2.1 Εισαγωγή κεφαλαίου

Στο κεφάλαιο αυτό θα αναλυθεί το θεωρητικό πλαίσιο της εργασίας εξετάζοντας προσεκτικά τύπους serializers που χρησιμοποιούνται στην βιομηχανία, θα γίνει μία σύντομη επισκόπηση των συστημάτων serialization προς ανάλυση στο κεφάλαιο 3, καθώς και μια σειρά άλλων σχετικών πηγών. Επιπλέον, θα οροθετηθεί η έννοια των Foreign Function Interfaces, οι λειτουργίες τους άλλα και επικείμενα σοβαρά ζητήματα του σχεδιασμού τους. Επιπροσθέτως, θα αναλυθούν οι δύο βασικές κατηγορίες των wrapper και ποια η θεωρεία τους και ο ορισμός τους. Τέλος, θα γίνει μία ανάλυση των μηχανισμών serialization και saving συστημάτων που περιέχονται στις μηχανές δημιουργίας βιντεοπαιχνιδιών Unity και Unreal Engine.

2.2 Persistent Data στην Πληροφορική

Η «διατήρηση δεδομένων»(data preservation) στην Πληροφορική είναι η διαδικασία διατήρησης αρχείων και δεδομένων σε ένα στάδιο που τα καθιστά προσβάσιμα καθ’ όλη τη διάρκεια του χρόνου. Τα δεδομένα αυτά θα πρέπει να αποθηκεύονται σε μορφές αρχείων που θα τα καθιστούν πιο χρήσιμα στο μέλλον, θα πρέπει να φυλάσσονται σε πολλές τοποθεσίες και τέλος να διατηρούνται σε ένα προστατευμένο «περιβάλλον» προκειμένου να διατηρηθούν(ref).

Συγκεκριμένα, τα μόνιμα δεδομένα(persistent data) αναφέρονται σε δεδομένα που διατηρούνται επ' αόριστόν σε μη πτητικές συσκευές αποθήκευσης(non-volatile storage), όπως μαγνητικές ταινίες, σκληρούς δίσκους, Solid State Drives αλλά και οπτικούς δίσκους. Τα δεδομένα αυτά διατηρούνται ακόμη και μετά την απενεργοποίηση της συσκευής, δηλαδή χωρίς ρεύμα, από όπου προέρχεται και η ονομασία τους Non-Volatile Memory. Φυσικά, χρησιμοποιούνται πολλές τακτικές για να εξασφαλιστεί η προσβασιμότητα και η μακροπρόθεσμη διατήρησή τους. Χρησιμοποιούνται συστήματα αρχείων για την οργάνωση των δεδομένων(file systems), εφαρμόζονται τεχνικές δημιουργίας αντιγράφων ασφαλείας, όπως διαφορικά(differential), αυξητικά(incremental) ή πλήρη(complete) αντίγραφα ασφαλείας, τα οποία αποθηκεύονται εντός, εκτός ή στο cloud, και γίνεται αντιγραφή των δεδομένων σε διάφορες συσκευές ή τοποθεσίες για λόγους ανθεκτικότητας. (ref)

2.3 Data Serialization

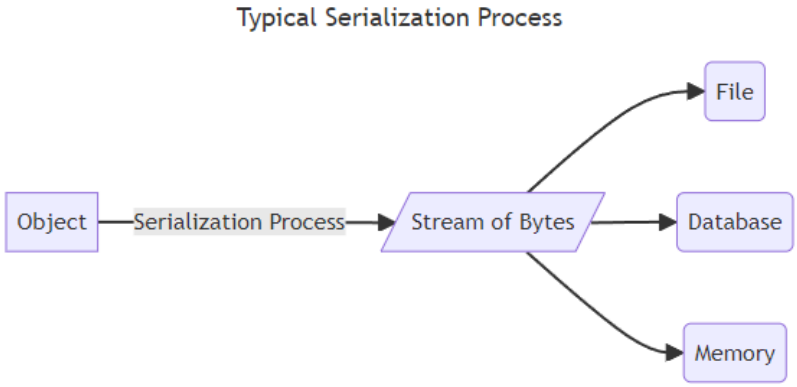
Στην υποενότητα αυτή γίνεται μία σύντομη ανασκόπηση στους ορισμούς του Serialization και Deserialization όπως αυτοί χρησιμοποιούνται στην Πληροφορική (εν. 2.3.1) και εξηγούνται οι βασικοί τύποι serialization όπως είναι οι XML, JSON, YAML και binary (εν. 2.3.2). Στη συνέχεια, γίνεται μία επεξήγηση της διαδικασίας του serialization και συγκεκριμένα της εξάλειψης των object reference στη μνήμη του υπολογιστή κατά τη διαδικασία αυτή (εν. 2.3.3). Τέλος, γίνεται μία επισκόπηση των επιλεγμένων serializer προς ανάλυση (εν. 2.3.5) όπως αυτή εμφανίζεται στο κεφάλαιο της Μεθοδολογίας (κεφ. 3).

2.3.1 Ορισμός του Serialization και Deserialization

Στον κόσμο της Πληροφορικής, η διαδικασία της μετατροπής δεδομένων σε μία μορφή κατανοητή από έναν υπολογιστή ονομάζεται Serialization. Αναλυτικότερα, όπως διακρίνεται στην Εικόνα 1, η σειριοποίηση δεδομένων(data serialization) αναφέρεται στην πράξη του μετασχηματισμού περίπλοκων δομών δεδομένων ή καταστάσεων αντικειμένων (ref) από τη κατάσταση τους στη μνήμη σε έναν πίνακα από bytes ή σε μια μορφή που μπορεί εύκολα να καταγραφεί σε ένα αρχείο, να μεταφερθεί και να ξαναδημιουργηθεί αργότερα μέσω της αντίστροφης διαδικασίας, ονόματη Deserialization. Συγκεκριμένα, το Deserialization είναι η διαδικασία μετάφρασης του αποθηκευμένου πίνακα ή αποθηκευμένης σειράς από bytes στην μορφή που ήταν πριν το serialization, κάτι που καθιστά τη διαδικασία του deserialization άμεσα συνδεδεμένη με τη διαδικασία του serialization. (ref)

