ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ - ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ, ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΔΙΚΤΥΩΝ

Ανάπτυξη λογισμικού υλοποίησης του ανοικτού πρότυπου EPC ALE v1.1 για εφαρμογές RFID

Διπλωματική Εργασία
Marie-Aurélie Nef

Επιβλέπων
Παναγιώτης Δημητρόπουλος, Λέκτορας Π.Θ.

Δεύτερο Μέλος Επιτροπής
Γεώγριος Σταμούλης, Καθηγητής Π.Θ.

ΙΟΥΛΙΟΣ 2008
ΒΟΛΟΣ

Πρόλογος

Ο στόχος της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη λογισμικού υλοποίησης του ανοικτού πρότυπου EPC ALE v1.1 για εφαρμογές RFID. Για την επίτευξη αυτού του στόχου γίνεται μελέτη σχετικών τεχνολογιών προγραμματισμού καθώς επίσης και μία ανασκόπηση των EPCglobal Standards για την καλύτερη κατανόηση του πρότυπου ALE.

Η ύλη διαρθρώνεται σε δύο βασικές ενότητες :

Η πρώτη ενότητα περιέχει μία περιγραφή των τεχνολογιών που χρησιμοποιούνται για την υλοποίηση του πρότυπου EPC ALE 1.1. Κατανέμονται σε 4 κεφάλαια ως εξής :

- Στο 1^{ο} κεφάλαιο, εισάγεται η έννοια του XML και του XML Schema.
- Στο 2^{ο} κεφάλαιο, παρουσιάζεται η τεχνολογία Java Architecture for XML Binding (JAXB).
- Στο 3^{ο} κεφάλαιο, παρουσιάζονται τα Servlets.
- Στο 4^{ο} κεφάλαιο, παρουσιάζεται η τεχνολογία Java API for XML Web Services (JAX-WS)

Η δεύτερη ενότητα περιέχει μία παρουσίαση του RFID, των EPCglobal Standards και μία πιο λεπτομερή περιγραφή της Application Level Events (ALE). Κατανέμεται σε 2 κεφάλαια ως εξής :

- Στο 5^{ο} κεφάλαιο, γίνεται μία εισαγωγή στην τεχνολογία RFID και παρουσιάζονται τα EPCglobal Standards.
- Στο 6^{ο} κεφάλαιο, παρουσιάζεται η ALE.

Στο σημείο αυτό θα ήθελα να ευχαριστήσω τον Λέκτορα Παναγιώτη Δημητρόπουλο και τον Καθηγητή Γεώργιο Σταμούλη για την καθοδήγηση και την βοήθειά τους σε όλη την διάρκεια της φοίτησής μου και για την ολοκλήρωση αυτής της διπλωματικής εργασίας. Επίσης θα ήθελα να ευχαριστήσω την οικογένειά μου και τους φίλους μου που ήταν πάντα δίπλα μου.

Ιούλιος 2008

Marie-Aurélie Nef

Περιεχόμενα

1 XML και XML Schema

1.1 XML

1.1.1 Εισαγωγή

Το XML (Extensible Markup Language) είναι μία προδιαγραφή (specification) γενικού σκοπού που επιτρέπει την δημιουργία μίας γλώσσας σήμανσης (markup language) προσαρμοσμένη στις ανάγκες μας. Παρέχει την δυνατότητα δόμησης της πληροφορίας και είναι platform independent.

Ένα XML αρχείο είναι "καλά – διαμορφωμένο" (well-formed) αν ακολουθεί τους συντακτικούς κανόνες ορισμένοι στην XML 1.0 specification. Πιο συγκεκριμένα, ένα XML αρχείο αποτελείται από ένα πρόλογο (header) και ένα δέντρο στοιχείων (elements).

1.1.2 Header

Ο πρόλογος περιέχει την XML δήλωση και μία προαιρετική αναφορά σε εξωτερικά αρχεία δόμησης. Για παράδειγμα:

```
<?xml version="1.0" encoding="UTF-16" standalone="no" ?>
```

Δηλώνει ότι το αρχείο αυτό είναι τύπου XML, ποια έκδοση χρησιμοποιείται, την κωδικοποίηση χαρακτήρων και αν υπάρχει ή όχι αναφορά σε εξωτερικά αρχεία δόμησης. Μία τέτοια αναφορά χρησιμοποιεί μία από τις δύο υπάρχουσες λέξεις-κλειδί : SYSTEM ή PUBLIC. Η SYSTEM χρησιμοποιείται όταν η αναφορά γίνεται σε τοπικό αρχείο ή με την χρήση ενός URL. Για παράδειγμα :

```
<!DOCTYPE example SYSTEM "example.dtd">
```

Αντίστοιχα, η PUBLIC χρησιμοποιείται όταν υπάρχει μία κοινή αναφορά που συνοδεύεται με ένα standard DTD ορισμένο από ένα consortium. Για παράδειγμα :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

1.1.3 Elements

Κάθε στοιχείο αποτελείται από ένα opening tag, το περιεχόμενό του (προαιρετικό) και ένα closing tag. Για παράδειγμα:

```
<element1> περιεχόμενο </element1>
```

Αν δεν υπάρχει περιεχόμενο, το στοιχείο ονομάζεται άδειο (empty) και έχει την εξής μορφή:

```
<element1></element1>
```

ή σε συντόμευση:

```
<element1/>
```

Κάθε άδειο στοιχείο δεν είναι απαραίτητα ασήμαντο διότι μπορεί, όπως κάθε κανονικό στοιχείο, να περιέχει και ιδιότητες που ονομάζονται χαρακτηριστικά (attributes). Το χαρακτηριστικό αποτελείται από ένα ζευγάρι όνομα-τιμή (name-value) και βρίσκεται στο opening tag του στοιχείου.
Παράδειγμα για ένα άδειο στοιχείο :

```
<element1  attribute1="value1"  attribute2="value2"/>
```

Παράδειγμα για κανονικό στοιχείο :

```
<element1  attribute1="value1"  attribute2="value2">
      < element2  attribute3="value3"/>
      < element2  attribute3="value4"/>
</element1>
```

Ενά στοιχείο μπορεί να περιέχει αναφορά προς άλλα στοιχεία χρησιμοποιώντας αναγνωριστή (identifier) ή προς άλλους τύπους χρησιμοποιώντας χαρακτηριστικό τύπου "type".

1.2 XML Schema

1.2.1 Εισαγωγή

Ένα XML αρχείο είναι καλά-διαμορφωμένο αν τηρεί ορισμένους συντακτικούς κανόνες. Όμως, αυτοί οι κανόνες δεν προσδιορίζουν την δομή και τους περιορισμούς (constraints) των δεδομένων. Όταν δύο εφαρμογές θέλουν να επικοινωνήσουν μεταξύ τους, πρέπει να χρησιμοποιούν το ίδιο λεξιλόγιο. Για να γίνει αυτό, πρέπει να οριστούν όσα στοιχεία και όσα χαρακτηριστικά είναι πιθανό να χρησιμοποιηθούν. Εκτός αυτού, πρέπει να οριστεί και η δομή τους καθώς επίσης και όσοι περιορισμοί υπάρχουν. Για παράδειγμα, ποιες τιμές

μπορεί να λαμβάνει ένα χαρακτηριστικό, ποια στοιχεία μπορεί ή πρέπει να υπάρχουν μέσα σε κάποιο άλλο στοιχείο κλπ.

Αν υπάρχει ο ορισμός αυτής της δομής, προσθέτεται και η δυνατότητα επικύρωσης (validation) του XML αρχείου. Ένα XML αρχείο είναι επικυρωμένο αν είναι καλά-διαμορφωμένο, αν χρησιμοποιεί αρχείο δόμησης και τηρεί αυτούς τους κανόνες δόμησης.

Υπάρχουν δύο τρόποι να οριστεί η δομή ενός XML αρχείου: τα DTDs (Document Type Definitions), ο παλιότερος και πιο περιορισμένος τρόπος, και το XML Schema, που προσφέρει περισσότερες δυνατότητες και στον οποίο θα εστιάσουμε.

1.2.2 Συντακτικό του XML Schema

Το συντακτικό του XML Schema βασίζεται στην XML. Έτσι, ένα XML Schema είναι ένα στοιχείο με πιθανό opening tag:

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

Το πρόθεμα xs δηλώνει το namespace αυτού του σχήματος.
Στην συνέχεια ακολουθούν τα στοιχεία του σχήματος με την δήλωση των τύπων στοιχείων και χαρακτηριστικών.

1.2.3 Τύποι στοιχείων

Υποχρεωτικά χαρακτηριστικά :

- `name` : όνομα στοιχείου

Προεραιτικά χαρακτηριστικά :

- `type` : τύπος στοιχείου (βλ. παρακάτω)
- περιορισμοί τύπου cardinality :
  - `minOccurs` : ελάχιστος αριθμός εμφανίσεων
  - `maxOccurs` : μέγιστος αριθμός εμφανίσεων

Παράδειγμα :

```
<xs:element name="example" type="string" minOccurs="0"
 maxOccurs="unbounded"/>
```

1.2.4 Τύποι χαρακτηριστικών

Υποχρεωτικά χαρακτηριστικά :

- `name` : όνομα στοιχείου

Προεραιτικά χαρακτηριστικά :

- `type` : τύπος στοιχείου (βλ. παρακάτω)
- `use` : δηλώνει την ύπαρξη, υποχρεωτική ή προαιρετική
- `use` & `value`: δηλώνει την προκαθορισμένη ή σταθερή τιμή `value`

Παραδείγματα :

```
<xs:attribute name="id" type="ID" use="required"/>

<xs:attribute name="speaks" type="Language" use="default" value="en"/>
```

1.2.5 Τύποι δεδομένων

Το XML Schema παρέχει πολλές δυνατότητες για τον ορισμό τύπων δεδομένων.
Υπάρχουν οι ενσωματωμένοι (built-in) τύποι δεδομένων, όπως για παράδειγμα :

- Αριθμητικοί τύποι δεδομένων : `integer`, `Short`, `Byte`, `Long`, `Decimal`, `Float` κλπ.
- Αλφαριθμητικοί τύποι δεδομένων : `string`, `ID`, `IDREF`, `CDATA`, `Language` κλπ.
- Χρονολογικοί τύποι δεδομένων : `time`, `Date`, `Month`, `Year` κλπ.

Επίσης, υπάρχουν και οι τύποι δεδομένων ορισμένοι από τον χρήστη (user-defined).
Υπάρχουν δύο κατηγορίες :

- Απλοί τύποι δεδομένων (simple) : δεν χρησιμοποιούν στοιχεία ή χαρακτηριστικά
- Σύνθετοι τύποι δεδομένων (complex) : δηλώνονται από ήδη υπάρχοντες τύποι δεδομένων χρησιμοποιώντας χαρακτηριστικά και τα εξής :
  - `sequence` : μία ακολουθία υπαρχόντων τύπων δεδομένων όπου η σειρά με την οποία εμφανίζονται έχει σημασία.
  - `all` : μία συλλογή στοιχείων που πρέπει να εμφανίζονται με οποιαδήποτε σειρά.
  - `choice` : μία συλλογή στοιχείων από την οποία πρέπει να διαλεχτεί ένα.

Παράδειγμα σύνθετου τύπου δεδομένων :

```
<xs:complexType name="complexType1">
   <xs:sequence>
   <xs:element name="elem1" type="string" minOccurs="0"
    maxOccurs="unbounded"/>
   <xs:element name="elem2" type="string"/>
   </xs:sequence>
   <xs:attribute name="attr1" type="string" use="optional"/>
</xs:complexType>
```

2 Java Architecture for XML Binding (JAXB)

2.1 Εισαγωγή : JAXP vs JAXB

Όταν μία Java εφαρμογή χρησιμοποιεί XML δεδομένα βασισμένα σε ένα ή περισσότερα σχήματα, το πρόγραμμα θα πρέπει να δημιουργήσει, να διαβάσει και να χειριστεί αρχεία που θα είναι επίσης βασισμένα στα ίδια σχήματα.

Ένας τυπικός τρόπος να γίνει αυτό θα ήταν να χρησιμοποιηθούν parsers που συμβιβάζονται με το Simple API for XML (SAX) ή το Document Object Model (DOM) που παρέχονται στο Java API for XML Processing (JAXP).

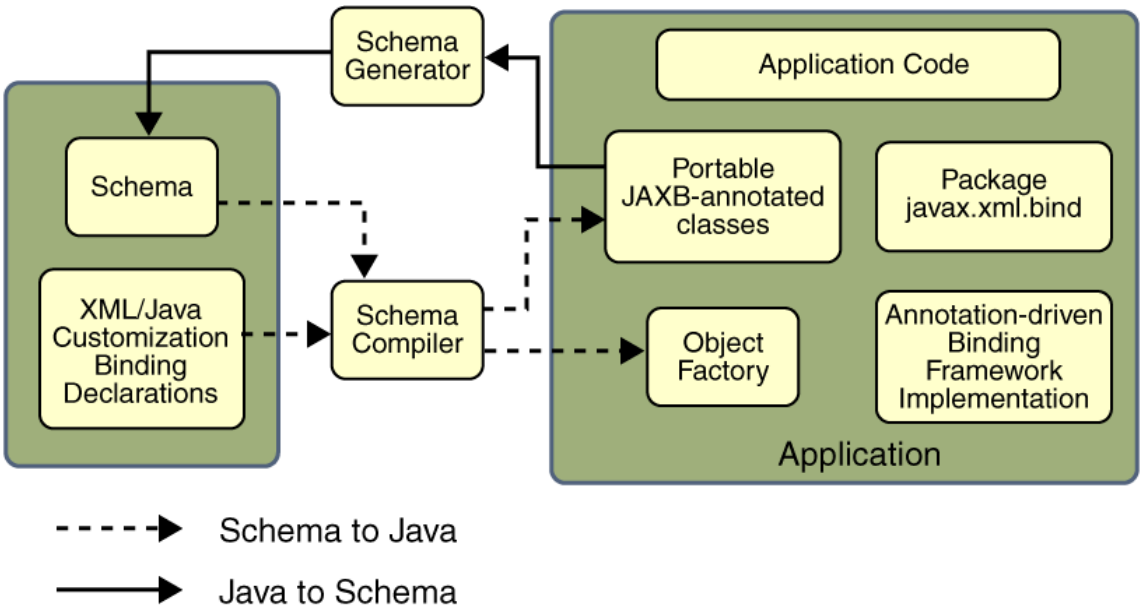
Στην SAX προσέγγιση, το parser ξεκινά στην αρχή του αρχείο και περνά κάθε κομμάτι του αρχείου στην εφαρμογή με την σειρά που τα διαβάζει. Τίποτα δεν αποθηκεύεται στην μνήμη. Η εφαρμογή μπορεί να χρησιμοποιήσει τα δεδομένα που λαμβάνει από τον parser, όμως δεν μπορεί να γίνει επεξεργασία μέσα στην μνήμη. Για παράδειγμα, δεν μπορεί να γίνει μία ενημέρωση δεδομένων μέσα στην μνήμη και να αποθηκευτεί αυτή η αλλαγή πίσω στο XML αρχείο.

Στην DOM προσέγγιση, το parser δημιουργεί ένα δέντρο αντικειμένων που αναπαριστούν το περιεχόμενο και την δομή των δεδομένων του αρχείου. Στην περίπτωση αυτή, το δέντρο υπάρχει στην μνήμη. Η εφαρμογή έπειτα να διαπεράσει το δέντρο να προσπελάσει τα δεδομένα που χρειάζεται και να τα χειριστεί κατάλληλα.

Στις δύο παραπάνω προσεγγίσεις, είναι δύσκολο να διατηρηθεί ο κώδικας της εφαρμογής όσο τα σχήματα εξελίσσονται. Επίσης, θα ήταν πολύ πιο εύκολο να γραφτούν εφαρμογές που χρησιμοποιούν XML αν υπήρχε τρόπος να απεικονίσουμε κατάλληλα τα δεδομένα του XML αρχείου, χρησιμοποιώντας το αντίστοιχο σχήμα, σε αντικείμενα (Objects) που θα βρίσκονται στην μνήμη. Οι κλάσεις αυτών των αντικειμένων θα αποτελούταν από υπάρχουσες κλάσεις για τους κοινούς τύπους δεδομένων, καθώς επίσης και από άλλες ειδικές κλάσεις προσαρμοσμένες στο σχήμα. Αυτή η διαδικασία παρέχεται από το Java Architecture for XML Binding (JAXB) που επιτρέπει την δημιουργία ενός Java object-level binding του σχήματος.

