catch (JMSException e) {
                        e.printStackTrace();
                  }
            }
      }
      private class BusinessContainer {
            private List<DatabaseClass> databaseList;
            private List<OperatingSysClass> operatingSysList;
            private List<CRMClass> crmList;
            private List<NetworkClass> networkList;
            public List<DatabaseClass> getDatabaseList() {
                  return databaseList;
            }
            public void setDatabaseList(List<DatabaseClass> databaseList) {
                  this.databaseList = databaseList;
            }
            public List<OperatingSysClass> getOperatingSysList() {
                  return operatingSysList;
            }
            public void setOperatingSysList(List<OperatingSysClass>
operatingSysList) {
                  this.operatingSysList = operatingSysList;
            }
            public List<CRMClass> getCrmList() {
                  return crmList;
            }
            public void setCrmList(List<CRMClass> crmList) {
                  this.crmList = crmList;
```

```
		}
		public List<NetworkClass> getNetworkList() {
			return networkList;
		}
		public void setNetworkList(List<NetworkClass> networkList) {
			this.networkList = networkList;
		}
	}
}
```

Η διαδικασία (process)μέσω της οποίας το Camel καλεί το web service που εκτελεί την Φάση της Τιμής (PricePhase.java)

```
package edu.gr.ntua.sylvia.process;
import javax.annotation.Resource;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Component
public class PricePhase {

	@Resource
	private RestTemplate restTemplate;

	public void PriceRequest(UserSystem system) {

	restTemplate.postForObject("http://localhost:8080/web/price/checkPrice",
system, UserSystem.class);
	}

	public void CommonPriceRequest(UserSystem system) {

	restTemplate.postForObject("http://localhost:8080/web/price/checkPriceComm
on", system, UserSystem.class);
	}
}
```

Ο controller που διαχειρίζεται το web service και εκτελεί τα αιτήματα του Camel στην Φάση της Τιμής (PriceController.java)

```
package edu.gr.ntua.sylvia.web.controller;
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.annotation.Resource;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.xml.bind.JAXBException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
```

```
import org.springframework.web.bind.annotation.ResponseBody;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Controller
@RequestMapping("/price")
public class PriceController {

	private Integer totalPrice;

	Logger logger = LoggerFactory.getLogger(PriceController.class);

	@Resource
	private ConnectionFactory connectionFactory;

	@Resource(name = "myJmsTemplate")
	private JmsTemplate jmsTemplate;

	@Resource(name = "myJmsTemplateCommon")
	private JmsTemplate jmsTemplateCommon;

	private Connection jmsConnection;
	private Session jmsSession;
	private Connection jmsConnectionCommon;
	private Session jmsSessionCommon;
	private MessageProducer jmsQueueProducer;
	private MessageProducer jmsCommonProducer;

	//Creation of connection between the web service of Price Controller and
the queue3
	@PostConstruct
	public void prepareJmsStuff() throws JMSException {
		jmsConnection = connectionFactory.createConnection();
		jmsConnection.start();
		jmsSession = jmsConnection.createSession(false,
Session.AUTO_ACKNOWLEDGE);
		Destination queue = jmsSession.createQueue("queue3");
		jmsQueueProducer = jmsSession.createProducer(queue);
		jmsQueueProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
	}

	@PreDestroy
	public void cleanUpJmsStuff() throws JMSException {
		if (jmsConnection != null) {
			jmsConnection.close();
		}
	}

	//Creation of connection between the web service of Price Controller and
the common queue
	@PostConstruct
	public void prepareJmsStuffCommon() throws JMSException {
		jmsConnectionCommon = connectionFactory.createConnection();
		jmsConnectionCommon.start();
		jmsSessionCommon = jmsConnectionCommon.createSession(false,
Session.AUTO_ACKNOWLEDGE);
		Destination common = jmsSessionCommon.createQueue("common");
		jmsCommonProducer = jmsSessionCommon.createProducer(common);
		jmsCommonProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);
	}
```