Επιπλέον, είναι σημαντικό να αναφερθεί πως ο Serializer είναι κάτι διαφορετικό από το Serialization Format που εξάγει μετά τη διαδικασία του serialization. Το serialization format που εξάγει κάθε serializer βασίζεται σε συγκεκριμένη σύνταξη αναπαράστασης των serialized δεδομένων είτε αυτό είναι ένα JSON string, ένα YAML ή XML structure είτε είναι η binary αναπαράσταση αυτών. (ref)

Τέλος, κάθε serializer και με τη σειρά της η μορφή serialized δεδομένων που εξάγει έχει πλεονεκτήματα και εφαρμογές, όπου αυτά μπορούν να κυμαίνονται από το ποσοστό δυνατότητας ανθρώπινης ανάγνωσης μέχρι και το τελικό μέγεθος του παραχθέντος αρχείου για λόγους αποθήκευσης και μεταφοράς.(ref)



Εικόνα 1: Serialization process

2.3.2 Serialization formats

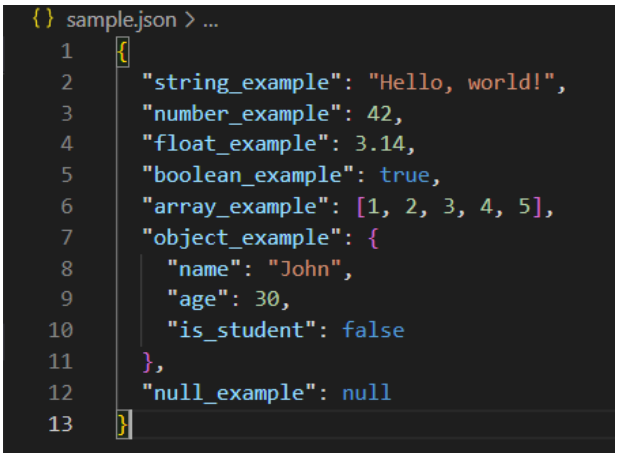
Η ακριβής σύνταξη και η δομή που χρησιμοποιείται για την αναπαράσταση των serialized δεδομένων καθορίζονται από τη μορφή του serialization. Οι μορφότυποι serialization εμφανίζουν ένα εύρος χαρακτηριστικών, όπως αναγνωσιμότητα, αποδοτικότητα και πολυπλοκότητα. Τα JSON, YAML, XML και Message Pack (binary) είναι μερικά παραδείγματα από αυτά τα μορφότυπα. Κάθε μορφή serialization περιγράφει κανόνες για την αναπαράσταση τύπων δεδομένων, την οργάνωση δομών δεδομένων και τον χειρισμό ειδικών περιπτώσεων όπως nested objects ή πίνακες, παρέχοντας έτσι μία σχεδίαση (schema) για το serialization δεδομένων σε διαφορετικά συστήματα και πλατφόρμες. (ref)

Τέλος, τα formats JSON, YAML, XML και binary διαθέτουν το καθένα μοναδικές ικανότητες προσαρμοσμένες σε συγκεκριμένες περιπτώσεις χρήσης, καλύπτοντας ποικίλες απαιτήσεις

αναπαράστασης (data presentation), μετάδοσης (transmission) και αποθήκευσης δεδομένων (data preservation).

2.3.2a JSON Serialization

Ιστορικά, η μορφή JSON ή αλλιώς JavaScript Object Notation, είναι βασισμένη στο ανοικτό πρότυπο ECMA-262 3rd Edition 12/1999 της JavaScript (ref ECMA-262) από όπου και δημιουργήθηκε το «ECMA-404 The JSON Data Interchange Standard». Σκοπός της δημιουργίας του ήταν η μετατροπή των αναπαραστάσεων δεδομένων σε μορφή απλού κειμένου που διαβάζονται από τον άνθρωπο σε αντικείμενα ECMAScript. Οι συμβολισμοί που χρησιμοποιεί είναι παρόμοιοι με εκείνους των κοινών γλωσσών προγραμματισμού όπως η C, η C++ και η Java και είναι εντελώς ανεξάρτητη από τις άλλες γλώσσες προγραμματισμού, έτσι αποτελεί μια κατάλληλη επιλογή για τη μετάδοση δεδομένων μεταξύ συστημάτων λόγω της αναγνωσιμότητας και της απλότητάς γραφής του (ref ECMA-404). Η ομαλή ενσωμάτωση καθίσταται δυνατή χάρη στον ελαφρύ πηγαίο σχεδιασμό του και την εγγενή υποστήριξη για την πλειονότητα των γλωσσών προγραμματισμού όπως προαναφέρθηκε. Το JSON λειτουργεί καλά σε καταστάσεις όπως τα διαδικτυακά API και τα αρχεία ρυθμίσεων, όπου είναι απαραίτητη η εύκολη ανάγνωση από τον άνθρωπο και η οργάνωση των δεδομένων. Όπως φαίνεται στην Εικόνα 2, τα δεδομένα αποθηκεύονται σαν ζεύγη κλειδιών-τιμών (key-value pairs) και οι διαθέσιμοι τύποι δεδομένων προς αποθήκευση κυμαίνονται στα strings, στους integer/floating-point αριθμούς, στις Boolean τιμές, στις λίστες τακτοποιημένων τιμών (ordered lists), στις συλλογές μη τακτοποιημένων ζευγών κλειδιών-τιμών (unordered key-value collections) και τέλος, την αναπαράσταση της απουσίας τιμής μέσω του keyword null. (ref) Αξίζει να σημειωθεί πως οι συλλογές με μη τακτοποιημένα ζεύγη κλειδιών-τιμών (unordered key-value collections) και αυτά εσωτερικά μπορούν με τη σειρά τους να αναπαραστήσουν δεδομένα με τους προαναφερθέντες τύπους. (ref)



Εικόνα 2: Δείγμα JSON αρχείου

2.3.2β XML Serialization

Η eXtensible Markup Language (XML) είναι μια απλή, πολύ ευέλικτη μορφή κειμένου που προέρχεται από την Standard Generalized Markup Language (SGML) του ISO 8879 (ref ISO 8879). Αρχικά σχεδιάστηκε από μία ομάδα του World Wide Web Consortium (W3C) το 1998 για να ανταποκριθεί στις προκλήσεις των μεγάλης κλίμακας ηλεκτρονικών εκδοτικών οίκων, όμως πλέον

παίζει επίσης ολοένα και πιο σημαντικό ρόλο στην ανταλλαγή μιας μεγάλης ποικιλίας δεδομένων στο Ίντερνετ και αλλού. (ref from W3C).