2.2 JAXB Αρχιτεκτονική

Στην εικόνα 2.1 παρουσιάζονται τα στοιχεία που αποτελούν μία JAXB υλοποίηση.



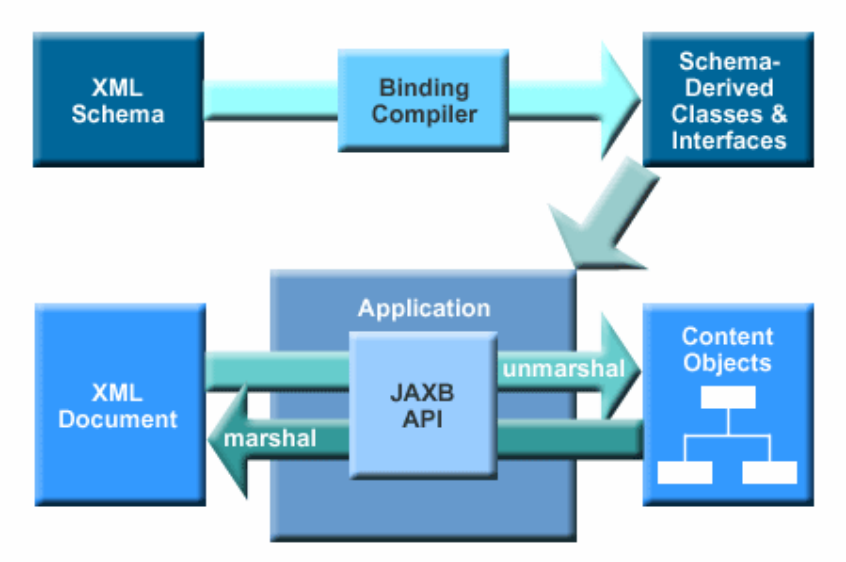
Εικόνα 2.1 : συνοπτική JAXB αρχιτεκτονική

Μία JAXB υλοποίηση αποτελείται από τα παρακάτω αρχιτεκτονικά στοιχεία :

- Schema compiler : κάνει bind ένα αρχικό σχήμα (source schema) σε ένα σύνολο στοιχείων προγράμματος παραγόμενα από το σχήμα (schema derived program elements). Το binding περιγράφεται με μία γλώσσα βασισμένη στην XML, binding language.
- Schema generator : κάνει map ένα σύνολο υπαρχόντων στοιχείων προγράμματος σε ένα derived schema. Αυτό το mapping περιγράφεται με τα σχόλια προγράμματος (program annotations).
- Binding runtime framework : παρέχει τις δύο βασικές λειτουργίες για προσπέλαση, χειρισμό και επικύρωση του XML περιεχομένου χρησιμοποιώντας είτε το schema derived είτε τα υπάρχοντα στοιχεία προγράμματος :
    - Unmarshalling : διαδικασία διαβάσματος ενός XML αρχείου και δημιουργίας ενός content objects δέντρου με τα περιεχόμενα του αρχείου. Η επικύρωση μπορεί, προαιρετικά, να γίνει στην διαδικασία unmarshalling.
    - Marshalling : αντίστροφο του unmarshalling, δηλαδή, είναι η διαδικασία διαπέρασης του δέντρου και δημιουργίας ενός XML αρχείο που απεικονίζει το περιεχόμενο του δέντρου αυτού. Η επικύρωση μπορεί, προαιρετικά, να γίνει στην διαδικασία marshalling.

2.3 Διαδικασία JAXB Binding

Στην εικόνα 2.2 παρουσιάζονται τα βήματα της διαδικασίας JAXB Binding



Εικόνα 2.2 : Διαδικασία JAXB Binding

Η βασική διαδικασία είναι η εξής :

i. *Δημιουργία κλάσεων*. Ένα XML σχήμα χρησιμοποιείται ως είσοδο στο JAXB binding compiler για την δημιουργία κλάσεων βασισμένες σε αυτό το σχήμα.
ii. *Μεταγλώττιση κλάσεων.* Όλες οι παραγόμενες κλάσεις, πηγαίων αρχείων και κώδικας εφαρμογής πρέπει να μεταγλωττίζονται.
iii. *Unmarshal.* Γίνεται unmarshal με το JAXB binding framework χρησιμοποιώντας XML αρχεία αλλά ακόμα και DOM nodes, string buffers, SAX Sources, κ.ο.κ.
iv. *Επικύρωση (προαιρετική).*
v. *Δημιουργία δέντρου.* Αναπαριστά την δομή και το περιεχόμενο των XML αρχείων.
vi. *Επεξεργασία περιεχομένου.* Η client εφαρμογή μπορεί να αλλάξει την XML πληροφορία απεικονισμένη στο δέντρο χρησιμοποιώντας την διεπαφή που έχει παραχθεί από το binding compiler.
vii. *Marshal.* Γίνεται marshal του τελικού δέντρου του βήματος vi. και παράγονται ένα ή περισσότερα XML αρχεία. Μπορεί προαιρετικά να γίνει επικύρωση πριν το marshalling.

2.4 Επικύρωση

Υπάρχουν τρεις τρόποι επικύρωσης στην αρχιτεκτονική JAXB :

- Unmarshal-time validation
  Η επικύρωση αυτή επιτρέπει στην client εφαρμογή να ενημερώνεται για τα λάθη και τις προειδοποιήσεις κατά την διαδικασία unmarshal του XML αρχείο σε Java content δέντρο.
- On-demand validation
  Αυτή η μέθοδος είναι πλέον deprecated στην JAXB 2.0.
- Fail-fast validation
  Η επικύρωση αυτή επιτρέπει στην client εφαρμογή να λαμβάνει άμεση ανάδραση για μία τυχόν αλλαγή στο Java content δέντρο που παραβιάζει κάποιο περιορισμό.

2.5 Bindings τύπων δεδομένων

2.5.1 XML Schema σε Java

Ο πίνακας 2.1 παρουσιάζει το binding που γίνεται από XML Schema σε Java.

XML Schema Τύπος	Java Τύπος δεδομένων
xsd:string	java.lang.String
xsd:integer	java.math.BigInteger
xsd:int	int
xsd:long	long
xsd:short	short
xsd:decimal	java.math.BigDecimal
xsd:float	float
xsd:double	double
xsd:boolean	boolean
xsd:byte	byte
xsd:QName	javax.xml.namespace.Qname
xsd:dateTime	javax.xml.datatype.XMLGregorianCalendar
xsd:base64Binary	byte[]
xsd:hexBinary	byte[]
xsd:unsignedInt	long
xsd:unsignedShort	int
xsd:unsignedByte	short
xsd:time	javax.xml.datatype.XMLGregorianCalendar

xsd:date	javax.xml.datatype.XMLGregorianCalendar
xsd:g	javax.xml.datatype.XMLGregorianCalendar
xsd:anySimpleType	java.lang.Object
xsd:anySimpleType	java.lang.String
xsd:duration	javax.xml.datatype.Duration
xsd:NOTATION	javax.xml.namespace.Qname

Πίνακας 2.1 : binding από XML Schema σε Java

2.5.2 Java σε XML Schema

Ο πίνακας 2.2 παρουσιάζει το binding που γίνεται από Java σε XML Schema.

Java Κλάση	XML Τύπος δεδομένων
java.lang.String	xs:string
java.math.BigInteger	xs:integer
java.math.BigDecimal	xs:decimal
java.util.Calendar	xs:dateTime
java.util.Date	xs:dateTime
javax.xml.namespace.Qname	xs:Qname
java.net.URI	xs:string
javax.xml.datatype.XMLGregorianCalendar	xs:anySimpleType
javax.xml.datatype.Duration	xs:duration
java.lang.Object	xs:anyType
java.awt.Image	xs:base64Binary
javax.activation.DataHandler	xs:base64Binary
javax.xml.transform.Source	xs:base64Binary
java.util.UUID	xs:string

Πίνακας 2.2 : binding από Java σε XML Schema

2.6 Παράδειγμα Υλοποίησης

2.6.1 Εισαγωγή

Για αυτό το παράδειγμα, θα χρησιμοποιήσουμε το αρχείο data.xsd από τον οποίο θα γίνει κατασκευή των κλάσεων Data.java, Report.java,TagList.java καθώς επίσης και μία κλάση ObjectFactory.java που χρησιμοποιείται κατά την δημιουργία αντικειμένων των τριών παραπάνω κλάσεων. Οι τέσσερις κλάσεις αυτές βρίσκονται σε ένα Package με όνομα data.

Για την υλοποίηση των λειτουργιών unmarshal και marshal όλων των σχημάτων που χρησιμοποιούνται στο πρόγραμμα, χρησιμοποιούμε δύο βοηθητικές κλάσεις Parser.java και Serializer.java αντίστοιχα που περιέχουν static μεθόδους. Για το παράδειγμά μας, θα εστιάσουμε μόνο στις μεθόδους που αφορούν το σχήμα data.

Επίσης, χρησιμοποιούμε μία κλάση MyValidationEventHandler που κάνει implement την κλάση ValidationEventHandler για τον ορισμό των EventHandler των unmarshaller και marshaller.

2.6.2 Υλοποίηση του parser

Κλάση Parser.java :

```
package util;

import static javax.xml.XMLConstants.W3C_XML_SCHEMA_NS_URI;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import org.xml.sax.SAXException;

import data.*;

public class Parser {

    public static Data parseData(String dataXML,URI dataXSDURI) {

      Data d = null;
      InputStream inputStream = new
            ByteArrayInputStream(dataXML.getBytes());

      try {
            JAXBContext jaxbContent = JAXBContext.newInstance("data");
            Unmarshaller unmarshaller = jaxbContent.createUnmarshaller();
            if(dataXSDURI!=null){
                  SchemaFactory schemaFactory =
                        SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI);
                  Schema schema = schemaFactory.newSchema(new
                        File(dataXSDURI.getPath()));
                  unmarshaller.setSchema(schema);
            }
            unmarshaller.setEventHandler(new MyValidationEventHandler());
            d = (Data)unmarshaller.unmarshal(inputStream);
            inputStream.close();

      } catch (JAXBException e) {
            e.printStackTrace();
```

```
        } catch (SAXException e) {
              e.printStackTrace();
        } catch (IOException e) {
              e.printStackTrace();
        }
        return d;
    }

    /* some other methods for other schemas */

}
```

Σύντομη ανάλυση βασικών λειτουργιών

Η μέθοδος parseData παίρνει ως παραμέτρους :

- `String dataXML` : το XML αρχείο ως String
- `URI dataXSDURI` : ένα προαιρετικό URI για την επικύρωση του XML αρχείου

και επιστρέφει ένα αντικείμενο τύπου Data.

Δημιουργούμε ένα InputStream με το XML αρχείο που δόθηκε σαν πρώτη παράμετρος :

```
InputStream inputStream = new ByteArrayInputStream(dataXML.getBytes());
```

Με το κατάλληλο χειρισμό εξαιρέσεων, δημιουργούμε ένα αντικείμενο τύπου JAXBContext. Αυτό το αντικείμενο μας δίνει πρόσβαση στο JAXB API. Παίρνει σαν παράμετρο το όνομα του package που δημιουργήθηκε από το binding compiler, στην περίπτωση μας : data

```
JAXBContext jaxbContent = JAXBContext.newInstance("data");
```

Δημιουργούμε τον umarshaller :

```
Unmarshaller unmarshaller = jaxbContent.createUnmarshaller();
```

Σε περίπτωση που υπάρχει URI για την επικύρωση των δεδομένων, παραμετροποιούμε την διαδικασία κατάλληλα, επίσης ορίζουμε και το EventHandler :

```
if(dataXSDURI!=null){
  SchemaFactory schemaFactory =
      SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI);
  Schema schema = schemaFactory.newSchema(new File(dataXSDURI.getPath()));
  unmarshaller.setSchema(schema);
}
unmarshaller.setEventHandler(new MyValidationEventHandler());
```

Καλούμε την μέθοδο unmarshal για το InputStream που είχαμε δημιουργήσει και επιστρέφει ένα αντικείμενο τύπου Data :

```
d = (Data)unmarshaller.unmarshal(inputStream);
```

2.6.3 Υλοποίηση του serializer

Κλάση Serializer.java :

```
package util;

import static javax.xml.XMLConstants.W3C_XML_SCHEMA_NS_URI;

import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.OutputStream;
import java.net.URI;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.PropertyException;
import javax.xml.validation.Schema;
import javax.xml.validation.SchemaFactory;
import org.xml.sax.SAXException;
import data.*;

public class Serializer {

      public static String serializeData(Data data, URI dataXSDURI) {

        String dataXML = null;
        OutputStream outputStream = new ByteArrayOutputStream();
        try {
            JAXBContext jaxbContent = JAXBContext.newInstance("data");
            Marshaller marshaller = jaxbContent.createMarshaller();
            if(dataXSDURI!=null){
                  SchemaFactory schemaFactory =
                      SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI);
                  Schema schema = schemaFactory.newSchema(new
                      File(dataXSDURI.getPath()));
                  marshaller.setSchema(schema);
            }
            marshaller.setEventHandler(new MyValidationEventHandler());
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
               Boolean.TRUE);
            marshaller.marshal(data, outputStream);
            dataXML = outputStream.toString();
            outputStream.close();

        } catch (PropertyException e) {
            e.printStackTrace();
        } catch (JAXBException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (SAXException e) {
            e.printStackTrace();
        }
        return dataXML;
      }
    /* some other methods for other schemas */
}
```

Σύντομη ανάλυση βασικών λειτουργιών

Η μέθοδος serializeData παίρνει ως παραμέτρους :

- `Data data` : το αντικείμενο που θα κάνουμε marshal
- `URI dataXSDURI` : ένα προαιρετικό URI για την επικύρωση του XML αρχείου

και επιστρέφει ένα String που είναι το XML αρχείο που δημιουργήθηκε.

Δημιουργούμε το String και το OutputStream που θα περιέχουν το αποτέλεσμα της διαδικασίας marshal :

```
String dataXML = null;
OutputStream outputStream = new ByteArrayOutputStream();
```

Με το κατάλληλο χειρισμό εξαιρέσεων, δημιουργούμε ένα αντικείμενο τύπου JAXBContext. Παίρνει σαν παράμετρο το όνομα του package που δημιουργήθηκε από το binding compiler, στην περίπτωση μας : data

```
JAXBContext jaxbContent = JAXBContext.newInstance("data");
```

Δημιουργούμε το marshaller :

```
Marshaller marshaller = jaxbContent.createMarshaller();
```

Σε περίπτωση που υπάρχει URI για την επικύρωση των δεδομένων, παραμετροποιούμε την διαδικασία κατάλληλα, επίσης ορίζουμε και το EventHandler :

```
if(dataXSDURI!=null){
        SchemaFactory schemaFactory =
            SchemaFactory.newInstance(W3C_XML_SCHEMA_NS_URI);
        Schema schema = schemaFactory.newSchema(new
            File(dataXSDURI.getPath()));
        marshaller.setSchema(schema);
}
marshaller.setEventHandler(new MyValidationEventHandler());
```

Παραμετροποιούμε το marshaller έτσι ώστε να υπάρχει αλλαγή γραμμής και κατάλληλη διάταξη στο αποτέλεσμα της διαδικασίας :

```
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
```

Καλούμε την μέθοδο marshal για το αντικείμενο Data που πήραμε ως όρισμα και δίνουμε το OutpuStream που δημιουργήσαμε για την επιστροφή των αποτελεσμάτων :

```
marshaller.marshal(data, outputStream);
```

Μετατρέπουμε το OutputStream σε String που αποτελεί το τελικό αποτέλεσμα :

```
dataXML = outputStream.toString();
```

3 Servlets

3.1 Εισαγωγή

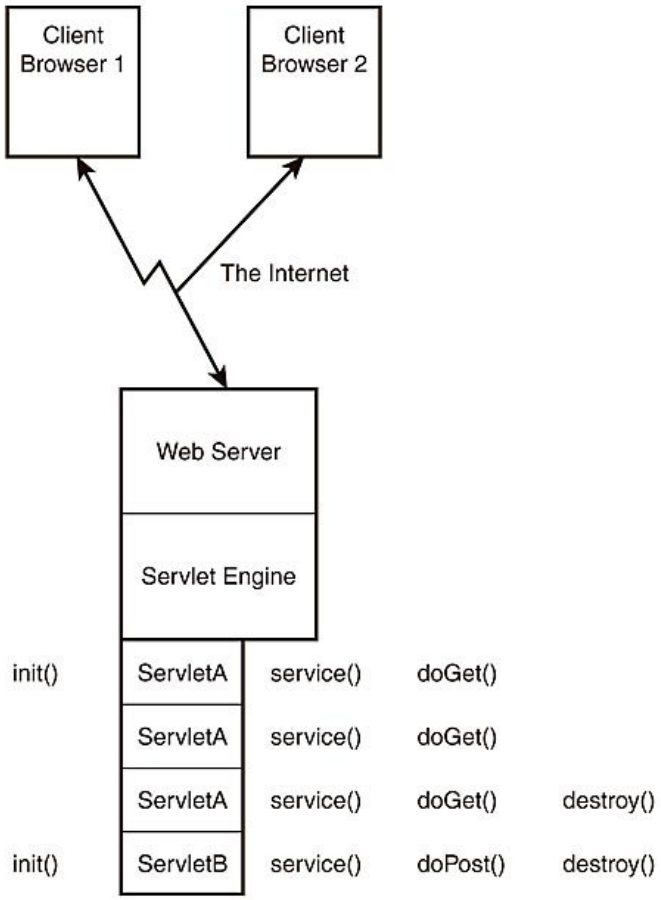
Οι servlets είναι μικρά προγράμματα που καλούνται από ένα άλλο πρόγραμμα που ονομάζεται container. Ο container μεσολαβεί στην επικοινωνία μεταξύ servlet και έξω κόσμου. Οι servlets δεν αποτελούν Java εφαρμογή διότι χρειάζεται να κληθούν μέσα στο containter, όμως σχεδόν κάθε υπάρχουσα λειτουργικότητα μιας Java εφαρμογής μπορεί να προστεθεί σε ένα servlet.