```
	@PreDestroy
	public void cleanUpJmsStuffCommon() throws JMSException {
		if (jmsConnectionCommon != null) {
			jmsConnectionCommon.close();
		}
	}

	@ResponseBody
	@RequestMapping(value = "/checkPrice", method = {RequestMethod.POST},
consumes = {"application/json", "application/xml"})
	public void handlePriceRequest(@RequestBody UserSystem mySystem) throws
JMSException, JAXBException {

		//computation of TotalPrice
		totalPrice =
mySystem.getBusiness().getDatabase().getPrice()+mySystem.getBusiness().getOperat
ingSystem().getPrice()+mySystem.getBusiness().getCrm().getPrice()+mySystem.getBu
siness().getNetwork().getPrice();
		mySystem.setTotalPrice(totalPrice);

		if
(totalPrice<=mySystem.getUserRequirements().getOptimization().getMaxPrice()) {
			final UserSystem messageSystem=mySystem;
			try {
				Message jmsMessage =
jmsSession.createObjectMessage(messageSystem);
				jmsQueueProducer.send(jmsMessage);
			} catch (JMSException e) {
				e.printStackTrace();
			}
		}
	}
	@ResponseBody
	@RequestMapping(value = "/checkPriceCommon", method =
{RequestMethod.POST}, consumes = {"application/json", "application/xml"})
	public void handleCommonPriceRequest(@RequestBody UserSystem mySystem)
throws JMSException, JAXBException {

		//computation of TotalPrice
		totalPrice =
mySystem.getBusiness().getDatabase().getPrice()+mySystem.getBusiness().getOperat
ingSystem().getPrice()+mySystem.getBusiness().getCrm().getPrice()+mySystem.getBu
siness().getNetwork().getPrice();
		mySystem.setTotalPrice(totalPrice);

		if
(totalPrice<=mySystem.getUserRequirements().getOptimization().getMaxPrice()) {
			mySystem.setId(4);
			final UserSystem messageSystem=mySystem;
			try {
				Message jmsMessageCommon =
jmsSessionCommon.createObjectMessage(messageSystem);
				jmsCommonProducer.send(jmsMessageCommon);
			} catch (JMSException e) {
				e.printStackTrace();
			}
		}
	}
```

```
}
```

Η διαδικασία (process)μέσω της οποίας το Camel καλεί το web service που εκτελεί την Φάση της τελικής Λύσης (ResolutionPhase.java)

```
package edu.gr.ntua.sylvia.process;
import javax.annotation.Resource;
import org.springframework.stereotype.Component;
import edu.gr.ntua.sylvia.web.model.UserSystem;
import org.springframework.web.client.RestTemplate;

@Component
public class ResolutionPhase {

	@Resource
	private RestTemplate restTemplate;

	public void Resolution(UserSystem system) {

	restTemplate.postForObject("http://localhost:8080/web/resolution/finalChec
k", system, UserSystem.class);
	}

	public void CommonResolution(UserSystem system) {

	restTemplate.postForObject("http://localhost:8080/web/resolution/finalChec
kCommon", system, UserSystem.class);
	}
}
```

Ο controller που διαχειρίζεται το web service και εκτελεί τα αιτήματα του Camel στην Φάση της τελικής Λύσης (ResolutionController.java)

```
package edu.gr.ntua.sylvia.web.controller;
import java.io.File;
import javax.annotation.PostConstruct;
import javax.annotation.Resource;
import javax.jms.ConnectionFactory;
import javax.jms.JMSException;
import javax.xml.bind.JAXBException;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.Marshaller;
import org.springframework.jms.core.JmsTemplate;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.ResponseBody;
import edu.gr.ntua.sylvia.web.model.UserSystem;

@Controller
@RequestMapping("/resolution")
public class ResolutionController {

	private Integer totalSecurity;
	private Boolean totalConfidentiality;
	private Integer totalReliability;
	private Integer totalReputation;
	private Integer totalTotal;
	private Integer c;
```

```
	private Integer global=-9999;
	private String
END_PATH="C:/Users/Sylvie/Desktop/camelRouteEnd/finalResult.xml";
	private String END_PATH_COMMON=
"C:/Users/Sylvie/Desktop/camelRouteEndArchitecture2/finalResultCommon.xml";