Η XML, αναπαριστά τα δομημένα δεδομένα με ετικέτες (tags) που περικλείονται σε αγκύλες. Αυτές οι ετικέτες οριοθετούν τη δομή των δεδομένων και μπορούν να περιλαμβάνουν χαρακτηριστικά και nested στοιχεία (elements). Κατά το serialization, τα αντικείμενα ή οι δομές δεδομένων μετατρέπονται σε μία μορφή XML με βάση το επιλεγμένο schema. Αυτό περιλαμβάνει τη διερεύνηση των ιδιοτήτων (attributes) ή των πεδίων (fields) του αντικειμένου και τη δημιουργία των αντίστοιχων στοιχείων και χαρακτηριστικών XML για την ακριβή αναπαράσταση των δεδομένων. Οι δομές XML Schema Definition (XSD) και Document Type Definition (DTD) προσφέρουν επίσημες προδιαγραφές για τον ορισμό της δομής και των περιορισμών των εγγράφων που δημιουργούνται (ref). Επιπλέον, η XML επιτρέπει την δημιουργία custom δομών όπου μερικές από τις αλλαγές που μπορούν να επισημανθούν είναι το τελικό XML format που θα δημιουργηθεί, η διαχείριση ειδικών τύπων δεδομένων, ο καθορισμός συμβάσεων ονοματοδοσίας και η διαχείριση των namespaces. (ref) Σαν serialization format, η XML βρίσκει ευρεία χρήση στις υπηρεσίες ιστού για την ανταλλαγή δεδομένων μεταξύ πελατών και διακομιστών. Το πρωτόκολλο Simple Object Access Protocol (SOAP), για παράδειγμα, αξιοποιεί την XML για τη μορφοποίηση μηνυμάτων που μοιράζονται μεταξύ των κατανεμημένων του συστημάτων (distributed systems).

Τέλος, είναι άξιο να σημειωθεί πως η XML μοιράζεται ομοιότητες με την HTML, καθώς και οι δύο είναι γλώσσες σήμανσης (markup languages) που χρησιμοποιούνται για τη δόμηση και την οργάνωση του περιεχομένου. Και οι δύο χρησιμοποιούν ετικέτες (tags) που περικλείονται σε αγκύλες για να ορίσουν τα στοιχεία μέσα σε ένα έγγραφο. Ωστόσο, η XML διαφέρει από την HTML στο ότι είναι πιο ευέλικτη και επεκτάσιμη, επιτρέποντας τη δημιουργία προσαρμοσμένων ετικετών και δομών εγγράφων προσαρμοσμένων σε συγκεκριμένες ανάγκες αναπαράστασης δεδομένων. Ενώ η HTML χρησιμοποιείται κυρίως για την εμφάνιση περιεχομένου στον ιστό, η XML χρησιμοποιείται για την αποθήκευση, τη μετάδοση και την ανταλλαγή δεδομένων σε διαφορετικά συστήματα και πλατφόρμες, επηρεασμένη από την προσέγγιση της HTML για τη σήμανση (markup) και τη δόμηση των δεδομένων. (ref)

Ως προς τους τύπους δεδομένων που υποστηρίζονται από την XML, όπως αυτά αναπαρίστανται στην Εικόνα 3, οι πιο βασικοί είναι το text το οποίο δέχεται από απλό κείμενο μέχρι και HTML markup μέσα σε ενότητες CDATA, αριθμούς όπως integers, floating-point αριθμούς, δεκαδικούς αριθμούς αλλά και επιστημονική σημειογραφία (scientific notation). Επιπλέον, υποστηρίζονται τιμές Boolean είτε σαν True-False είτε με 1 για True και 0 για False αλλά και η δυνατότητα αποθήκευσης δεδομένων σχετικά με την χρονολογία και την ώρα με βάση κάποιο format, όπως για παράδειγμα το "YYYY-MM-DD" για την χρονολογία (ISO 8601) (ref). Παρόλο που η XML είναι ένα format με βάση το κείμενο (text based) υποστηρίζει την αναπαράσταση δυαδικών δεδομένων (binary data) με τη χρήση τεχνικών όπως η κωδικοποίηση σε Base64 (ref) τα οποία και αποθηκεύονται σαν text μέσα στα XML στοιχεία. Επιπροσθέτως, προσφέρει ιεραρχικές διατάξεις στοιχείων (elements) και χαρακτηριστικών (attributes), διευκολύνοντας τη δημιουργία περίπλοκων δομών δεδομένων. Ακόμη, παρέχει στους χρήστες τη δυνατότητα να προσαρμόζουν την αναπαράσταση των δεδομένων, ορίζοντας προσαρμοσμένους τύπους δεδομένων με τη χρήση elements και attributes, ικανοποιώντας έτσι τις απαιτήσεις συγκεκριμένων τομέων. Τέλος, ενισχύει την ολοκληρωμένη διαχείριση δεδομένων, επιτρέποντας τη συμπερίληψη μεταδεδομένων (metadata ή information about data) στα έγγραφα. Αυτά τα metadata περιλαμβάνουν κρίσιμες λεπτομέρειες όπως το συγγραφικό δικαίωμα, οι ημερομηνίες δημιουργίας, οι πληροφορίες

έκδοσης και άλλα συναφή metadata, ενισχύοντας έτσι την κατανόηση του πλαισίου και τη διαχείριση των δεδομένων. (ref)

codeSamples > </> sample.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<data>
    <!-- Textual data -->
    <name>John Doe</name>

    <!-- Numeric data -->
    <age>30</age>

    <!-- Boolean value -->
    <is_student>true</is_student>

    <!-- Date and time -->
    <registration_date>2024-04-05T08:00:00</registration_date>

    <!-- Binary data (Base64 encoded) -->
    <profile_picture>base64_encoded_data_here</profile_picture>

    <!-- Structured data -->
    <address>
        <street>123 Main St</street>
        <city>New York</city>
        <state>NY</state>
        <zip_code>10001</zip_code>
    </address>

    <!-- Custom data type -->
    <product>
        <name>Laptop</name>
        <price>999.99</price>
        <manufacturer>XYZ Electronics</manufacturer>
    </product>