Μία άλλη λύση για την επικοινωνία με την χρήση πρωτοκόλλου HTTP είναι η χρήση HTTP server-side εφαρμογών που επεξεργάζουν κατευθείαν τις δικές τους αιτήσεις. Όμως, οι servlet containers μας επιτρέπουν να μην χρειάζεται να περιέχεται ολόκληρο το HTTP header processing code σε κάθε πρόγραμμα. Επίσης, οι servlets κλιμακώνουν με ικανοποιητικό τρόπο διότι αντί να δημιουργούνται νέες διεργασίες (processes), όπως γίνεται για κάθε HTTP εφαρμογή, ο servlet container δημιουργεί νήματα (threads) όποτε χρειάζεται και έχει την δυνατότητα να επαναχρησιμοποιήσει κάποιο νήμα αντί να το απορρίψει.

3.2 Λειτουργία των Servlets

Ο servlet container είναι υπεύθυνο για την instantiation του servlet και την κλήση των κατάλληλων μεθόδων την κατάλληλη στιγμή.

Η εικόνα 3.1 παρουσιάζει ένα παράδειγμα της λειτουργίας αυτής



Εικόνα 3.1 : Παράδειγμα λειτουργίας

i. Ένας χρήστης πληκτρολογεί το URL του servlet που θέλει να καλέσει.
ii. Το browser του δημιουργεί μία κατάλληλη αίτηση προς το αντίστοιχο Web Server.
iii. Ο server λαμβάνει την αίτηση και την προωθεί στο servlet container.
iv. Ο servlet container ελέγχει αν υπάρχουν ήδη κάποια instance του servlet αυτού στην μνήμη. Αν όχι, φορτώνει μία instance και τρέχει την init() μέθοδο του servlet.
v. Ο servlet container περιμένει να επιστρέψει η init() μέθοδος και στην συνέχεια καλεί την service() μέθοδο του servlet για την δημιουργία νέου νήματος.
vi. Η service() μέθοδος καλεί την doGet() ή την doPost() μέθοδο ανάλογα με το τύπο της HTTP αίτηση (GET ή POST).
vii. Ένας δεύτερος χρήστης καλεί τον ίδιο servlet.
viii. Ο servlet container διαπιστώνει ότι υπάρχει ήδη instance του servlet στην μνήμη και δημιουργεί ένα άλλο νήμα του ίδιου servlet.
ix. Γίνεται κλήση της service() μέθοδο και στην συνέχεια, της doGet() ή της doPost() όπως στο βήμα vi.
x. Το πρώτο νήμα τελειώνει και επιστρέφει απάντηση στον Web Server, ο οποίος την προωθεί στο browser του πρώτου χρήστη.
xi. Το δεύτερο νήμα τελειώνει και επιστρέφει απάντηση στον Web Server, ο οποίος την προωθεί στο browser του δεύτερου χρήστη.

xii. Κάποια στιγμή στο μέλλον, ο servlet container αποφασίζει να κάνει deinstantiate το servlet και καλεί την destroy() μέθοδο του servlet για να ελευθερωθεί η μνήμη.

3.3 Προγραμματισμός ενός servlet

Για την δημιουργία των servlets, αντί να χρησιμοποιηθούν java κλάσεις που ξεκινούν με την κλήση της main μεθόδου, οι servlets είναι extensions άλλων κλάσεων που ξέρουν ήδη πως να αλληλεπιδρούν με τους Web servers και servlet containers.

Οι παρακάτω κλάσεις και διεπαφές είναι οι σημαντικότερες κατά την ανάπτυξη μίας εφαρμογής που χρησιμοποιεί servlets :

- Servlet : Η διεπαφή αυτή ορίζει τις init(), service() και destroy() μεθόδους. Επίσης, ορίζει δύο επιπλέον μεθόδους που οι implementing κλάσεις πρέπει να παρέχουν, getServletConfig() και getServletInfo(), που επιτρέπουν τις αιτήσεις που αφορούν τον ίδιο το servlet (δημιουργός, έκδοση, copyright, κ.ο.κ.). Ο servlet containers είναι προγραμματισμένος έτσι ώστε να περιμένει από ένα servlet να υλοποιεί αυτές τις 5 μεθόδους.
- GenericServlet : Η κλάση αυτή προσφέρει μία απλή υλοποίηση της διεπαφής Servlet. Ονομάζεται "Generic" διότι δεν υποθέτει ότι θα επεξεργάσει αιτήσεις τύπου http και έτσι επιτρέπει στον προγραμματιστεί να χρησιμοποιήσει οποιοδήποτε πρωτόκολλο. Σε αυτήν την περίπτωση, γίνεται override της service() μεθόδου.
- HttpServlet : Η κλάση αυτή είναι η πιο συχνά extended όταν γράφονται servlet εφαρμογές. Κάνει extend την GenericServlet και προσφέρει λειτουργία επεξεργασίας http αιτήσεων. Σε αυτήν την περίπτωση, γίνεται override των μεθόδων doGet() και doPost(). Η κλάση περιέχει υλοποίηση της μεθόδου service() που καλεί την doGet() ή την doPost() ανάλογα με την αίτηση που έλαβε.
- ServletRequest : Η διεπαφή αυτή ορίζει ένα αντικείμενο που περιέχει την πληροφορία που έστειλε ο χρήστης με την αίτησή του. Μία υλοποίηση αυτής της διεπαφής αποτελεί η κλάση ServletRequestWrapper.
- HttpServletRequestWrapper : Αποτελεί extension της ServletRequestWrapper κλάση, η πληροφορίες που διαθέτει υποθέτουν μία http αίτηση.
- ServletResponse : Η διεπαφή αυτή ορίζει τον τρόπο με τον οποίο ο servlet θα στείλει στον servlet container την πληροφορία που θέλει να στείλει στον client. Μία υλοποίηση αυτής της διεπαφής αποτελεί η κλάση ServletResponseWrapper.
- HttpServletResponseWrapper : Αποτελεί extension της ServletResponseWrapper κλάση, υποθέτει ότι η απάντηση θα έχει http μορφή.

3.4 Παράδειγμα υλοποίησης

3.4.1 Εισαγωγή

Στο παράδειγμα αυτό, θα αναπτύξουμε ένα Servlet και μία Client εφαρμογή. Η Client εφαρμογή θα έχει δύο επιλογές: να επικοινωνήσει με το Servlet με μία HTTP αίτηση τύπου GET ή με μία HTTP αίτηση τύπου POST.

Στην GET περίπτωση (η `boolean` μεταβλητή `type` στο κώδικα έχει τιμή `true`), θα στέλνει τρεις παραμέτρους στο Servlet που θα επιστρέφει πίσω στον client μία αριθμημένη λίστα ονομάτων και τιμών των παραμέτρων.

Στην POST περίπτωση (η `boolean` μεταβλητή `type` στο κώδικα έχει τιμή `false`), θα χρησιμοποιήσουμε την JAXB τεχνολογία και την υλοποίηση του παραδείγματος του υποκεφαλαίου ..... Στο σχήμα data και στο αντίστοιχο Package, θα προσθέσουμε ένα σχήμα person.xsd και τις παραγώμενες κλάσεις Person.java και ObjectFactory.java που βρίσκονται στο Package με όνομα mydata. Επίσης, στις κλάσεις Parser.java και Serializer.java έχουν προστεθεί οι αντίστοιχες μεθόδους για την διαδικασία marshalling και unmarshalling για το σχήμα person.
Η τελική διαδικασία που εφαρμόζεται είναι η εξής (η αντίστοιχη αρίθμηση των βημάτων βρίσκεται σαν σχόλιο στο κώδικα που βρίσκεται παρακάτω):

i. Η Client εφαρμογή δημιουργεί ένα αντικείμενο τύπου Person αν η `boolean` μεταβλητή `classes` έχει τιμή `true`, αλλιώς δημιουργεί ένα αντικείμενο τύπου Data.
ii. Στην συνέχεια, ορίζει ένα `URI` προς το κατάλληλο XML Schema αρχείο για την επικύρωση,
iii. Κάνει serialize την κλάση και εκτυπώνει το αποτέλεσμα.
iv. Η Client εφαρμογή ορίζει μία `URL` με το την διεύθυνση του Servlet και ανοίγει μία `HttpURLConnection`.
v. Την παραμετροποιεί
vi. Και στην συνέχεια δημιουργεί ένα `DataOutputStream` για να στείλει στο Servlet το αποτέλεσμα της serialization.
vii. Ο Servlet ορίζει ένα `PrintWriter` για να επιστρέψει στο Client το αποτέλεσμα της επεξεργασίας.
viii. Λαμβάνει από το κατάλληλο `InputStream` τα δεδομένα που έστειλε ο Client και τα μετατρέπει σε μία συμβολοσειρά `s`.
ix. Ελέγχει την συμβολοσειρά για να προσδιορίσει το σχήμα που χρησιμοποιείται.
x. Στην συνέχεια, ορίζει ένα `URI` προς το κατάλληλο XML Schema αρχείο για την επικύρωση.
xi. Κάνει parse την συμβολοσειρά και παίρνει ένα αντικείμενο σαν αποτέλεσμα.
xii. Κάνει serialize αυτό το αντικείμενο και
xiii. Στέλνει πίσω την νέα συμβολοσειρά στο Client μέσω `PrintWriter`.
xiv. Η Client εφαρμογή λαμβάνει την απάντηση του Servlet μέσω `InputStream`, την μετατρέπει σε συμβολοσειρά και την εκτυπώνει.

xv. Ανάλογα με την τιμή της μεταβλητής `classes`, κάνει parse την συμβολοσειρά με την κατάλληλη μέθοδος και εκτυπώνει τις τιμές των χαρακτηριστικών της.

Παρακάτω παρουσιάζεται ο κώδικας της Client εφαρμογής (TestClient.java) και του Servlet (testServlet.java).

3.4.2 Υλοποίηση της Client εφαρμογής

Κλάση TestClient.java :

```
import java.net.*;
import java.io.*;
import java.math.BigInteger;
import java.util.*;
import mydata.*;
import data.*;
import util.*;

public class TestClient {

  public static void main(String[] args) {

      HttpURLConnection connection = null;
      boolean type = false;   //true=get; false=post
      boolean classes = false; //true-> send Person ; false-> send Data

      // GET REQUEST
      if(type){

            String params = null;
            params = "?param1=test1&param2=test2&param3=test3";
            try {
                  URL netURL = new
                     URL("http://localhost:8080/nef/testServlet"+params);
                  connection = (HttpURLConnection) netURL.openConnection();
                  connection.setRequestMethod( "GET" );
            } catch (MalformedURLException e) {
                  e.printStackTrace();
            } catch (ProtocolException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            }

      }

      //POST REQUEST
      else{
            String s = null;

            // (i)
            //test of person class
            if(classes){
                  Person p=new Person();
```

```
		p.setAge(new BigInteger("22"));
		p.setNom("Nef");
		p.setPrenom("Marie");
		// (ii)
		URI uri = null;
		try {
			uri = new URI("file:/c:/person.xsd");
		} catch (URISyntaxException e) {
			e.printStackTrace();
		}
		// (iii)
		s=Serializer.serializePerson(p, uri);
		System.out.println("String i'm sending (person): \n"+s);
	}
	//(i)
	//test of data class
	else{
		Data d = new Data();
		Report r1 = new Report();
		Report r2 = new Report();
		TagList t1 = new TagList();
		TagList t2 = new TagList();

		t1.getTag().add("report1-tag1");
		t1.getTag().add("report1-tag2");

		t2.getTag().add("report2-tag1");
		t2.getTag().add("report2-tag2");
		t2.getTag().add("report2-tag3");

		r1.setTagList(t1);
		r1.setAntenna("ant1");

		r2.setTagList(t2);
		r2.setAntenna("ant2");

		d.getReport().add(r1);
		d.getReport().add(r2);
		// (ii)
		URI uri = null;
		try {
			uri = new URI("file:/c:/data.xsd");
		} catch (URISyntaxException e) {
			e.printStackTrace();
		}
		//(iii)
		s=Serializer.serializeData(d, uri);
		System.out.println("String i'm sending (data): \n"+s);
	}

	// (iv)
	try {
		URL netURL = new
			URL("http://localhost:8080/nef/testServlet");
		connection = (HttpURLConnection) netURL.openConnection();
		connection.setRequestMethod( "POST" );
	} catch (MalformedURLException e) {
		e.printStackTrace();
	} catch (ProtocolException e) {
		e.printStackTrace();
	} catch (IOException e) {
```

```
                e.printStackTrace();
        }

        // (v)
        connection.setRequestProperty("Content-Length", ""
                +Integer.toString(s.getBytes().length));
        connection.setUseCaches(false);
        connection.setDoInput(true);
        connection.setDoOutput(true);

        // (vi)
        try {
                DataOutputStream out = new
                        DataOutputStream(connection.getOutputStream());
                out.writeBytes( s );
                out.flush();
                out.close();
        } catch (IOException e) {
                e.printStackTrace();
        }
}

/*********************************************************************/

// RESPONSE FROM SERVER

String response = null;
BufferedReader is = null;

// RESPONSE TO THE GET REQUEST
if(type){
        try {

                is = new BufferedReader(new
                        InputStreamReader(connection.getInputStream()));
                response = is.readLine();
        } catch (IOException e) {
                e.printStackTrace();
        }
        System.out.println("Response from server: ");
        while( response != null )
        {
            System.out.println( response );
            try {
                        response = is.readLine();
              } catch (IOException e) {
                        e.printStackTrace();
                }

        }
}

//RESPONSE TO THE POST REQUEST
else{
        // (xiv)
        try {
                is = new BufferedReader(new
                        InputStreamReader(connection.getInputStream()));
        } catch (IOException e) {
                e.printStackTrace();
        }
```

```
                StringBuffer sb = new StringBuffer(10000);
                String buf = null;
                try {
                      while((buf = is.readLine())!= null){
                          sb.append(buf);
                          sb.append('\n');
                      }
                } catch (IOException e) {
                      e.printStackTrace();
                }
                response = sb.toString();
                System.out.println("String i got back from server:
                      \n"+response);

                // (xv)
                if(classes){
                      System.out.println("\nPerson elements after parsing
                            string:");
                      Person pp = null;
                      URI uri = null;
                      try {
                            uri = new URI("file:/c:/person.xsd");
                      } catch (URISyntaxException e) {
                            e.printStackTrace();
                      }
                      pp=Parser.parsePerson(response, uri);
                      System.out.println("Nom: " + pp.getNom());
                      System.out.println("Prenom: " + pp.getPrenom());
                      System.out.println("Age: " + pp.getAge());
                }
                else{
                      System.out.println("\nData elements after parsing
                            string:");
                      Data dd = null;
                      URI uri = null;
                      try {
                            uri = new URI("file:/c:/data.xsd");
                      } catch (URISyntaxException e) {
                            e.printStackTrace();
                      }
                      dd = Parser.parseData(response, uri);
                      List<Report> rr = dd.getReport();
                      Iterator<Report> i = rr.iterator();
                      while(i.hasNext()){
                            Report rrr = i.next();
                            System.out.println("Report : ");
                            System.out.println("\tAntenna : " +
                                  rrr.getAntenna());
                            TagList tt=rrr.getTagList();
                            List<String> str = tt.getTag();
                            Iterator<String> j = str.iterator();
                            System.out.println("\tTagList : ");
                            while(j.hasNext()){
                                  System.out.println("\t\t- " + j.next());
                            }
                      }
                }
          }
          System.out.println("\ntest finished");
    }
}
```