	@Resource
	private ConnectionFactory connectionFactory;

	@Resource(name = "myJmsTemplate")
	private JmsTemplate jmsTemplate;

	@Resource(name = "myJmsTemplateCommon")
	private JmsTemplate jmsTemplateCommon;

private Marshaller jaxbMarshaller;

	@PostConstruct
	private void prepareMarshaller() throws JAXBException {
		JAXBContext jaxbContext =
JAXBContext.newInstance(UserSystem.class);
		jaxbMarshaller = jaxbContext.createMarshaller();
		jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
	}

	@ResponseBody
	@RequestMapping(value = "/finalCheck", method = {RequestMethod.POST},
consumes = {"application/json", "application/xml"})
	public void handleResolutionRequest(@RequestBody UserSystem mySystem)
throws JMSException, JAXBException {

		UserSystem finalSystem = computeTotal(mySystem);

		//Check if total values are good enough according to user's
requirements
		if
(finalSystem.getTotalConfidentiality().equalsIgnoreCase(finalSystem.getUserRequi
rements().getOptimization().getConfidentiality()) &&
(finalSystem.getTotalAvailability()>=finalSystem.getUserRequirements().getOptimi
zation().getMinAvailability()) &&
(finalSystem.getTotalSecurity()>=finalSystem.getUserRequirements().getOptimizati
on().getMinSecurity()) &&
(finalSystem.getTotalReliability()>=finalSystem.getUserRequirements().getOptimiz
ation().getMinReliability()) &&
finalSystem.getTotalReputation()>=finalSystem.getUserRequirements().getOptimizat
ion().getMinReputation()) {

			//Comparison of this TotalTotal and exit's TotalTotal and
keep the best TotalTotal
			if (global<finalSystem.getTotalTotal()) {
				global=finalSystem.getTotalTotal();
				jaxbMarshaller.marshal(finalSystem, new
File(END_PATH));
			}
		}
	}

	@ResponseBody
	@RequestMapping(value = "/finalCheckCommon", method =
```

```
{RequestMethod.POST}, consumes = {"application/json", "application/xml"})
	public void handleCommonResolutionRequest(@RequestBody UserSystem
mySystem) throws JMSException, JAXBException {

		UserSystem finalSystem = computeTotal(mySystem);

		//Check if total values are good enough according to user's
requirements
			if
(finalSystem.getTotalConfidentiality().equalsIgnoreCase(finalSystem.getUserRequi
rements().getOptimization().getConfidentiality()) &&
(finalSystem.getTotalAvailability()>=finalSystem.getUserRequirements().getOptimi
zation().getMinAvailability()) &&
(finalSystem.getTotalSecurity()>=finalSystem.getUserRequirements().getOptimizati
on().getMinSecurity()) &&
(finalSystem.getTotalReliability()>=finalSystem.getUserRequirements().getOptimiz
ation().getMinReliability()) &&
finalSystem.getTotalReputation()>=finalSystem.getUserRequirements().getOptimizat
ion().getMinReputation()) {

				//Comparison of this TotalTotal and exit's
TotalTotal and keep the best TotalTotal
				if (global<finalSystem.getTotalTotal()) {
					global=finalSystem.getTotalTotal();
					jaxbMarshaller.marshal(finalSystem, new
File(END_PATH_COMMON));
				}
			}
	}

	private UserSystem computeTotal(UserSystem userSystem) {
		//computation of TotalAvailability
		final Integer totalAvailability = (int)
((userSystem.getBusiness().getDatabase().getAvailability()+userSystem.getBusines
s().getOperatingSystem().getAvailability()+userSystem.getBusiness().getCrm().get
Availability()+userSystem.getBusiness().getNetwork().getAvailability())/4);
		userSystem.setTotalAvailability(totalAvailability);