    <!-- Metadata -->
    <metadata>
        <author>John Smith</author>
        <creation_date>2024-04-05</creation_date>
        <version>1.0</version>
    </metadata>
</data>
```

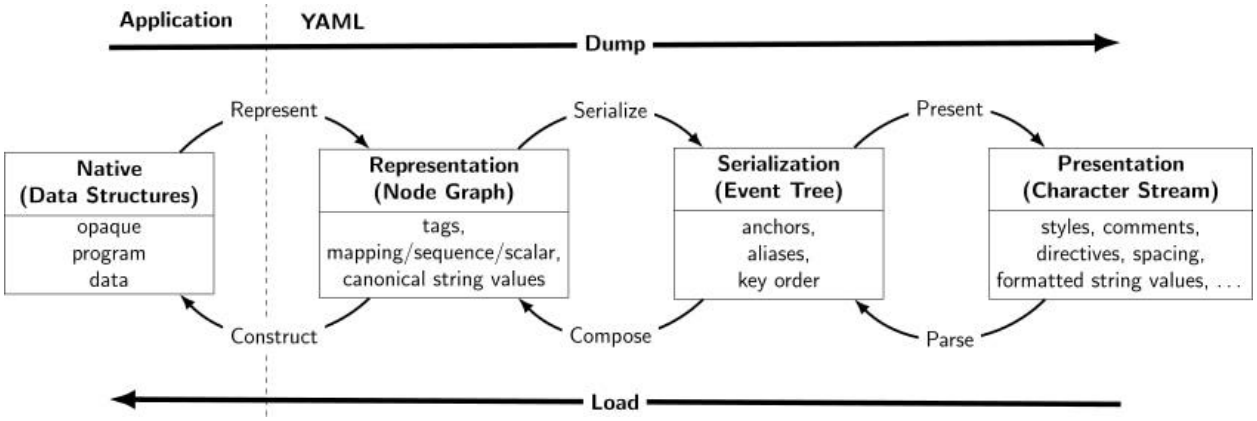
Εικόνα 3: Δείγμα XML αρχείου

2.3.2γ YAML Serialization

Η YAML Ain't Markup Language (YAML) είναι ένα serialization format δεδομένων που χρησιμοποιείται για την αναπαράσταση δομημένων δεδομένων (structured data). Λόγο της εύκολης ανάγνωσης του από τον άνθρωπο χρησιμοποιείται συχνά σε αρχεία ρυθμίσεων, στην ανταλλαγή δεδομένων μεταξύ προγραμμάτων και σε εφαρμογές όπου η αναγνωσιμότητα από τον άνθρωπο αποτελεί προτεραιότητα.

Ιστορικά, η YAML εμπνευσμένη από την XML, δημιουργήθηκε το 2001 από μία ομάδα ερευνητών με κύριους στόχους την εύκολη ανάγνωση της από τον άνθρωπο, την εύκολη μεταφορά και μετάδοση δεδομένων μεταξύ διαφόρων γλωσσών προγραμματισμού και την ευκολία χρήσης της. (ref) Παρόλο που η YAML δεν στηρίζεται σε κάποιο προ υπάρχων standard, όπως η JSON και η XML (ref), μερικά βασικά χαρακτηριστικά του σχεδιασμού της περιέχουν την προσπάθεια ομοιογένειας μεταξύ των τύπων δεδομένων της με τις εγγενείς δομές δεδομένων (native data structures) που έχουν οι δυναμικές γλώσσες προγραμματισμού (dynamic languages), τη σχεδίαση ενός συνεπής μοντέλου το οποίο θα υποστηρίζει γενικά εργαλεία (generic tools) και την έμφαση στην εκφραστικότητα και την επεκτασιμότητα της.

Τέλος, ο μηχανισμός επεξεργασίας της με βάση την επίσημη έρευνα σχεδιασμού της, όπως αυτή αναπαρίσταται στην Εικόνα 4, χαρακτηρίζεται σαν μηχανισμός “one-pass processing” (ref). Αυτό σημαίνει ότι ο αναλυτής (parser) της YAML διαβάζει τα δεδομένα μόνο μία φορά από την αρχή έως το τέλος και τα επεξεργάζεται καθώς τα συναντά. (ref)



Εικόνα 4: Επισκόπηση επεξεργασίας YAML (ref from book)

Επιγραμματικά, οι τύποι δεδομένων που υποστηρίζει η YAML ως προς την αποθήκευση τους, όπως αυτά αναπαρίστανται στην Εικόνα 5, οι πιο βασικοί είναι οι κλίμακες (scalars) οι οποίες μπορούν να αποθηκεύσουν strings και multi-line strings, integers και floating-point αριθμούς, τιμές Boolean αλλά και την απουσία τιμής είτε με το keyword null ή με το σύμβολο ~ . Επιπλέον, υποστηρίζονται οι διατεταγμένες συλλογές αντικειμένων (ordered item collections) ή όπως τις ονομάζει η YAML, τα Sequences όπου μπορούν να περιέχουν κάθε αναφερόμενο τύπο δεδομένων, τις συλλογές (Mappings) από ζεύγη κλειδιών-τιμών (key-value pairs) όπου το κλειδί πρέπει να είναι μοναδικό μέσα στο σύνολο του Mapping αλλά η τιμή μπορεί να περιέχει ακόμα και nested sequences ή άλλα mappings. Τέλος η YAML περιέχει και τα λεγόμενα Anchors και References. Συγκεκριμένα, το χαρακτηριστικό των anchors και των references που διακρίνεται στη YAML δεν αναφέρεται στα memory address references των αντικειμένων που μόλις έγιναν serialized αλλά χρησιμοποιούνται για τη δημιουργία ψευδώνυμων (aliases) στο ίδιο το YAML αρχείο. Μέσω της χρήσης τους, αποφεύγεται η διπλοτυπία διατύπωσης των δεδομένων μέσα στο αρχείο και αυξάνεται η ευκολία ανάγνωσης του. Όταν ορίζεται ένα anchor και μετά γίνεται reference αυτού σε κάποιο σημείο μέσα στο YAML αρχείο, ουσιαστικά δίνεται η εντολή στον parser να χειριστεί αυτές τις εμφανίσεις σαν πανομοιότυπες ως προς το περιεχόμενο τους. Εν γένει, είναι ένας μηχανισμός για την επαναχρησιμοποίηση δομών δεδομένων εντός του ίδιου αρχείου YAML, αλλά δεν δημιουργεί καμία σύνδεση με τα αρχικά serialized αντικείμενα ή δομές δεδομένων εκτός του πλαισίου του αρχείου. (ref)

sample.yaml