3.4.3 Υλοποίηση του Servlet

Κλάση testServlet.java :

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.URI;
import java.net.URISyntaxException;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import mydata.*;
import data.*;
import util.*;


public class testServlet extends javax.servlet.http.HttpServlet implements
javax.servlet.Servlet {
	static final long serialVersionUID = 1L;

	public testServlet() {
		super();
	}


	protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		PrintWriter printWriter = response.getWriter();
		printWriter.println("Hello from server!!");

		Enumeration params = request.getParameterNames();
		String paramName = null;
		String[] paramValues = null;

		while (params.hasMoreElements()) {
		    paramName = (String) params.nextElement();
		    paramValues = request.getParameterValues(paramName);
		    printWriter.print("\nParameter name is " + paramName);
		    for (int i = 0; i < paramValues.length; i++) {
		        printWriter.println(", value " + i + " is " +
		            paramValues[i].toString());
		    }
		}
		printWriter.flush();
	}


	protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

		// (vii)
		PrintWriter printWriter = response.getWriter();
```

```
            // (viii)
            BufferedReader is = new BufferedReader(new
                  InputStreamReader(request.getInputStream()));
            StringBuffer sb = new StringBuffer(10000);
            String buf = null;
            while((buf = is.readLine())!= null){
                sb.append(buf);
            }
            String s = sb.toString();
            String ss = null;

            // (ix)
            if(s.contains("<person>")){
                  // (x)
                  URI uri = null;
                  try {
                        uri = new URI("file:/c:/person.xsd");
                  } catch (URISyntaxException e) {
                        e.printStackTrace();
                  }
                  // (xi)
                  Person pp = null;
                  pp = Parser.parsePerson(s, uri);
                  // (xii)
                  ss = Serializer.serializePerson(pp, uri);
            }
            // (ix)
            else{
                  // (x)
                  URI uri = null;
                  try {
                        uri = new URI("file:/c:/data.xsd");
                  } catch (URISyntaxException e) {
                        e.printStackTrace();
                  }
                  // (xi)
                  Data dd = null;
                  dd = Parser.parseData(s, uri);
                  // (xii)
                  ss = Serializer.serializeData(dd, uri);
            }
            // (xiii)
            printWriter.print(ss);
            printWriter.flush();
      }
}
```

4 Java API for XML Web Services (JAX-WS)

4.1 Εισαγωγή

JAX-WS είναι μία τεχνολογία για την ανάπτυξη web services και clients που επικοινωνούν χρησιμοποιώντας XML. Χρησιμοποιεί επίσης annotations που αποτελεί τρόπος εισαγωγής μεταδεδομένων (metadata) στο πηγαίο κώδικα.

Στο JAX-WS, η κλήση Web Service λειτουργίας αναπαριστάται με XML-based πρωτόκολλα όπως η SOAP (Simple Object Access Protocol). Η SOAP specification ορίζει την envelope structure, τους κανόνες κωδικοποίησης, καθώς επίσης και συμβάσεις για την αναπαράσταση web service κλήσεων και απαντήσεων. Αυτές οι κλήσεις και απαντήσεις μεταδίδονται στην μορφή SOAP μηνυμάτων (XML αρχεία) μέσω HTTP.

Αν και τα SOAP μηνύματα είναι σχετικά σύνθετα, το JAX-WS κρύβει αυτήν την πολυπλοκότητα στον προγραμματιστή. Επίσης, ένα άλλο πλεονέκτημα του JAX-WS είναι η ανεξαρτησία πλατφόρμας της γλώσσας προγραμματισμού Java. Το JAX-WS δεν είναι περιοριστικός : ένα JAX-WS Client μπορεί να έχει πρόσβαση σε ένα Web Service που δεν τρέχει σε μία Java πλατφόρμας και αντίστροφα. Αυτή η ευελιξία οφείλεται στην χρήση τεχνολογιών ορισμένες από το World Wide Web Consortium (W3C) : HTTP, SOAP, και την Web Service Description Language (WSDL). Η WSDL ορίζει ένα XML format για την περιγραφή μιας υπηρεσίας.

4.2 Παράδειγμα υλοποίησης

4.2.1 Εισαγωγή

Στο παράδειγμα αυτό, θα υλοποιήσουμε ένα απλό Web Service που προσθέτει δύο ακέραιους και μία Client εφαρμογή που θα καλεί την αυτήν την πρόσθεση δέκα φορές, προσθέτοντας το 10 με τους αριθμούς από το 0 μέχρι το 9.

4.2.2 Ανάπτυξη του Web Service

Κλάση Calculator.java :

```
package endpoint;

import javax.jws.WebService;
import javax.jws.WebMethod;

@WebService(
    name="Calculator",
    serviceName="CalculatorService",
    targetNamespace="http://techtip.com/jaxws/sample"
)
public class Calculator {
        public Calculator() {}

        @WebMethod(operationName="add", action="urn:Add")
        public int add(int i, int j) {
                int k = i +j ;
                System.out.println(i + "+" + j +" = " + k);

                return k;
        }
}
```

Παρατηρούμε ότι χρησιμοποιούνται δύο annotations : @WebService και @WebMethod.

Μία έγκυρη κλάση υλοποίησης ενός endpoint πρέπει να περιέχει ένα @WebService annotation. Η τιμή της ιδιότητας name στο @WebService annotation προσδιορίζει ένα WSDL portType, στην συγκεκριμένη περίπτωση : "Calculator". Το serviceName ("CalculatorService") είναι μία WSDL υπηρεσία. Το targetNamespace προσδιορίζει το XML namespace που χρησιμοποιείται για το WSDL. Όλες αυτές οι ιδιότητες είναι προαιρετικές.

Το @WebMethod δηλώνει ότι μία μέθοδος είναι τύπου Web Service method. Η τιμής της operationName ιδιότητας προσδιορίζει μία WSDL λειτουργία, στην συγκεκριμένη περίπτωση : add. Το action ("urn:Add") προσδιορίζει ένα XML namespace για το WSDL. Οι δύο ιδιότητες είναι προαιρετικές, αν δεν υπάρχουν χρησιμοποιείται το όνομα της μεθόδους για το operationName και την τιμή του targetNamespace για το action.

4.2.3 Ανάπτυξη της Client εφαρμογής

Κλάση JAXWSClient.java :

```
package client;
import javax.xml.ws.WebServiceRef;
import com.techtip.jaxws.sample.CalculatorService;
import com.techtip.jaxws.sample.Calculator;

public class JAXWSClient {
   @WebServiceRef(wsdlLocation=
      "http://localhost:8080/jaxws-webservice/CalculatorService?WSDL")

   static CalculatorService service;

   public static void main(String[] args) {
      try {
        JAXWSClient client = new JAXWSClient();
        client.doTest(args);
      } catch(Exception e) {
        e.printStackTrace();
      }
   }