		//computation of TotalSecurity
		totalSecurity=
userSystem.getBusiness().getDatabase().getSecurity();

		if
(totalSecurity>userSystem.getBusiness().getOperatingSystem().getSecurity()){

	totalSecurity=userSystem.getBusiness().getOperatingSystem().getSecurity();
		}
		if (totalSecurity>userSystem.getBusiness().getCrm().getSecurity())
{

	totalSecurity=userSystem.getBusiness().getCrm().getSecurity();
		}
		if
(totalSecurity>userSystem.getBusiness().getNetwork().getSecurity()) {

	totalSecurity=userSystem.getBusiness().getNetwork().getSecurity();
		}

		userSystem.setTotalSecurity(totalSecurity);
```

```
            totalConfidentiality =
((userSystem.getBusiness().getDatabase().getConfidentiality().equalsIgnoreCase("
YES")) &&
(userSystem.getBusiness().getOperatingSystem().getConfidentiality().equalsIgnore
Case("YES")) &&
(userSystem.getBusiness().getCrm().getConfidentiality().equalsIgnoreCase("YES"))
&&
(userSystem.getBusiness().getNetwork().getConfidentiality().equalsIgnoreCase("YE
S")));
            if (totalConfidentiality==true) {
                  userSystem.setTotalConfidentiality("YES");
                  c=1;
            }
            if (totalConfidentiality==false) {
                  userSystem.setTotalConfidentiality("NO");
                  c=0;
            }

            //computation of TotalReliability
            totalReliability =
(userSystem.getBusiness().getDatabase().getReliability()+userSystem.getBusiness(
).getOperatingSystem().getReliability()+userSystem.getBusiness().getCrm().getRel
iability())/4;
            userSystem.setTotalReliability(totalReliability);

            //computation of TotalReputation
            totalReputation=
(userSystem.getBusiness().getDatabase().getReputation()+userSystem.getBusiness()
.getOperatingSystem().getReputation()+userSystem.getBusiness().getCrm().getReput
ation()+userSystem.getBusiness().getNetwork().getReputation())/4;
            userSystem.setTotalReputation(totalReputation);

            //computation of TotalTotal
            totalTotal = (int) ((0.2 * userSystem.getTotalAvailability() + 0.1
* userSystem.getTotalSecurity() + 0.1 * userSystem.getTotalReputation() + 0.05 *
userSystem.getTotalReliability() + 0.05*c)*80- 0.5 *
userSystem.getTotalPrice());
            userSystem.setTotalTotal(totalTotal);
            return userSystem;
      }

}
```

Το pom.xml του project του Eclipse για τα web services

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>
      <groupId>edu.gr.ntua.sylvia</groupId>
      <artifactId>web</artifactId>
      <version>1.0-SNAPSHOT</version>

      <packaging>war</packaging>

      <properties>
            <slf4j-version>1.7.12</slf4j-version>
```

```
        <spring-version>4.2.1.RELEASE</spring-version>
        <jackson-version>2.6.1</jackson-version>
        <logback.version>1.1.3</logback.version>
    </properties>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.3</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring-version}</version>
            <exclusions>
                <exclusion>
                    <artifactId>commons-logging</artifactId>
                    <groupId>commons-logging</groupId>
                </exclusion>
            </exclusions>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.core</groupId>
            <artifactId>jackson-databind</artifactId>
            <version>${jackson-version}</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>${spring-version}</version>
        </dependency>

        <dependency>
            <groupId>ch.qos.logback</groupId>
            <artifactId>logback-classic</artifactId>
            <version>${logback.version}</version>
        </dependency>

        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>jcl-over-slf4j</artifactId>
            <version>${slf4j-version}</version>
        </dependency>

        <dependency>
            <groupId>org.apache.activemq</groupId>
            <artifactId>activemq-all</artifactId>
```

```
                <version>5.10.0</version>
            </dependency>

            <dependency>
                <groupId>org.springframework</groupId>
                <artifactId>spring-jms</artifactId>
                <version>${spring-version}</version>
            </dependency>

            <dependency>
                <groupId>commons-io</groupId>
                <artifactId>commons-io</artifactId>
                <version>2.4</version>
            </dependency>

        </dependencies>

</project>
```