```yaml
# Scalars
name: John Doe
age: 30
is_student: false
null_value: null

# Sequences
hobbies:
  - Reading
  - Hiking
  - Cooking

# Mappings
address:
  city: New York
  street: 123 Main St
  zip_code: "10001"

# Anchors and References
person1: &person
  name: Alice
  age: 25

#Reference to anchor &person
person2: *person

```

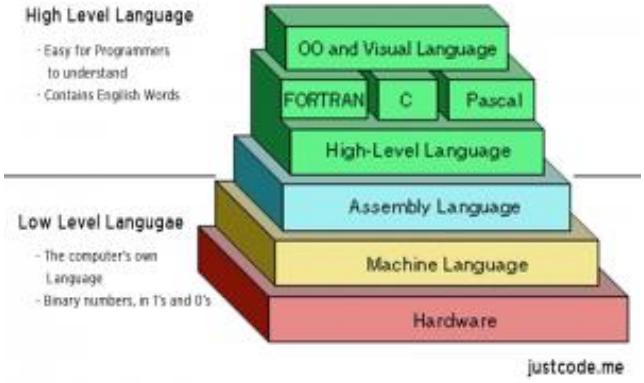
Εικόνα 5: Δείγμα αρχείου YAML

2.3.2δ Binary Serialization

Η δυαδική σειριοποίηση (binary serialization) είναι μια διαδικασία μετατροπής δομών δεδομένων ή αντικειμένων σε δυαδική μορφή (binary), ώστε να μπορούν να αποθηκευτούν, να μεταδοθούν ή να ανακατασκευαστούν αποτελεσματικά αργότερα. Σε αντίθεση με τις μορφές JSON ή XML και YAML, οι οποίες είναι δυνατό να αναγνωστούν από έναν άνθρωπο, το binary serialization κωδικοποιεί τα δεδομένα σε μια συμπαγή, αναγνώσιμη μόνο από μηχανήματα μορφή, η οποία είναι ιδιαίτερα χρήσιμη σε σενάρια όπου η αποδοτικότητα, η ταχύτητα, το τελικό μέγεθος των serialized δεδομένων ή η μειωμένη χρήση bandwidth είναι σημαντικά ζητήματα.

Αρχικά, σε αντίθεση με τα προαναφερθέντα serialization formats και serializers, το binary serialization ως έννοια, υπάρχει εγγενώς λόγω της φύσης των υπολογιστών που χειρίζονται δυαδικά δεδομένα. Από τις πρώτες ημέρες της πληροφορικής, οι προγραμματιστές χρειάζονταν τρόπους

αποθήκευσης και μετάδοσης δεδομένων σε μορφή που οι υπολογιστές μπορούσαν να κατανοήσουν αποτελεσματικά. Οι ρίζες του binary serialization μπορούν να εντοπιστούν στην πρώιμη ανάπτυξη των συστημάτων υπολογιστών και των γλωσσών προγραμματισμού. (ref) Καθώς οι υπολογιστές εξελίσσονταν και γίνονταν πιο ισχυροί, οι προγραμματιστές επινόησαν μεθόδους για την αναπαράσταση δεδομένων σε δυαδική μορφή για τη βελτιστοποίηση του χώρου αποθήκευσης, τη βελτίωση της απόδοσης και τη διευκόλυνση της επικοινωνίας μεταξύ διαφορετικών συστημάτων. Στις αρχές της πληροφορικής, το binary serialization ήταν συχνά χειροκίνητα κωδικοποιημένη, με τους προγραμματιστές να κωδικοποιούν και να αποκωδικοποιούν χειροκίνητα δομές δεδομένων σε δυαδική μορφή χρησιμοποιώντας τεχνικές χαμηλού επιπέδου (low-level), όπως bit-manipulation και οι πράξεις σε επίπεδο byte (ref). Καθώς οι γλώσσες προγραμματισμού και οι πρακτικές ανάπτυξης λογισμικού ωρίμαζαν, αναπτύχθηκαν αφαιρετικές (abstractions) μέθοδοι και βιβλιοθήκες υψηλότερου επιπέδου (high-level) για το binary serialization για την απλούστευση της διαδικασίας και τη βελτίωση της παραγωγικότητας. Στην Εικόνα 6 διακρίνονται οι έννοιες low-level και high-level.



Εικόνα 6: Low-Level και High-Level έννοιες σε γράφημα (ref)

Ισχυρά παραδείγματα είναι οι γλώσσες C και Pascal, οι οποίες περιείχαν low-level εντολές για την ανάγνωση και γραφή δυαδικών δεδομένων σε αρχεία, ενώ μεταγενέστερες γλώσσες όπως η Java και η C# εισήγαγαν ενσωματωμένα frameworks για serialization όπως το ObjectOutputStream/ObjectInputStream (ref) και ο BinaryFormatter (ref) αντίστοιχα, για την αυτοματοποίηση της διαδικασίας του serialization και τον χειρισμό πολύπλοκων δομών αντικειμένων.

Επιπροσθέτως, με βάση αυτές τις αρχές οι binary serializers που έχουν δημιουργηθεί μπορούν να υποστηρίξουν ένα μεγάλο εύρος τύπων δεδομένων όπως (ref):

- **Primitive** τύπους: Οι binary serializers μπορούν να χειριστούν Primitive τύπους δεδομένων, όπως ακέραιους αριθμούς (π.χ. int, long, short), αριθμούς κινητής υποδιαστολής (π.χ. float, double), χαρακτήρες (π.χ. char) και τιμές Boolean.
- **Composite** τύπους: Οι binary serializers μπορούν να χειριστούν Composite τύπους δεδομένων που αποτελούνται από πολλούς primitive τύπους. Σε αυτούς περιλαμβάνονται πίνακες, λίστες, maps (ή dictionaries), sets, tuples και άλλοι collection τύποι.
- **Custom Objects**: Προσαρμοσμένα αντικείμενα ή κλάσεις, οι οποίες μπορεί να περιέχουν ένα συνδυασμό πρωτόγονων τύπων, σύνθετων τύπων και references σε άλλα αντικείμενα μπορούν επίσης να γίνουν serialized από έναν binary serializer. Οι βιβλιοθήκες serialization συχνά παρέχουν μηχανισμούς για την προσαρμογή της διαδικασίας serialization αυτών των αντικειμένων είτε μέσω κάποιου configuration αρχείου ή κάποιου object schema.

- **Nullable** τύπους: Οι nullable τύποι ή οι τιμές στις οποίες μπορεί να ανατεθεί είτε ένα null reference είτε μια έγκυρη τιμή του υποκείμενου τύπου, διαχειρίζονται επίσης από τους binary serializers.
- **Strings**: Τα strings (πίνακες από char) είναι μία ιδιαίτερη περίπτωση όσον αφορά το binary serialization. Λόγω των διαφορετικών κωδικοποιήσεων όπως είναι οι UTF-8 και UTF-16 - μαζί με μία πληθώρα πολλών άλλον- συχνά οι binary serialization βιβλιοθήκες περιέχουν μηχανισμούς για τη διαχείριση του μεγέθους των string αλλά και υποκείμενα προβλήματα σχετικά με την επιλεγμένη κωδικοποίηση.