   public void doTest(String[] args) {
     try {
       System.out.println("Retrieving port from the service " + service);
       Calculator port = service.getCalculatorPort();
       System.out.println("Invoking add operation on the calculator port");
       for (int i=0;i>10;i++) {
          int ret = port.add(i, 10);
          if(ret != (i + 10)) {
            System.out.println("Unexpected greeting " + ret);
            return;
          }
          System.out.println(" Adding : " + i + " + 10 = "  + ret);
       }
     } catch(Exception e) {
       e.printStackTrace();
     }
   }
}
```

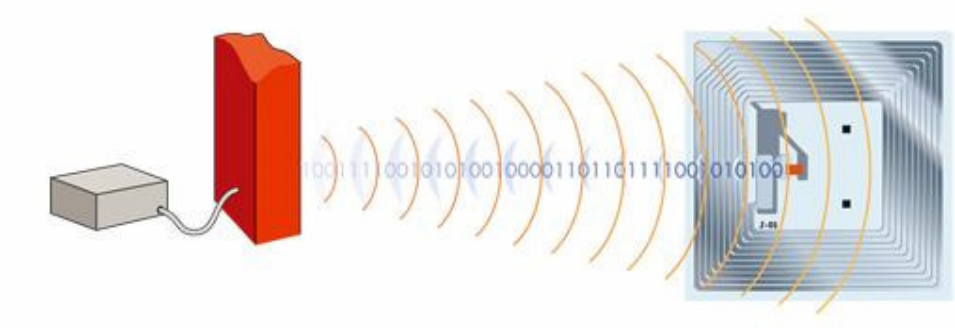
Το @WebServiceRef annotation χρησιμοποιείται για την δήλωση αναφοράς σε ένα Web Service. Η τιμή της ιδιότητας wsdlLocation είναι το URL του WSDL αρχείου της συγκεκριμένης υπηρεσίας. Το Client ανακτά το endpoint Calculator από το CalculatorService με την μέθοδο getWebServiceRefNamePort, όπου WebServiceRefName είναι το όνομα της ιδιότητας του @WebServiceRef, ή την τιμή του WSDP port στο παραγόμενο WSDL αρχείο. Όταν το Client ανακτά το endpoint, καλεί την add λειτουργία δέκα φορές.

5 RFID και EPCglobal

5.1 Εισαγωγή

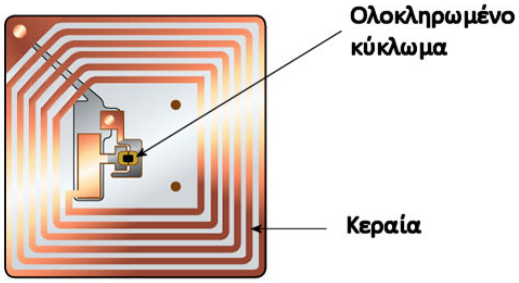
5.1.1 RFID

Η τεχνολογία RFID (Radio Frequency Identification) επιτρέπει την αναγνώριση αντικειμένων (ή ζωντανών πλασμάτων) και των χαρακτηριστικών τους, καθώς επίσης και την παρακολούθηση της πορείας τους από απόσταση χρησιμοποιώντας ένα tag. Αυτό το tag εκπέμπει ραδιοκύματα και βρίσκεται πάνω ή ενσωματωμένο στο αντικείμενο. Η τεχνολογία RFID επιτρέπει την αναγνώριση των tags που βρίσκονται μερικά μέτρα μακριά χωρίς απαραίτητη άμεση οπτική επαφή με τον reader (σε αντίθεση με το barcode) και μπορεί να διαπερνά λεπτά στρώματα υλικών όπως η μπογιά, το χιόνι κλπ. Η διαδικασία αυτή απεικονίζεται στην εικόνα 5.1.



Εικόνα 5.1 : Επικοινωνία reader/tag

Οι περισσότεροι RFID tags περιέχει τουλάχιστον δύο μέρη όπως φαίνεται στην εικόνα 5.2. Το πρώτο είναι ένα ολοκληρωμένο κύκλωμα για την αποθήκευση και την επεξεργασία της πληροφορίας, για την διαμόρφωση και αποδιαμόρφωση του σήματος, και άλλες ειδικές λειτουργίες. Το δεύτερο μέρος αποτελείται από μία κεραία για την λήψη και την μετάδοση του σήματος.



Εικόνα 5.2 : RFID tag

Μερικές από τις χρήσης της τεχνολογίας αυτή είναι :

- Έξυπνα labels και labels ασφαλείας
- Διαχείριση προϊόντων και απογραφή
- Τοποθέτηση σε κλειδιά αυτοκινήτου για καλύτερη ασφάλεια
- Έλεγχος κλοπής
- Παρακολούθηση αντικειμένων, ζώων, ανθρώπων
- Τοποθέτηση πάνω σε φάρμακα για την αποφυγή πλαστών φαρμάκων στην νόμιμη αλυσίδα παραγωγής
- Ευκολότερη πρόσβαση σε πανεπιστημιακά κτίρια από τους φοιτητές
- Ευκολότερη διαχείριση βιβλιοθήκης
- Καλύτερη αποδοτικότητα στην πρόσβαση στο χώρο διεξαγωγής ψυχαγωγικών ή αθλητικών γεγονότων
- Καλύτερη αποδοτικότητα στα διόδια
- ...

5.1.2 EPCglobal Standards

Τα EPCglobal Standards αποτελούνται από ένα σύστημα κωδικοποίησης των προϊόντων, το Electronic Product Code (EPC), από ένα standard του RFID tag και από ένα δίκτυο διαμοίρασης πληροφοριών, το EPC Network.

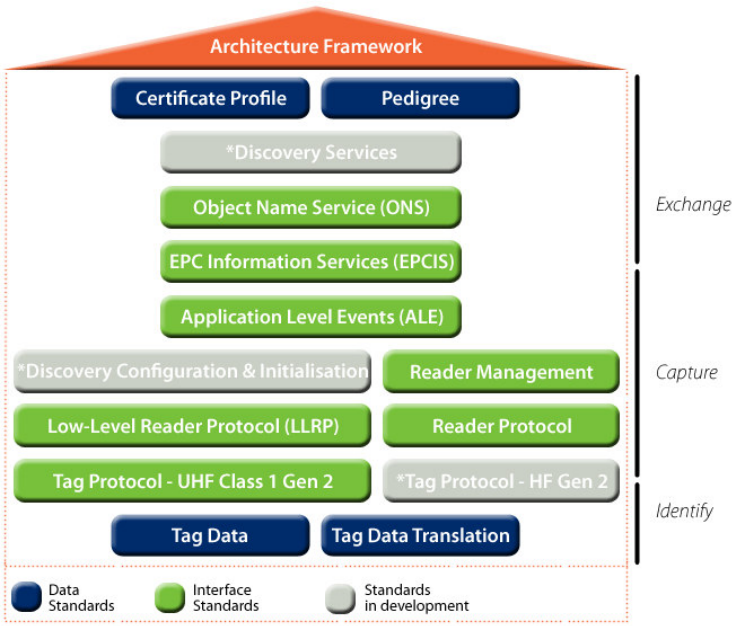
Σε γενικές γραμμές επιτρέπουν :

- Την αναγνώριση των ατομικών προϊόντων, κιβώτιων, αγαθών, κλπ., έτσι ώστε να παρακολουθούνται ατομικά
- Την συλλογή δεδομένων για την πορεία των φυσικών αγαθών, δημιουργώντας έτσι μία ορατότητα
- Την ανταλλαγή δεδομένων με IT εφαρμογές και εμπορικούς συνέταιρους, μετατρέποντας έτσι την ορατότητα σε πληροφορίες και δράσεις.

5.2 EPCGlobal Framework Architecture

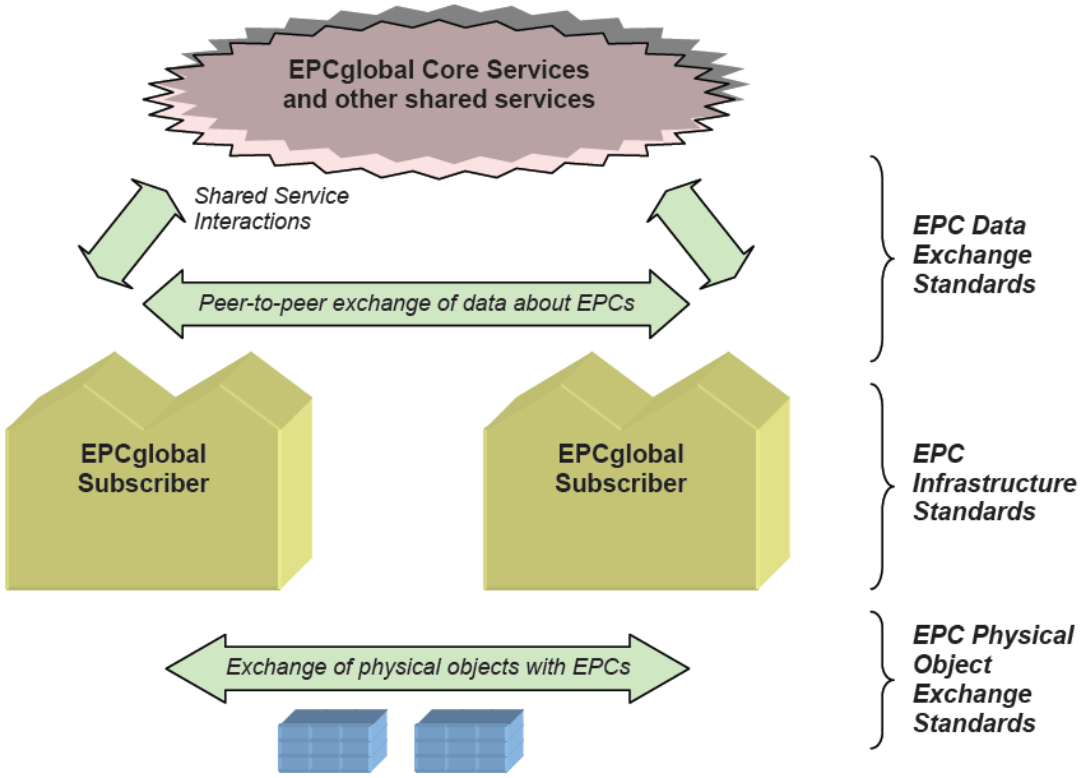
5.2.1 Εισαγωγή

Η εικόνα 5.3 παρουσιάζει μία σύνοψη των EPCglobal Standards.



Εικόνα 5.3 : EPCglobal Standards

Η εικόνα 5.4 απεικονίζει τις δραστηριότητες που εκτελούν οι EPCglobal Subscribers και το ρόλο των στοιχείων του EPCglobal Architecture Framework στην διευκόλυνση αυτών των δραστηριοτήτων.



Εικόνα 5.4 : Κυριότερες δραστηριότητες των EPCglobal Subscribers και των στοιχείων του EPCglobal Architecture Framework

Στην παραπάνω εικόνα, παρουσιάζονται τρεις κυριότερες δραστηριότητες, η καθεμία από αυτές στηρίζεται σε μία ομάδα από standards :

- EPC Physical Object Exchange : Οι subscribers ανταλλάζουν φυσικά αντικείμενα που αναγνωρίζονται με EPCs. Για πολλούς τελικούς χρήστες του EPCglobal Network, τα φυσικά αντικείμενα είναι εμπορεύματα, οι subscribers είναι συμμέτοχοι στην αλυσίδα παραγωγής αυτών των εμπορευμάτων, και το physical object exchange αποτελείται από διαδικασίες όπως η διεκπεραίωση, η παραλαβή κ.ο.κ. Υπάρχουν βέβαια πολλές άλλες εφαρμογές που διαφέρουν από αυτό το μοντέλο εμπορευμάτων όμως κάθε φορά υπάρχει ανάγκη να γίνει tagging των αντικειμένων. Το EPCglobal Architecture Framework ορίζει EPC physical object exchange standards, σχεδιασμένα έτσι ώστε όταν ένας subscriber παραδίδει κάποιο φυσικό αντικείμενο σε άλλο subscriber, ο τελευταίος να μπορέσει να προσδιορίσει το EPC του φυσικού αντικειμένου και να το ερμηνεύσει κατάλληλα.
- EPC Data Exchange : Οι subscribers ωφελούνται από το EPCglobal Network ανταλλάζοντας πληροφορίες μεταξύ τους και έτσι αυξάνοντας την δυνατότητα παρακολούθησης των αντικειμένων. Το EPCglobal Architecture Framework ορίζει EPC data exchange standards που προσφέρουν στους subscribers έναν τρόπο να ανταλλάζουν πληροφορίες για τα EPCs μεταξύ ορισμένων ομάδων χρηστών ή με το κοινό. Επίσης, προσφέρουν πρόσβαση στο EPCglobal Core

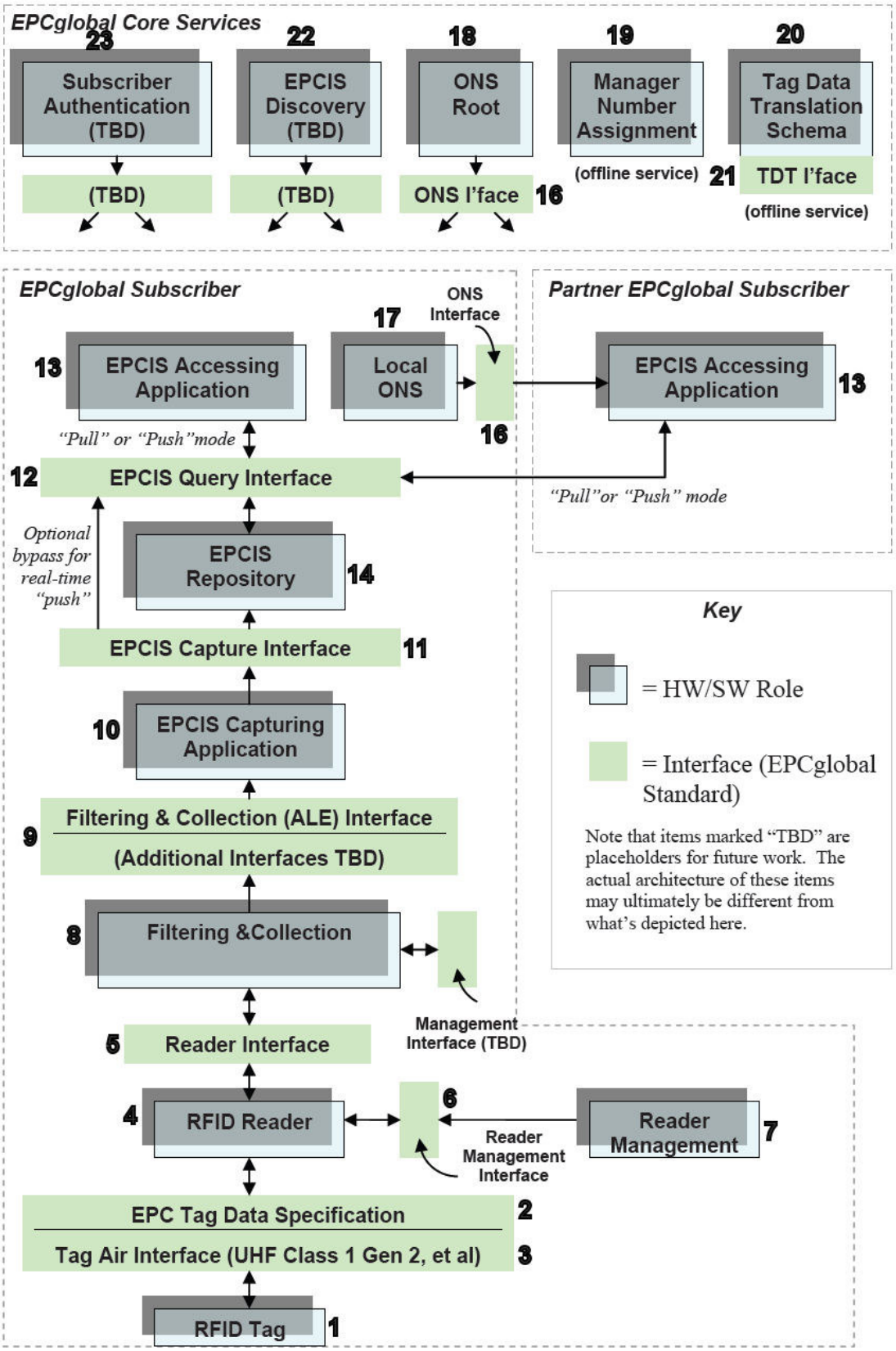
Services και σε άλλες διαμοιρασμένες υπηρεσίες που διευκολύνουν αυτές τις ανταλλαγές.

- EPC Infrastructure : Για να υπάρχουν πληροφορίες προς ανταλλαγή, κάθε subscriber πρέπει, μέσα στην εταιρία του, να δημιουργεί νέα EPCs για νέα αντικείμενα, να παρακολουθεί την πορεία των αντικειμένων με την ανίχνευση του EPC κώδικά τους, και να συλλέγει όλες αυτές τις πληροφορίες σε ένα εσωτερικό σύστημα εγγραφών. Το EPCglobal Architecture Framework ορίζει interface standards για τα κυριότερα στοιχεία της υποδομής που χρειάζεται για να γίνει συλλογή και αποθήκευση των EPC δεδομένων, επιτρέποντας έτσι στους subscribers να χτίσουν το δικό τους εσωτερικό σύστημα.

Αυτός ο διαχωρισμός βοηθά στην κατανόηση της γενικής λειτουργίας του EPCglobal Architecture Framework αλλά δεν πρέπει να θεωρηθεί ως απολύτως αυστηρή.

5.2.2 Περιγραφή ρόλων και διεπαφών

Στην εικόνα 5.5 παρουσιάζονται οι σχέσεις μεταξύ των EPCglobal Standards βασισμένες στην ροή των δεδομένων. Ακολουθεί μία λεπτομερής περιγραφή των ρόλων και των διεπαφών με την σχετική αρίθμηση.



Εικόνα 5.5 : Σχέσεις μεταξύ των EPCglobal Standards βασισμένες στην ροή δεδομένων

1. *RFID Tag (Ρόλος)*

   Το EPCglobal όρισε ένα σύστημα ταξινόμησης των tags για την περιγραφή των λειτουργιών τους.

   Class-1 : Identity Tags : Passive-backscatter Tags

   Ευθύνες :

   - Περιέχει ένα EPC αναγνωριστή. Μπορεί να επιτρέπει την αλλαγή του κώδικα αυτό μετά από την κατασκευή.
   - Μπορεί να περιέχει έναν αμετάβλητο κώδικα που περιέχει πληροφορίες κατασκευής όπως την ταυτότητα του κατασκευαστή, έναν μοναδικό σειριακό αριθμό κατασκευής, κλπ.
   - Μπορεί να παρέχει την δυνατότητα μόνιμης εξουδετέρωσης του Tag. Η λειτουργία αυτή μπορεί να υποθέτει πρόσθετη πληροφορία αποθηκευμένη στο Tag όπως κάποιο kill κωδικό.
   - Μπορεί να περιέχει πρόσθετες δυνατότητες όπως το κλείδωμα, τον έλεγχο πρόσβασης, κλπ. Οι λειτουργίες αυτές μπορεί να υποθέτουν πρόσθετη πληροφορία αποθηκευμένη στο Tag όπως κάποιο κωδικό κλειδώματος, την κατάσταση κλειδώματος, ένα συνθηματικό πρόσβασης, κλπ.
   - Μπορεί να περιέχουν πρόσθετες πληροφορίες του χρήστη εκτός από το EPC αναγνωριστή.

   Class-2 : Higher-Functionality Tags : Passive Tags

   Ευθύνες :

   - Ευθύνες των Class-1 Tags.
   - Εκτεταμένη μνήμη χρήστη.
   - Έλεγχος πρόσβασης με αυθεντικοποίηση.

   Class-3 : Semi-Passive Tags

   Ευθύνες :

   - Ευθύνες των Class-2 Tags.
   - Μία ολοκληρωμένη πηγή ενέργειας.
   - Integrated sensing circuitry.

   Class-4 : Active Tags

   Ευθύνες :

   - Ευθύνες των Class-3 Tags.
   - Επικοινωνία Tag-to-Tag.
   - Ενεργή επικοινωνία.
   - Δυνατότητες Ad-hoc networking.

2. *EPC Tag Data Specification (Διεπαφή)*

   Ευθύνες :

   - Ορίζει την ολική δομή του EPC, συμπεριλαμβανομένου και το μηχανισμό σύνδεσης διαφορετικών σχημάτων κωδικοποίησης.
   - Ορίζει ειδικά σχήματα κωδικοποίησης.
   - Για κάθε EPCglobal σχήμα κωδικοποίησης, ορίζει την δυαδική αναπαράσταση του για την χρήση σε RFID tags, την αναπαράσταση σε μορφή κειμένου για την χρήση στα συστήματα πληροφορίας (ειδικά στο ALE επίπεδο και παραπάνω στο EPCglobal Architecture Framework) και ορίζει κανόνες για την μετατροπή από μία αναπαράσταση στην άλλη.

3. *Tag Air Interface (Διεπαφή)*
   Ευθύνες :
   - Μεταδίδει μία εντολή σε ένα tag από ένα RFID Reader.
   - Μεταδίδει μία απάντηση ενός tag προς το RFID Reader που έκπεμψε την εντολή.
   - Παρέχει δυνατότητα σε ένα reader να ξεχωρίσει ατομικά tags όταν περισσότερο από ένα βρίσκονται στην εμβέλειά του.
   - Παρέχει δυνατότητα στους readers και στα tags να ελαχιστοποιήσουν την παρεμβολή μεταξύ τους.

4. *RFID Reader (Ρόλος)*
   Ευθύνες :
   - Διαβάζει τα EPCs από RFID Tags που βρίσκονται στην εμβέλεια μίας ή περισσότερων κεραιών (via Tag Air Interface) και διαδίδει τους EPCs σε μία host εφαρμογή (via Reader Interface).
   - Όταν το RFID Tag επιτρέπει να ξαναγραφτεί ο EPC κώδικας μετά την κατασκευή, γράφει το EPC στο tag (via Tag Air Interface) αφού έχει παραχθεί εντολή από μία host εφαρμογή (via Reader Interface).
   - Όταν το RFID Tag προσφέρει πρόσθετη πληροφορία χρήστη εκτός από το EPC κώδικα, διαβάζει και γράφει αυτήν την πληροφορία (via Tag Air Interface) αφού έχει παραχθεί εντολή από μία host εφαρμογή (via Reader Interface).
   - Όταν το RFID Tag προσφέρει πρόσθετες λειτουργίες όπως kill, lock, κλπ, χειρίζεται αυτές τις λειτουργίες (via Tag Air Interface) αφού έχει παραχθεί εντολή από μία host εφαρμογή (via Reader Interface).
   - Μπορεί να προσφέρει πρόσθετες λειτουργίες όπως το φιλτράρισμα των EPCs, την συσσώρευση της πληροφορίας που έχει διαβαστεί κ.ο.κ.

5. *Reader Interface (Διεπαφή)*
   Ευθύνες :
   - Προσφέρει δυνατότητα να σταλούν εντολές στον Reader για να διαβάζει τους EPC κώδικες, να διαβάζει την άλλη πληροφορία που βρίσκεται στα tags, να γράφει tags (EPC κώδικες ή άλλη πληροφορία) και να έχει πρόσβαση στις άλλες λειτουργίες όπως kill, lock κλπ.
   - Μπορεί να προσφέρει δυνατότητα πρόσβασης στις λειτουργίες διαχείρισης του RFID Reader όπως την ανακάλυψη, την ρύθμιση ή την ενημέρωση του firmware/software, την παρακολούθηση της κατάστασης, την συγκέντρωση στατιστικών στοιχείων, την συνδετικότητα της κεραίας, το επίπεδο της ισχύος της μετάδοσης και την διαχείριση κατανάλωσης ενέργειας του reader.
   - Μπορεί να προσφέρει δυνατότητα ελέγχου της RF όψης του RFID Reader όπως τον έλεγχο της χρήσης του RF φάσμα, την ανίχνευση και την μέτρηση των παρεμβολών, την μορφή της διαμόρφωσης, κλπ.
   - Μπορεί να προσφέρει δυνατότητα ελέγχου μερικών λειτουργιών της Tag Air Interface όπως τους παραμέτρους πρωτοκόλλου και τους singulation παραμέτρους.
   - Μπορεί να προσφέρει πρόσβαση σε επεξεργαστικές λειτουργίες όπως το φιλτράρισμα των EPCs, την συσσώρευση της πληροφορίας που έχει διαβαστεί κ.ο.κ.

6. *Reader Management Interface (Διεπαφή)*
   Ευθύνες :
   - Παρέχει δυνατότητα αίτησης της ρύθμισης ενός RFID Reader όπως την ταυτότητά του, τον αριθμό κεραιών κ.ο.κ.
   - Παρέχει δυνατότητα διαχείρισης της λειτουργικής κατάστασης ενός RFID Reader όπως τον αριθμό διαβασμένων tags, την κατάσταση των καναλιών επικοινωνίας, την παρακολούθηση της κατάστασης, την συνδετικότητα της κεραίας, το επίπεδο της ισχύος της μετάδοσης, κ.ο.κ.
   - Παρέχει σε ένα RFID Reader την δυνατότητα ενημέρωσης σταθμών διαχείρισης για πιθανά λειτουργικά προβλήματα.
   - Παρέχει δυνατότητα ελέγχου της ρύθμισης ενός RFID Reader όπως την ενεργοποίηση / απενεργοποίηση συγκεκριμένων κεραιών ή λειτουργιών κ.ο.κ
   - Μπορεί να προσφέρει δυνατότητα πρόσβασης στις λειτουργίες διαχείρισης του RFID Reader όπως την ανακάλυψη, την ρύθμιση ή την ενημέρωση του firmware/software, την παρακολούθηση της κατάστασης, την συγκέντρωση στατιστικών στοιχείων, την συνδετικότητα της κεραίας, το επίπεδο της ισχύος της μετάδοσης και την διαχείριση κατανάλωσης ενέργειας του reader.

7. *Reader Management (Ρόλος)*
   Ευθύνες :
   - Διαχειρίζεται την λειτουργική κατάσταση ενός ή περισσότερων RFID Readers σε μία υποδομή.
   - Προσφέρει στους RFID Readers μηχανισμούς προειδοποίησης των σταθμών διαχείρισης για πιθανά προβλήματα.
   - Διαχειρίζεται την ρύθμιση ενός ή περισσότερων RFID Readers.
   - Παρέχει πρόσβαση στις λειτουργίες διαχείρισης του RFID Reader όπως την ανακάλυψη, την ρύθμιση ή την ενημέρωση του firmware/software και την διαχείριση κατανάλωσης ενέργειας του reader.

8. *Filtering & Collection (Ρόλος)*
   Ευθύνες :
   - Λαμβάνει ανεπεξέργαστα tag reads από ένα ή περισσότερα RFID Readers.
   - Μειώνει τον όγκο των EPC δεδομένων μετατρέποντας ανεπεξέργαστα tags reads σε ροή γεγονότων καταλληλότερα για εφαρμογές.
   - Εκτελεί λειτουργίες γραψίματος ή άλλες λειτουργίες σε ένα ή περισσότερα RFID Readers.
   - Αποφασίζει ποιες από τις παραπάνω λειτουργίες μπορούν να εκτελεστούν από τον ίδιο τον RFID Reader και ποιες πρέπει να εκτελεστούν από τον ίδιο το Filtering & Collection ρόλο.
   - Αποκωδικοποιεί ανεπεξέργαστες tag τιμές σε URI αναπαράσταση και αντίστροφα.
   - Κάνει mapping τα "λογικά ονόματα των readers" και τις φυσικές συσκευές.
   - Μπορεί να προσφέρει αποκωδικοποίηση και κωδικοποίηση των μη EPC tag δεδομένων στην μνήμη του Tag διαθέσιμη για τις πληροφορίες του χρήστη.

- Όταν ο Filtering & Collection ρόλος προσπελαύνεται από περισσότερες από μία εφαρμογές, μεσολαβεί κατάλληλα για τις αιτήσεις που αφορούν την ίδια ομάδα RFID Readers.
- Καθορίζει και ελέγχει την στρατηγικής ανακάλυψης των tags που χρησιμοποιείται από τους RFID Readers.
- Μπορεί να συντονίσει τις λειτουργίες πολλών RFID Readers έτσι ώστε η μία λειτουργία να μην εμποδίζει τις άλλες.

9. *Filtering & Collection (ALE) Interface (Διεπαφή)*
   Ευθύνες :
   - Προσφέρει σε μία ή περισσότερες client εφαρμογές την δυνατότητα αίτησης EPC δεδομένων από μία ή περισσότερες Tag πηγές.
   - Προσφέρει σε μία ή περισσότερες client εφαρμογές την δυνατότητα αίτησης εφαρμογής μίας ομάδας λειτουργιών πάνω σε Tags όπως το γράψιμο, το κλείδωμα και το killing.
   - Απομονώνει τις clients εφαρμογές από την γνώση των αριθμών των φυσικών συσκευών έτσι ώστε να φαίνεται να υπάρχει μία λογική Tag πηγή.
   - Προσφέρει στις client εφαρμογές ένα δηλωτικό τρόπο να προσδιορίσουν ποιες λειτουργίες πρέπει να εφαρμοστούν στα EPC δεδομένα.
   - Προσφέρει στις client εφαρμογές την δυνατότητα αίτησης δεδομένων ή λειτουργιών on demand (σύγχρονη απάντηση) ή σαν μία standing αίτηση (ασύγχρονη απάντηση).
   - Προσφέρει σε πολλαπλές client εφαρμογές τρόπο διαμοίρασης δεδομένων από τον / τους ίδιο / ίδιους Readers.