Το τελικό παραχθέν αρχείο, όπως διακρίνεται στην Εικόνα 7, δεν αποτελεί μία μορφή η οποία μπορεί να διαβαστεί από τον άνθρωπο και έτσι εάν κάποιος επιχειρήσει να το ανοίξει με κάποιον text editor όπως Notepad ή Notepad++ θα διακρίνει μία αλληλουχία γραμμάτων, συμβόλων, αριθμών ή και ακόμα μη κατανοητά από τον άνθρωπο σύμβολα. Αυτό το φαινόμενο συμβαίνει διότι το παραχθέν αρχείο περιέχει raw binary δεδομένα ή αλλιώς machine code (ref) τα οποία και είναι κατανοητά μόνο από έναν υπολογιστή. Με αυτό τον τρόπο παρέχεται και ένα μικρό επίπεδο ασφάλειας ως προς την ανάγνωση και τροποποίηση των δεδομένων, χωρίς όμως αυτό να αποτελεί κάποιο επίπεδο encryption (ref).

Εικόνα 7: Δείγμα binary αρχείου

Τέλος, στην Εικόνα 8, φαίνεται η αναπαράσταση του δείγματος στην Εικόνα 7 μέσω του προγράμματος HxD32 (ref) που με βάση την επίσημη ιστοσελίδα του περιγράφεται ως:

«*HxD is a carefully designed and fast hex editor which, additionally to raw disk editing and modifying of main memory (RAM), handles files of any size*. »(ref)

sample.bin

Offset (h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	00	01	00	00	00	FF	FF	FF	FF	01	00	00	00	00	00	00
00000010	00	0C	02	00	00	00	42	43	6F	6E	73	6F	6C	65	41	70
00000020	70	31	2C	20	56	65	72	73	69	6F	6E	3D	31	2E	30	2E
00000030	30	2E	30	2C	20	43	75	6C	74	75	72	65	3D	6E	65	75
00000040	74	72	61	6C	2C	20	50	75	62	6C	69	63	4B	65	79	54
00000050	6F	6B	65	6E	3D	6E	75	6C	6C	05	01	00	00	00	10	53
00000060	65	72	69	61	6C	69	7A	61	62	6C	65	44	61	74	61	01
00000070	00	00	00	17	3C	50	68	72	61	73	65	3E	6B	5F	5F	42
00000080	61	63	6B	69	6E	67	46	69	65	6C	64	01	02	00	00	00
00000090	06	03	00	00	00	61	59	6F	75	20	61	63	74	75	61	6C
000000A0	6C	79	20	64	65	73	65	72	69	61	6C	69	7A	65	64	20
000000B0	6D	79	20	6D	61	6A	6F	72	20	73	61	6D	70	6C	65	20
000000C0	66	69	6C	65	2E	20	43	6F	6E	67	72	61	74	75	6C	61
000000D0	74	69	6F	6E	73	20	74	6F	20	79	6F	75	2C	20	62	79
000000E0	20	4D	41	44	20	66	6F	72	20	53	41	45	20	77	69	74
000000F0	68	20	6C	6F	76	65	21	0B								

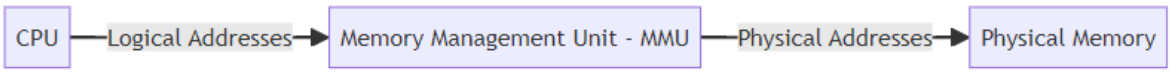
Εικόνα 8: Inspection του αρχείου sample.bin με HxD32

2.4 Memory management, addresses και serialization

Στην ενότητα αυτή γίνεται μία σύντομη ανάλυση των τρόπων με τους οποίους ένα υπολογιστικό σύστημα διαχειρίζεται τη μνήμη που του παρέχεται. Γίνεται μία αναφορά στις λογικές και φυσικές διευθύνσεις που παράγει ο επεξεργαστής, τις λειτουργίες διαχείρισης μνήμης και εικονικής μνήμης και τέλος εξηγείται τι συμβαίνει σε αυτές τις διευθύνσεις κατά τη διαδικασία του serialization.

2.4.1 Logical και Physical addresses

Στον τομέα των υπολογιστικών, οι λογικές διευθύνσεις (logical addresses) και οι φυσικές διευθύνσεις (physical addresses) έχουν έναν σημαντικό ρόλο στη διαδικασία διαχείρισης της μνήμης. Οι λογικές διευθύνσεις είναι εικονικές διευθύνσεις (virtual addresses) που δημιουργούνται από την CPU και αντιπροσωπεύουν θέσεις στο λογικό χώρο διευθύνσεων που είναι διαθέσιμες και προσβάσιμες σε ένα πρόγραμμα. Αυτές οι διευθύνσεις στην πραγματικότητα, αποτελούν μία αφηρημένη (abstracted) και ανεξάρτητη έννοια από την πραγματική διαμόρφωση της φυσικής μνήμης, παρέχοντας ένα σταθερό περιβάλλον διεπαφής για την αλληλεπίδραση των προγραμμάτων με τη μνήμη (ref). Από την άλλη πλευρά, οι φυσικές διευθύνσεις αναφέρονται στις φυσικές θέσεις στο hardware μνήμης του υπολογιστή, όπου αποθηκεύονται τα δεδομένα. Αντιστοιχούν άμεσα στα πραγματικά memory cells του hardware, καθορίζοντας τις πραγματικές θέσεις στις οποίες βρίσκονται τα δεδομένα. Η μετάφραση μεταξύ των λογικών διευθύνσεων και των φυσικών διευθύνσεων διαχειρίζεται από τη μονάδα διαχείρισης μνήμης (Memory Management Unit - MMU) του λειτουργικού συστήματος, η οποία αντιστοιχίζει τις λογικές διευθύνσεις στις αντίστοιχες φυσικές διευθύνσεις, διευκολύνοντας την αποτελεσματική πρόσβαση και διαχείριση της μνήμης. Αυτή η διαδικασία αναπαρίσταται στην Εικόνα 9. Αυτό το επίπεδο αφαίρεσης μεταξύ λογικών και φυσικών διευθύνσεων επιτρέπει την ευέλικτη κατανομή, προστασία και αργότερα, virtualization της μνήμης στα σύγχρονα συστήματα υπολογιστών (ref).



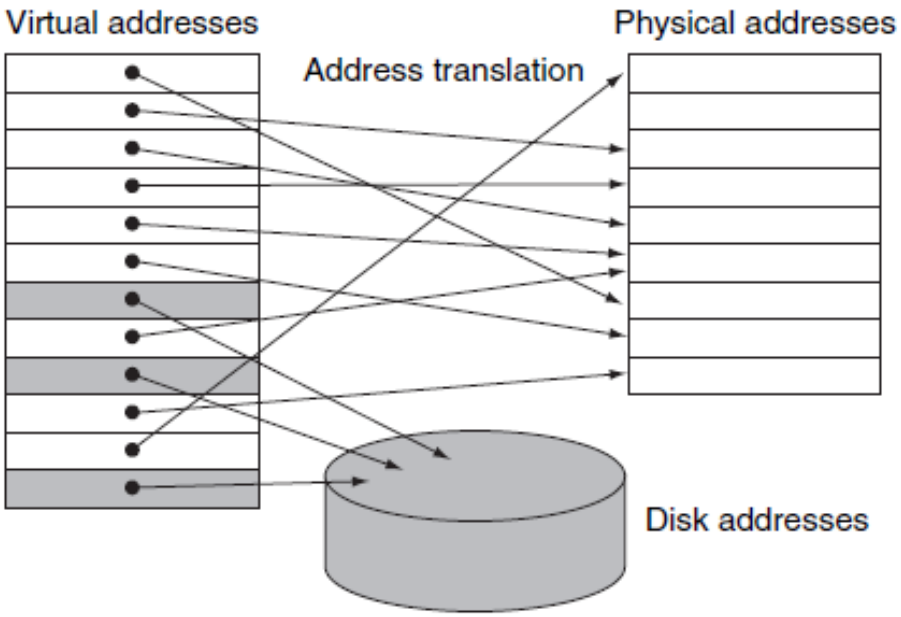
Εικόνα 9: Address binding on the MMU (ref from book 2022)

2.4.2 Memory Management και Virtual Memory σε ένα λειτουργικό σύστημα

Η διαχείριση της μνήμης σε ένα λειτουργικό σύστημα περιλαμβάνει την αποτελεσματική κατανομή και χρήση της μνήμης για την υποστήριξη των διεργασιών που εκτελούνται. Αυτό επιτυγχάνεται μέσω διαφόρων τεχνικών και schemes. Δύο βασικές τεχνικές που χρησιμοποιούνται στη διαχείριση της μνήμης είναι το paging και το segmentation. Το paging χωρίζει τη φυσική μνήμη σε μπλοκ σταθερού μεγέθους που ονομάζονται frames και τη λογική μνήμη σε μπλοκ σταθερού μεγέθους που ονομάζονται pages, επιτρέποντας την ευέλικτη κατανομή μνήμης και την αποτελεσματική χρήση της φυσικής μνήμης (ref). Όταν μια διεργασία (process) ζητά μνήμη, το λειτουργικό σύστημα κατανέμει ένα ή περισσότερα pages, τα οποία διαχειρίζεται μέσω ενός πίνακα σελίδων (page table). Εάν το ζητούμενο page δεν βρίσκεται στην κύρια μνήμη – όπως η RAM και η ROM - εμφανίζεται το λεγόμενο