   - Προσφέρει τυποποιημένη αναπαράσταση για τις client αιτήσεις EPC δεδομένων και λειτουργιών, καθώς επίσης και μία τυποποιημένη αναπαράσταση για την μετάδοση φιλτραρισμένων συλλεγμένων EPC δεδομένων και των αποτελεσμάτων των ολοκληρωμένων λειτουργιών.

10. *EPCIS (EPC Information Service) Capturing Application (Ρόλος)*
    Ευθύνες :
    - Αναγνωρίζει την ύπαρξη EPC-related business γεγονότων και τα μεταδίδει ως EPCIS δεδομένα.
    - Μπορεί να συντονίσει πολλαπλές πηγές δεδομένων για την αναγνώριση ενός ατομικού EPCIS γεγονότος.
    - Μπορεί να ελέγχει την εξέλιξη των ενεργειών στο φυσικό περιβάλλον όπως το γράψιμο RFID Tags και τον έλεγχο άλλων συσκευών.

11. *EPCIS Capture Interface (Διεπαφή)*
    Ευθύνες :
    - Προσφέρει έναν τρόπο επικοινωνίας για την μετάδοση των EPCIS γεγονότων που παράχθηκαν από τις EPCIS Capturing Applications σε άλλου ρόλους που τα χρειάζονται όπως τα EPCIS Repositories, εσωτερικές EPCIS Accessing Applications και Partner EPCIS Accessing Applications.

12. EPCIS Query Interface (Διεπαφή)

Ευθύνες :

- Προσφέρει σε μία EPCIS Accessing Application την δυνατότητα αίτησης EPCIS δεδομένων από ένα EPCIS Repository ή μία EPCIS Capturing Application καθώς επίσης και τον τρόπο με τον οποίο επιστρέφεται η απάντηση.
- Προσφέρει την δυνατότητα αυθεντικοποίησης των δύο πλευρών.
- Επιστρέφει το αποτέλεσμα των αποφάσεων εξουσιοδότησης που έχει πάρει η προμηθευτική πλευρά.

13. EPCIS Accessing Application (Ρόλος)

Ευθύνες :

- Εξασφαλίζει όλες τις business λειτουργίες της επιχείρησης, όπως την διαχείριση αποθηκών, την διεκπεραίωση και την παραλαβή, την ανάλυση του ιστορικού παραγωγής, κ.ο.κ., με την βοήθεια των EPC-related δεδομένων.

14. EPCIS Repository (Ρόλος)

Ευθύνες :

- Αποθηκεύει τα EPCIS-level γεγονότα που έχουν παραχθεί από μία ή περισσότερες EPCIS Capturing Application, και τις κάνει διαθέσιμες για μελλοντικές αιτήσεις των EPCIS Accessing Applications.

15. Drug Pedigree Messaging (Διεπαφή)

Ευθύνες :

- Ορίζει μία επίσημη συλλογή XML σχημάτων και σχετικών οδηγίες χρήσης υπό μίας Drug Pedigree Specification που μπορεί να υιοθετηθεί από τα μέλη της φαρμακευτικής αλυσίδας παραγωγής για την αποφυγή εισαγωγής πλαστών φαρμάκων μέσα στην αλυσίδα αυτή.

16. Object Name Service (ONS) Interface (Διεπαφή)

Ευθύνες :

- Προσφέρει σε μία EPCIS υπηρεσία ή μία άλλη υπηρεσία συνδεδεμένη με κάποιο EPC την δυνατότητα αναζήτησης αναφοράς. Η λίστα των συνδεδεμένων υπηρεσιών με κάποιο EPC διατηρείται από το EPC Manager αυτού του EPC.

17. Local ONS (Ρόλος)

Ευθύνες :

- Εκπληρώνει τις ONS lookup αιτήσεις για EPCs που βρίσκονται υπό ελέγχου μίας εταιρίας που διαχειρίζεται το Local ONS, δηλαδή, EPCs των οποίων η εταιρία είναι το EPC Manager.

18. ONS Root (Core Service)

Ευθύνες :

- Προσφέρει αρχικό σημείο επικοινωνίας για ONS lookups.
- Σε πολλές περιπτώσεις, κατευθύνει τις lookup λειτουργίες προς τα κατάλληλα Local ONS.

- Μπορεί να εκπληρώνει τις lookup λειτουργίες όταν δεν υπάρχει κατάλληλο Local ONS.
- Προσφέρει lookup υπηρεσία για 64-bit Manager Index values όπως προβλέπεται από το EPC Tag Data Specification.

19. Manager Number Assignment (Core Service)

Ευθύνες :

- Εξασφαλίζει την γενική μοναδικότητα των EPCs με την διατήρηση της μοναδικότητας των EPC Manager Numbers εκχωρημένοι στους EPCglobal Subscribers.

20. Tag Data Translation Schema (Core Service)

Ευθύνες :

- Προσφέρει ένα machine-readable αρχείο που ορίζει πως πρέπει να γίνεται η μετάφραση μεταξύ EPC κωδικοποιήσεων.

21. Tag Data Translation Interface (Διεπαφή)

Ευθύνες :

- Κωδικοποιεί σε machine-readable μορφή τους κανόνες για το πως πρέπει να γίνεται η μετάφραση μεταξύ EPC κωδικοποιήσεων.

22. EPCIS Discovery (Core Service – TBD)

Ευθύνες :

- Προσφέρει τρόπος εντοπισμού όλων των EPCIS υπηρεσιών που πιθανόν να έχουν πληροφορίες για κάποιο συγκεκριμένο EPC.
- Μπορεί να προσφέρει και cache για τα επιλεγμένα EPCIS δεδομένα.
- Ενισχύει τις πολιτικές εξουσιοδοτήσεων με σεβασμό στην πρόσβαση στα παραπάνω δεδομένα.

23. Subscriber Authentication (Core Service – TBD)

Ευθύνες :

- Αυθεντικοποιεί την ταυτότητα ενός EPCglobal Subscriber.
- Εκδίδει πιστοποιητικά που μπορούν να χρησιμοποιηθούν από τους EPCglobal Subscribers για να αυθεντικοποιούνται μεταξύ τους.
- Αυθεντικοποιεί την συμμετοχή στις δικτυακές υπηρεσίες με την έγκριση μιας ενεργής EPCglobal Subscription.

24. Filtering & Collection Management Interface (Διεπαφή – TDB)

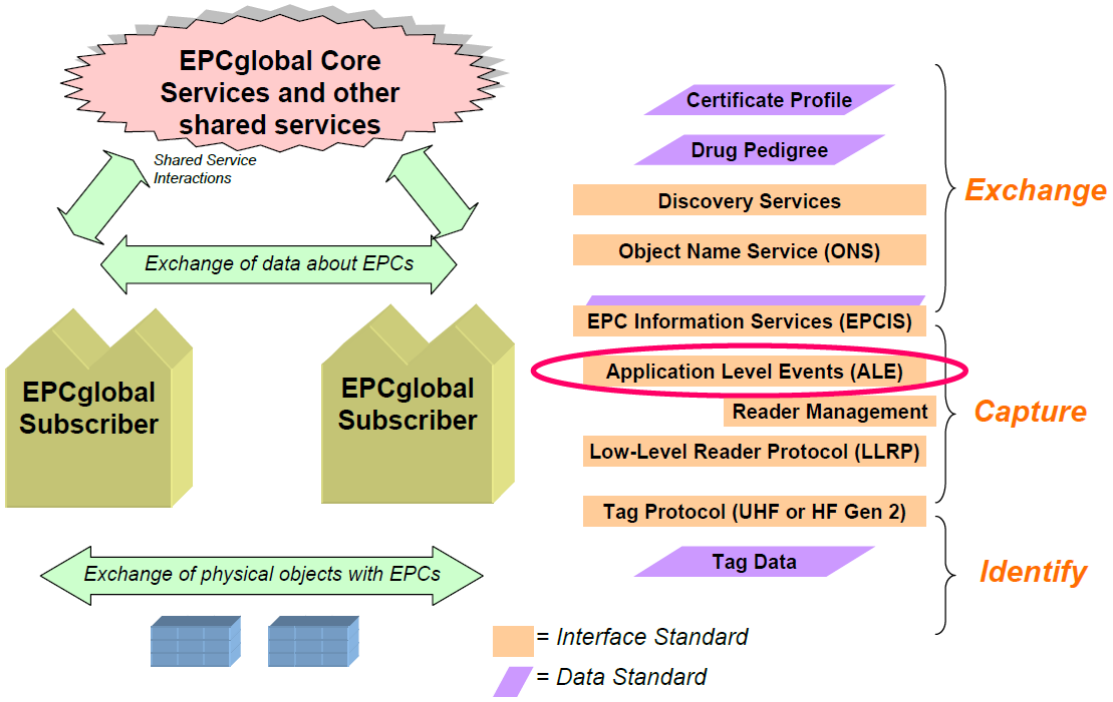
Ευθύνες :

- Προσφέρει δυνατότητα αναζήτησης ρυθμίσεων των συστημάτων που έχουν Filtering & Collection ευθύνες.
- Προσφέρει δυνατότητα διαχείρισης της λειτουργικής κατάστασης των συστημάτων που έχουν Filtering & Collection ευθύνες.
- Προσφέρει δυνατότητα διαχείρισης των ρυθμίσεων των συστημάτων που έχουν Filtering & Collection ευθύνες.

6 Application Level Events (ALE)

6.1 Εισαγωγή

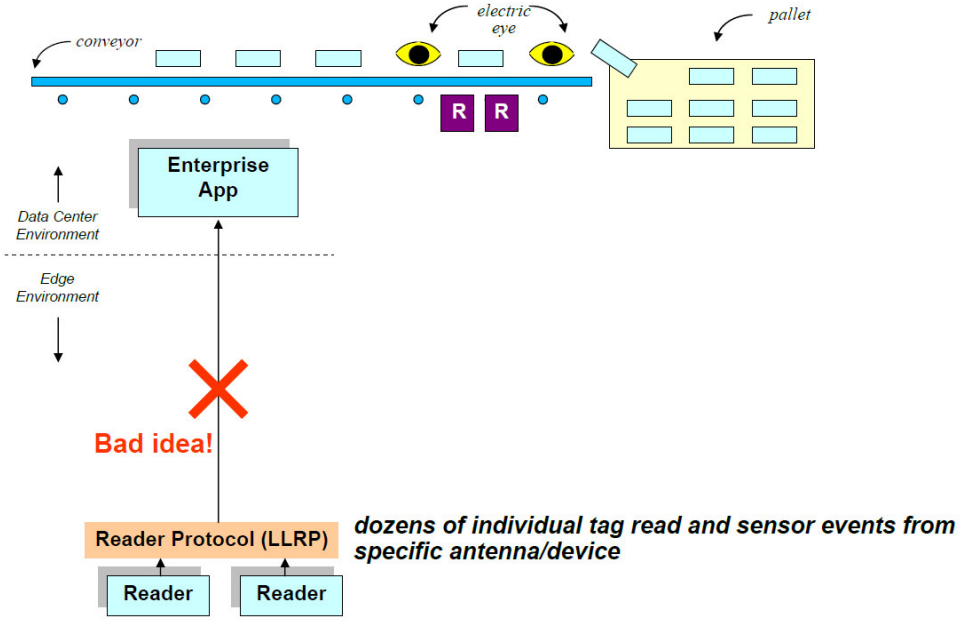
Η εικόνα 6.1 υπενθυμίζει την θέση της ALE στο σύνολο των EPCglobal Standards.



Εικόνα 6.1 : Η ALE στο σύνολο των EPCglobal Standards

Εισαγωγικό παράδειγμα συλλογής των δεδομένων : φόρτιση παλετών μεταφορών.

Στην εικόνα 6.2 παρουσιάζεται ένα σύστημα από τον οποίο λείπει η ALE και η EPCIS.

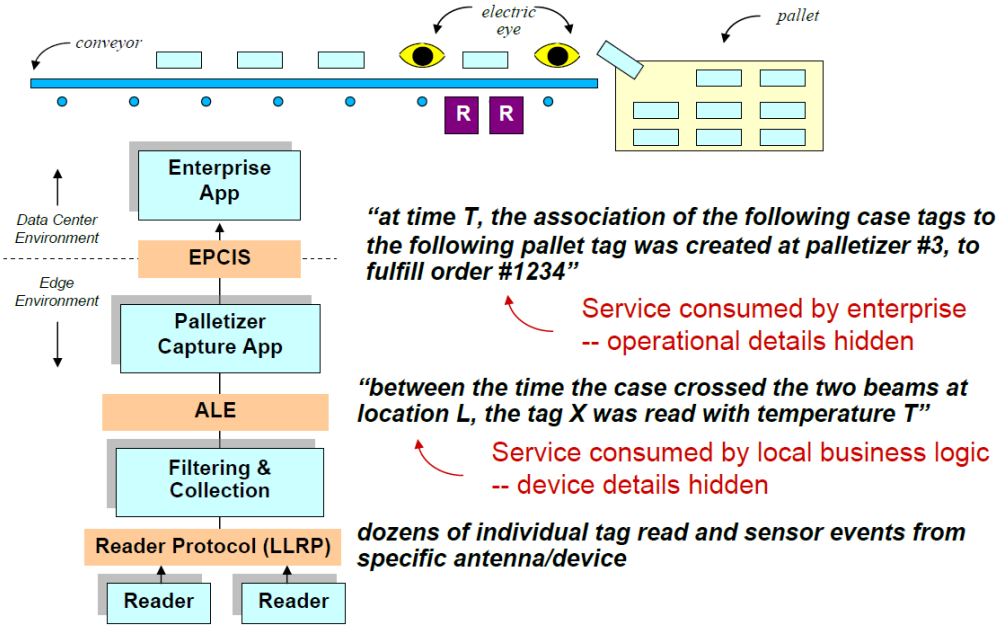


Εικόνα 6.2 : Προσέγγιση χωρίς την χρήση της ALE και της EPCIS

Παρατηρούμε πως θα ήταν πολύ δύσκολο να διαχειριστεί αυτό το σύστημα. Συγκεκριμένα, θα υπήρχαν προβλήματα όπως :

- Τεράστιος όγκος δεδομένων και δύσκολη διαχείριση από την εφαρμογή της επιχείρησης
- Ορατότητα των φυσικών δεδομένων για την εφαρμογή της επιχείρησης : είναι δύσκολο να προσδιοριστεί η σημασία των δεδομένων και να αλλαχτεί η λειτουργική διαδικασία

Στην εικόνα 6.3 παρουσιάζεται το ίδιο σύστημα με λειτουργίες ALE και EPCIS.



Εικόνα 6.3 : Προσέγγιση με την χρήση της ALE και της EPCIS

Παρατηρούμε πως πλέον η διαχείριση των δεδομένων γίνεται πολύ πιο εύκολα διότι σε κάθε επίπεδο, γίνεται επεξεργασία της παρακάτω πληροφορίας με το κατάλληλο επίπεδο αφαίρεσης των δεδομένων. Πλέον η εφαρμογή της εταιρίας λαμβάνει σύνθετη και κατανοητή φιλτραρισμένη πληροφορία.

6.2 Σκοπός της ALE

- Η μείωση του όγκου των δεδομένων από τους readers προς τις εφαρμογές
- Η ανύψωση του επιπέδου αφαίρεσης για τους προγραμματιστές εφαρμογών
- Η απομόνωση των εφαρμογών από τις λεπτομέρειες των συσκευών
- Η διαμοίραση δεδομένων μεταξύ πολλαπλών εφαρμογών
- Η επεκτασιμότητα στις αλλαγές
- Η εύκολη ενσωμάτωση χρησιμοποιώντας standard XML / Web Services τεχνολογία

6.3 Τρόποι υλοποίησης της ALE

Υπάρχουν τρεις βασικοί τρόποι υλοποίησης :

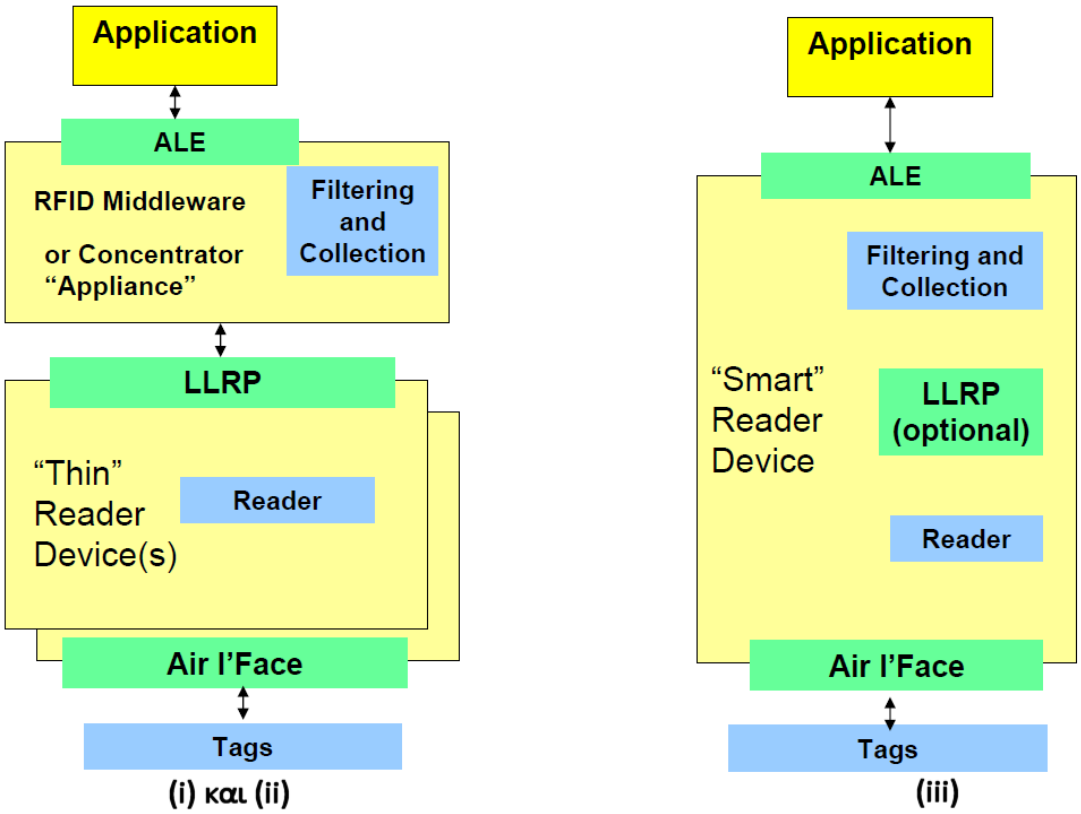
i. Software "middleware" εφαρμογές : είναι λογισμικό που διασυνδέεται τυπικά με πολλά διαφορετικά RFID readers, printers και σχετικές συσκευές μέσω LAN και

προσφέρει δυνατότητα ελέγχου και διαχείρισης αυτών των συσκευών. Middleware προϊόντα όπως αυτά παρέχουν την ALE σαν μία Application Programming Interface (API) για εφαρμογές που αναπτύσσονται πάνω σε αυτό το middleware.

ii. Hardware "controller" συσκευές : είναι hardware προϊόντα που παρέχουν περίπου τις ίδιες λειτουργίες με το sofware "middleware", όμως ενσωματώνονται σαν ένα "εργαλείο" στο δίκτυο του τελικού χρήστη αντί να γίνει κάποια εγκατάσταση λογισμικού σε έναν υπολογιστή γενικού σκοπού. Όπως και με τις "middleware" εφαρμογές, οι "controllers" παρέχουν την ALE σαν μία Application Programming Interface (API) για εφαρμογές που αναπτύσσονται πάνω σε αυτό το controller.