page fault, προτρέποντας το λειτουργικό σύστημα να φέρει το page από τον δευτερεύοντα αποθηκευτικό χώρο – όπως HDDs, SSDs και παρόμοια αποθηκευτικά μέσα - στη μνήμη. Το segmentation, από την άλλη πλευρά, διαιρεί τη λογική μνήμη σε τμήματα μεταβλητού μεγέθους, που αντιπροσωπεύουν διαφορετικά τμήματα ενός προγράμματος (ref). Τα τμήματα μπορεί να περιλαμβάνουν κώδικα, δεδομένα, stack και heap. Ενώ το segmentation προσφέρει ευελιξία, εισάγει προκλήσεις όπως το λεγόμενο fragmentation. Το external fragmentation, όπως αποκαλείται, προκύπτει όταν η ελεύθερη μνήμη διαιρείται σε μικρά, μη συνεχόμενα μπλοκ, ενώ το internal fragmentation εμφανίζεται όταν τα τμήματα είναι μεγαλύτερα από τα απαραίτητα, οδηγώντας σε σπατάλη χώρου εντός των τμημάτων. Για την αντιμετώπιση αυτών των προκλήσεων, τα σύγχρονα λειτουργικά συστήματα χρησιμοποιούν συχνά έναν συνδυασμό paging και segmentation, γνωστό ως segmented paging, για να επιτύχουν αποτελεσματική διαχείριση της μνήμης. (ref)

Η εικονική μνήμη (virtual memory), όπως αυτή αναπαρίσταται στην Εικόνα 10, είναι μια θεμελιώδης έννοια στα λειτουργικά συστήματα που επιτρέπει την αποτελεσματική διαχείριση της μνήμης με την επέκταση της διαθέσιμης φυσικής μνήμης μέσω της χρήσης δευτερεύουσας αποθήκευσης, συνήθως ενός HDD ή ενός SSD. Στον πυρήνα της, η εικονική μνήμη επιτρέπει την εκτέλεση διεργασιών που ενδέχεται να μην χωρούν εξ ολοκλήρου στη διαθέσιμη φυσική μνήμη. Αντί να απαιτείται η ταυτόχρονη φόρτωση όλων των τμημάτων ενός προγράμματος στη μνήμη RAM, η εικονική μνήμη διαιρεί το χώρο διευθύνσεων μιας διεργασίας σε μικρότερες μονάδες που ονομάζονται pages ή segments. Αυτές οι μονάδες αντιστοιχίζονται στη συνέχεια δυναμικά μεταξύ της κύριας μνήμης και του δευτερεύοντος αποθηκευτικού χώρου από το λειτουργικό σύστημα μέσω μίας διαδικασίας ονόματη memory mapping, επιτρέποντας στη CPU να έχει πρόσβαση στα δεδομένα ανάλογα με τις ανάγκες της. Αυτή η προσέγγιση επιτρέπει την ψευδαίσθηση ενός τεράστιου και συνεχούς χώρου μνήμης, ακόμη και όταν η φυσική μνήμη είναι περιορισμένη, με τη άμεση ανταλλαγή δεδομένων μεταξύ της RAM και του δίσκου, όπως απαιτείται. Η εικονική μνήμη παίζει καθοριστικό ρόλο σε multitasking περιβάλλοντα, επιτρέποντας την ταυτόχρονη εκτέλεση πολλαπλών διεργασιών, βελτιστοποιώντας παράλληλα τη χρήση της μνήμης και την απόδοση της.



Εικόνα 10: Virtual memory representation (ref from book 2022)

2.4.3 Τα memory addresses κατά το serialization

Όπως αναλύθηκε στις προηγούμενες ενότητες, οι διευθύνσεις μνήμης είναι συγκεκριμένες για το περιβάλλον εκτέλεσης στο οποίο εκτελείται ένα πρόγραμμα και δεν έχουν νόημα εκτός αυτού λόγω της παροδικής τους φύσης. Σε ένα εκτελούμενο πρόγραμμα, οι διευθύνσεις μνήμης εκχωρούνται δυναμικά από MMU του λειτουργικού συστήματος καθώς οι δομές δεδομένων και οι μεταβλητές κατανέμονται στη μνήμη. Αυτές οι διευθύνσεις είναι σχετικές με το χώρο μνήμης του προγράμματος και μπορεί να αλλάζουν κάθε φορά που εκτελείται το πρόγραμμα ή ακόμη και κατά τη διάρκεια της εκτέλεσης του, καθώς η μνήμη κατανέμεται και αποδεσμεύεται δυναμικά. Κατά συνέπεια, οι διευθύνσεις μνήμης στερούνται της φορητότητας και δεν μπορούν να βασιστούν για μόνιμη αποθήκευση ή επικοινωνία μεταξύ διαφορετικών εκτελέσεων ενός προγράμματος ή μεταξύ διαφορετικών συστημάτων. Αντ' αυτού, οι διαδικασίες serialization, όπως προ αναλύθηκε στην ενότητα 2.3, επικεντρώνονται στην καταγραφή της κατάστασης των δεδομένων ενός αντικειμένου, όπως οι τιμές και η δομή του, η οποία μπορεί να ανακατασκευαστεί ανεξάρτητα από συγκεκριμένες διευθύνσεις μνήμης όταν γίνει deserialize σε άλλο περιβάλλον εκτέλεσης ή σύστημα.

2.5 Επισκόπηση Serializers προς ανάλυση

Στην ενότητα αυτή θα γίνει μία σχολαστική εξέταση των μεθόδων serialization δεδομένων όπου αποτελούν επιτακτική ανάγκη για τη βελτιστοποίηση της χρήσης των πόρων και τη βελτίωση της συνολικής απόδοσης ενός βιντεοπαιχνιδιού. Στο πλαίσιο αυτό, η επιλογή των κατάλληλων serializers αποκτά ύψιστη σημασία, ιδίως κατά την ανάπτυξη συστημάτων αποθήκευσης, τα οποία απαιτούν γρήγορους και αποτελεσματικούς μηχανισμούς αποθήκευσης και ανάκτησης δεδομένων. Μεταξύ της πληθώρας των διαθέσιμων επιλογών serialization, το MessagePack και το Protocol Buffers (protobuf) αναδεικνύονται για τη συμπαγή δομή, την ταχύτητα και την ευελιξία τους.

2.5.1 Protocol Buffers

Το Protocol Buffers (Protobuf) είναι μια μέθοδος serialization δομημένων δεδομένων, που αναπτύχθηκε από την Google, κάτι που τον καθιστά ιδιαίτερα δημοφιλή αλλά και σταθερό. Έχει σχεδιαστεί για να είναι ένας γρήγορος, αποδοτικός και γλωσσικά ουδέτερος μηχανισμός serialization δομημένων δεδομένων, καθιστώντας τον ιδανικό για πρωτόκολλα επικοινωνίας, αποθήκευση δεδομένων και συστήματα RPC (Remote Procedure Call). Αυτό το επιτυγχάνει με την δημιουργία των λεγόμενων “messages”, όπως αυτή φαίνεται στην Εικόνα 11, όπου και παίζουν κομβικό ρόλο στην συνολική λειτουργία του serializer (ref).

sample.proto