iii. "Smart" readers ή printers : είναι RFID readers και printers που παρέχουν μία πιο ολοκληρωμένη λύση ανάπτυξης εφαρμογής από τους "thin" readers και printers. Αφού διαθέτει την ALE κατευθείαν σαν μία διεπαφή, ένα "smart" reader ή printer επιτρέπει την κατασκευή εφαρμογών χωρίς την διαμεσολάβηση κάποιου "middleware" ή "controller".

Οι τρεις υλοποιήσεις αυτές παρουσιάζονται στην εικόνα 6.4.



Εικόνα 6.4 : Οι τρεις υλοποιήσεις της ALE

6.4 ALE APIs

Ο πίνακας 6.1 συνοψίζει τις βασικές λειτουργίες των ALE APIs.

APIs	**Κύριες χρήσεις**
Reading API *Read tags, report*	Χρησιμοποιούνται κυριότερα από εφαρμογές (επίπεδο δεδομένων)
Writing API *Initialize, read, write, lock, kill*	
Tag Memory API *Define symbolic names for memory fields, for use by Reading & Writing APIs*	Χρησιμοποιούνται κυριότερα για την ρύθμιση και την διαχείριση (επίπεδο ελέγχου)
Logical Reader API *Define symbolic names for reader/device resources, for use by Reading & Writing APIs*	
Access Control API *Control access by clients to other API features*	

Πίνακας 6.1 : Βασικές λειτουργίες και χρήσεις των ALE APIs

6.5 Αρχές της ALE

6.5.1 Εισαγωγή

Η ALE δηλώνει μία διεπαφή :

- ALE Client : η εφαρμογή που θέλει να βασίσει την λειτουργία της πάνω στα Tags.
- ALE Implementation : μέρος του συστήματος που υλοποιεί τα ALE APIs και επεξεργάζει τις αιτήσεις του client αλληλεπιδρώντας με τους readers ή με τις άλλες συσκευές.

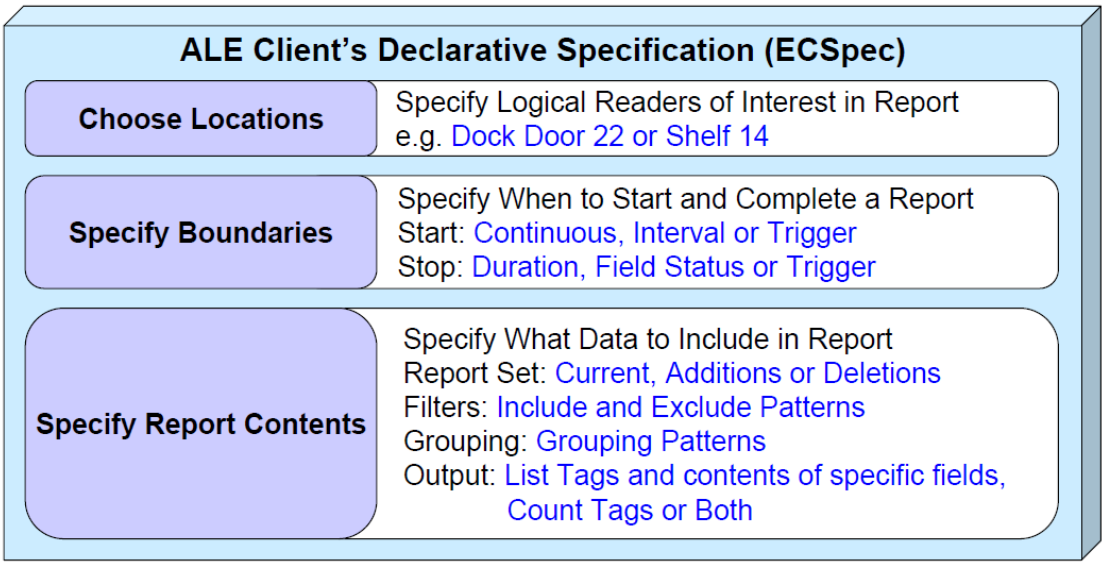
Η ALE είναι δηλωτική :

- Ο ALE Client δηλώνει αυτό που θέλει να γίνει.
- Η ALE Implementation αποφασίζει ποιος είναι ο καλύτερος τρόπος να εκτελεστεί αυτή η αίτηση (με την δυνατότητα συνδυασμού ταυτοχρόνων αιτήσεων).

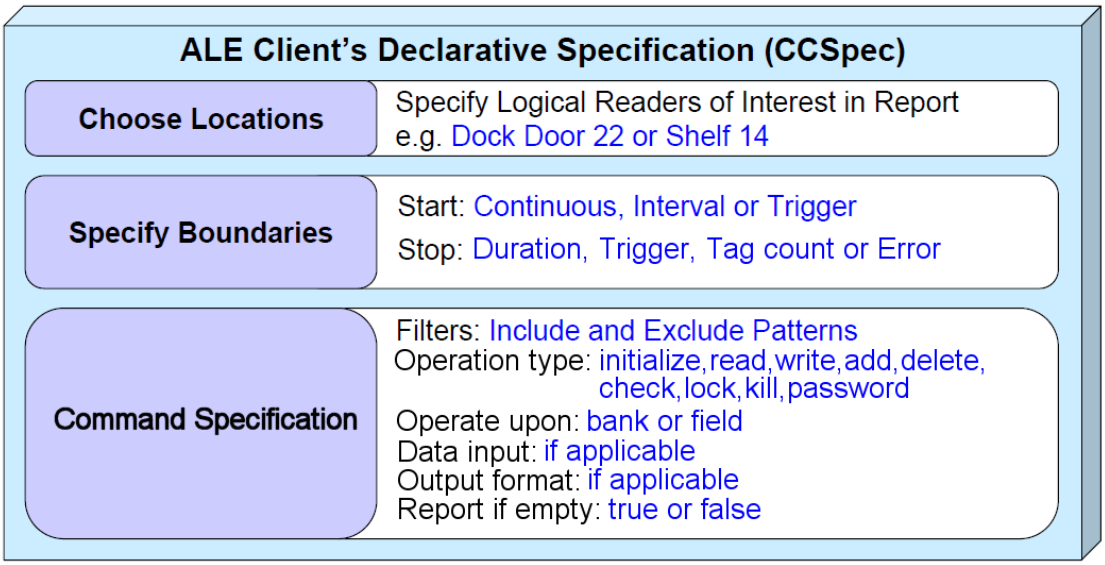
6.5.2 EC/CCSpecs και EC/CCReports

Η ALE διεπαφή επικεντρώνεται στα "specs" και "reports" :

- Event Cycle Spec (ECSpec) : η αίτηση του ALE Client στο Reading API (Εικόνα 6.5)
- Event Cycle Report (ECReport) : η απάντηση της ALE Implementation στο Reading API
- Command Cycle Spec (CCSpec) : η αίτηση του ALE Client στο Writing API (Εικόνα 6.6)
- Command Cycle Report (CCReport) : η απάντηση της ALE Implementation στο Writing API



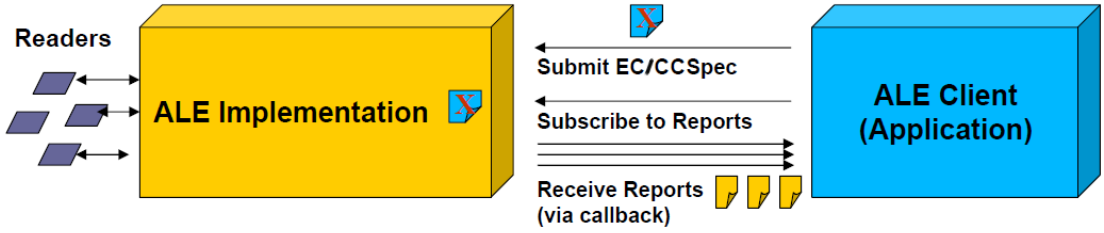
Εικόνα 6.5 : Γενική μορφή ενός ECSpec



Εικόνα 6.6 : Γενική μορφή ενός CCSpec

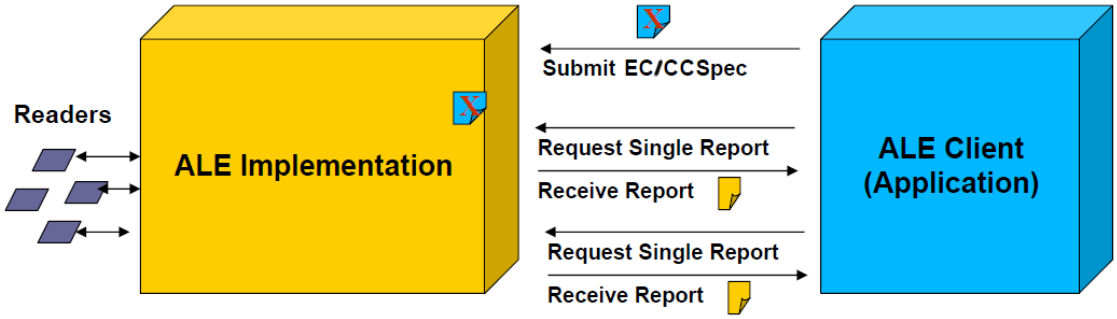
Υπάρχουν 3 βασικούς τρόπους να γίνουν αιτήσεις από τον ALEClient :

- Subscribe ("push") : Παράγονται ασύγχρονα reports για μία standing αίτηση όπως φαίνεται στην εικόνα 6.7. Χρησιμοποιείται συχνά για συνεχείς λειτουργίες ή όταν γίνεται triggering με χρόνο ή εξωτερικά γεγονότα.



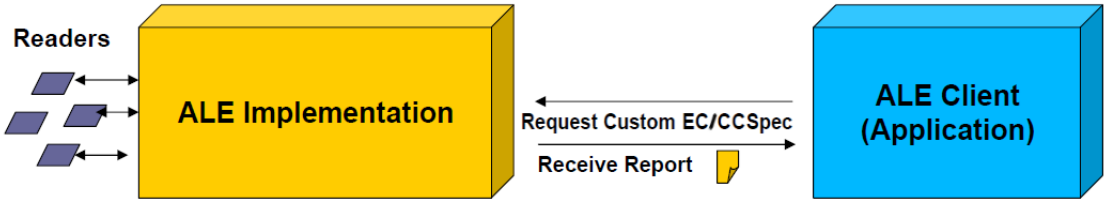
Εικόνα 6.7 : Subscribe μέθοδος

- Poll (on-demand, "pull") : Παράγονται σύγχρονα reports για μία standing αίτηση όπως φαίνεται στην εικόνα 6.8. Χρησιμοποιείται συχνά όταν γίνεται triggering προγραμματιστικά (για παράδειγμα, με χρήση GUI).



Εικόνα 6.8 : Poll μέθοδος

- Immediate : Παράγεται σύγχρονο report για μία μοναδική αίτηση όπως φαίνεται στην εικόνα 6.9.

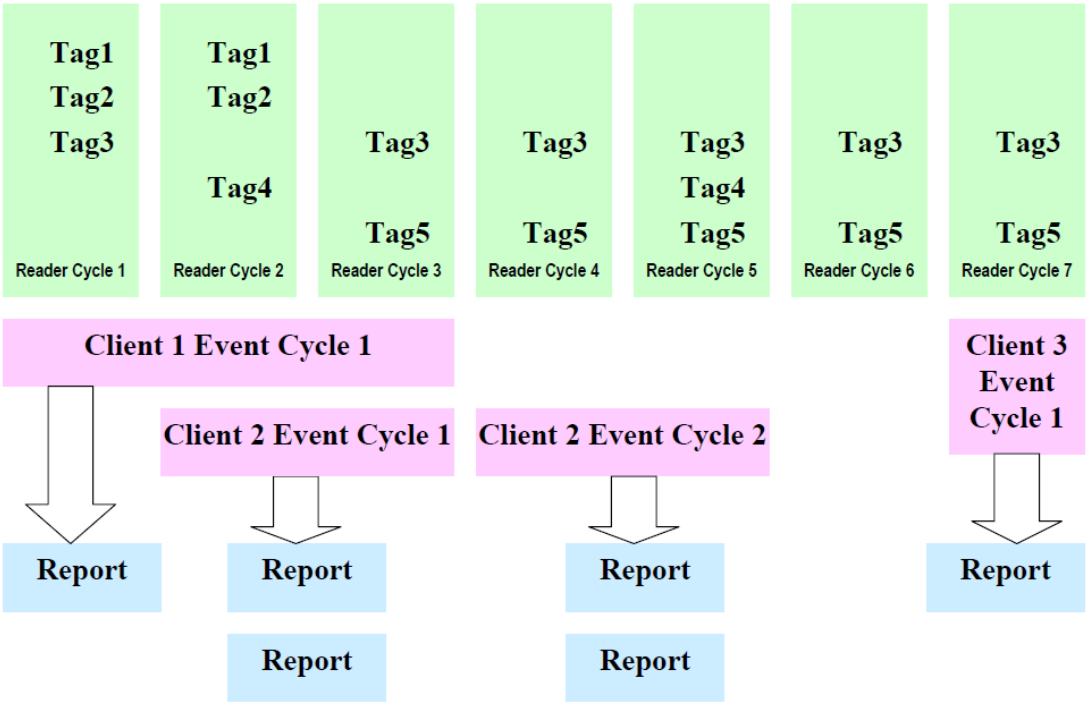


Εικόνα 6.9 : Immediate μέθοδος

6.5.3 Event Cycles και Command Cycles

Ένα event cycle ή ένα command cycle είναι ένα διάστημα χρόνου στο οποίο μία ALE Implementation αλληλεπιδρά με ένα ή περισσότερα Readers για ένα ALE Client.

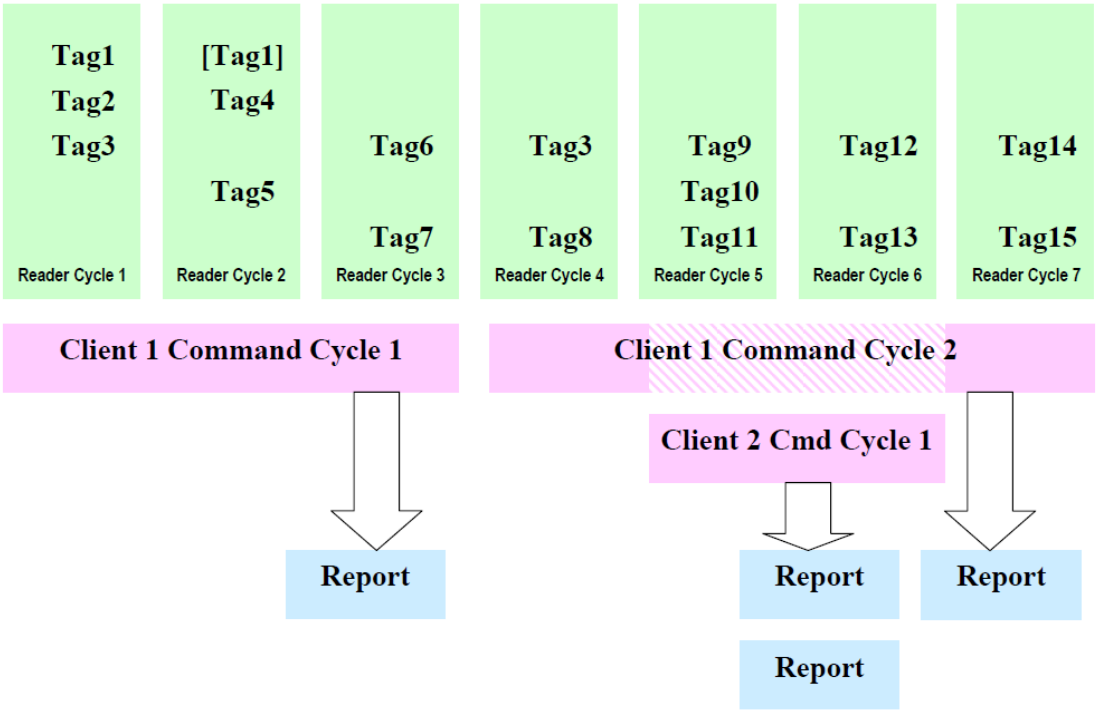
Ένα event cycle αποτελεί την μικρότερη μονάδα αλληλεπίδρασης μεταξύ ALE Client και ALE Implementation μέσω ALE Reading API. Μία οπτικοποίηση των event cycles παρουσιάζεται στην εικόνα 6.10.
Παράδειγμα λειτουργίας :
Ένα ALE Client στέλνει ένα ECSpec σε μία ALE Implementation για να περιγράψει ένα event cycle και ποιο/ποια ECReports θα πρέπει να παραχθούν. Περιέχει μία λίστα λογικών Readers (τα δεδομένα των οποίων θα περιέχονται στο event cycle), μία specification των χρονικών ορίων (συνθήκες εκκίνησης και διακοπής) του event cycle, καθώς επίσης και μία λίστα από specifications για το πως και πόσα reports πρέπει να παραχθούν για το event cycle αυτό.



Εικόνα 6.10 : Event cycles

Αντίστοιχα, ένα command cycle αποτελεί την μικρότερη μονάδα αλληλεπίδρασης μεταξύ ALE Client και ALE Implementation μέσω ALE Writing API. Μία οπτικοποίηση των command cycles παρουσιάζεται στην εικόνα 6.11.

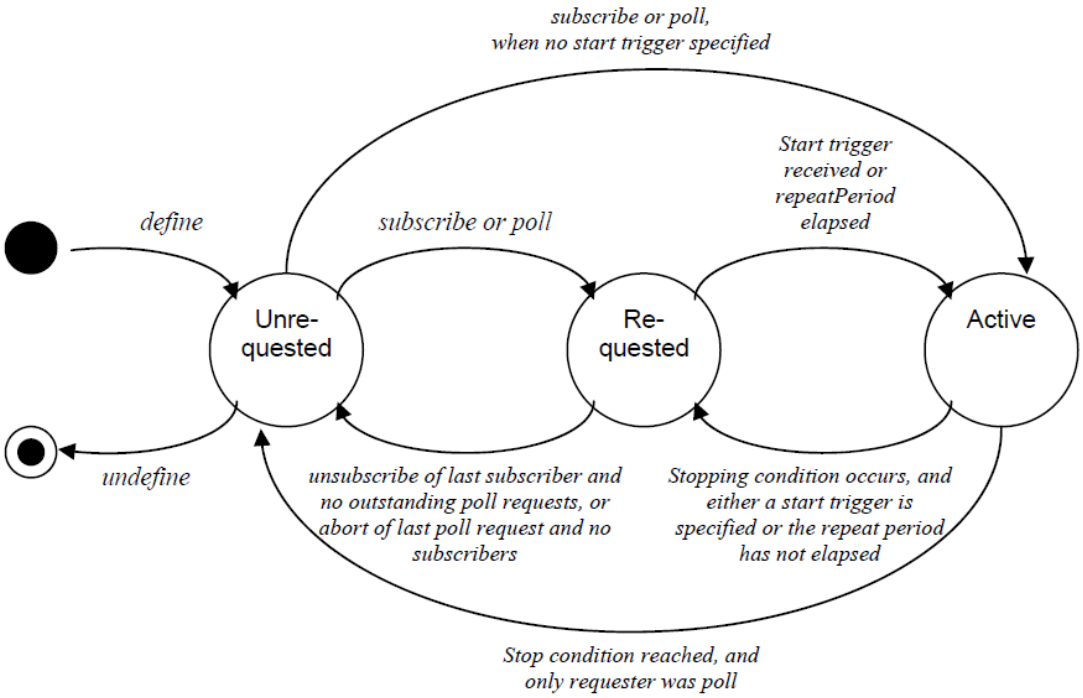
Παράδειγμα λειτουργίας :
Ένα ALE Client στέλνει ένα CCSpec σε μία ALE Implementation για να περιγράψει ένα command cycle και ποιο/ποια CCReports θα πρέπει να παραχθούν. Περιέχει μία λίστα λογικών Readers (τα Tags των οποίων θα διαχειριστούν στο command cycle), μία specification των χρονικών ορίων (συνθήκες εκκίνησης και διακοπής) του command cycle, καθώς επίσης και μία διατεταγμένη λίστα λειτουργιών που θα εφαρμοστούν στα Tags στο command cycle αυτό.



Εικόνα 6.11 : Command cycles

6.5.4 Lifecycle των EC/CCSpecs

Ο κύκλος ζωής των EC/CCSpecs περιγράφεται στο διάγραμμα κατάστασης της εικόνας 6.12. Οι τρεις καταστάσεις του περιγράφονται στον πίνακα 6.1.



Εικόνα 6.12 : Διάγραμμα κατάστασης του κύκλου ζωής ενός EC/CCSpec

Κατάσταση	**Περιγραφή**
Unrequested	Το EC/CCSpec έχει οριστεί αλλά κανένα client δεν έχει δηλώσει ενδιαφέρον κάνοντας subscribe ή poll.
Requested	Υπάρχει τουλάχιστον ένα client που έχει δηλώσει ενδιαφέρον γι' αυτό το EC/CCSpec, όμως τα Tags δεν βρίσκονται σε επεξεργασία για ένα event/command cycle.
Active	Τα Tags επεξεργάζονται για ένα event/command cycle.

Πίνακας 6.1 : Περιγραφή των τριών καταστάσεων του κύκλου ζωής των EC/CCSpec

Βιβλιογραφία

[1] Antoniou, G., Van Harmelen, F., *A Semantic Web Primer*, The MIT Press, Massachusetts, 2003

[2] McLaughlin, B., *Java & XML*, 2nd edition, O'Reilly, 2001

[3] Vajjhala, S., Fialli, J., *The Java™ Architecture for XML Binding (JAXB) 2.0*, Sun Microsystems Inc, Santa Clara, 2006

[4] Ort, E., Mehta, B., *Java Architecture for XML Binding (JAXB)*, Sun Developer Network, http://java.sun.com/developer/technicalArticles/WebServices/jaxb/ , 2003

[5] Haines, S., Potts, S., *Java 2 Primer Plus*, Sams, 2003

[6] Sun Microsystems, *The Java™ Web Services Tutorial For Java Web Services Developer's Pack v2.0*, Sun Microsystems Inc, Santa Clara, 2006

[7] Chinnici, R., Hadley, M., Mordani, R., *The Java API for XML Web Services (JAX-WS) 2.0*, Sun Microsystems Inc, Santa Clara, 2005

[8] Umbarje, M., *Developing Web Services Using JAX-WS*, http://www.java-tips.org/java-ee-tips/java-api-for-xml-web-services/developing-web-services-using-j.html, 2005

[9] Sun Microsystems, *The Java EE 5 Tutorial*, http://java.sun.com/javaee/5/docs/tutorial/doc/bnayl.html, 2007

[10] GS1 France, *RFID et EPCglobal*, http://www.gs1.fr/gs1_fr/standards_gs1__1/rfid_et_epcglobal, 2007

[11] Armenio, F., et al., *The EPCglobal Framework Architecture*, http://www.epcglobalinc.org/standards/architecture/architecture_1_2-framework-20070910.pdf, 2007

[12] EPCglobal, *The Application Level Events (ALE) Specification, Version 1.1*, http://www.epcglobalinc.org/standards/ale/ale_1_1-standard-core-20080227.pdf, 2008

[13] EPCglobal, *The Application Level Events (ALE) Frequently Asked Questions*, http://www.epcglobalinc.org/standards/ale/ale_1_1-faq-20080305.pdf, 2008

[14] EPCglobal, *The Application Level Events (ALE) Presentations*, http://www.epcglobalinc.org/standards/ale/ale_1_1-presentationOverview-20080305.pdf, 2008