```
1   syntax = "proto3";
2
3   // Define a message called "Person"
4   message Person {
5     string name = 1;
6     int32 age = 2;
7     string email = 3;
8
9     // Nested message for address
10    message Address {
11      string street = 1;
12      string city = 2;
13      string postal_code = 3;
14    }
15
16    Address address = 4;
17
18    // Field for a list of phone numbers
19    repeated string phone_numbers = 5;
20  }
```

Εικόνα 11: Δείγμα .proto message αρχείου

Τα βασικά χαρακτηριστικά που επιλέχθηκε το Protobuf προς ανάλυση είναι τα παρακάτω:

- **Αποδοτικό serialization**: Το Protobuf χρησιμοποιεί μια binary μορφή serialization, η οποία είναι πιο συμπαγής και αποδοτική σε σύγκριση με άλλες μορφές όπως η XML, YAML ή η JSON. Αυτό έχει ως αποτέλεσμα - όπως και αναλύθηκε στην προηγούμενη ενότητα - μικρότερα μεγέθη αρχείων μειώνοντας έτσι τις απαιτήσεις αποθήκευσης.

- **Schema Definition Language**: Το Protobuf χρησιμοποιεί μία language-agnostic γλώσσα δημιουργίας schemas για να ορίσει τη δομή των δεδομένων που γίνονται serialize. Αυτή η γλώσσα δημιουργίας schema επιτρέπει τον ορισμό πεδίων, τους τύπους αυτών και την σειρά τους μέσα στα “messages”.

- **Data Streaming**: Το Protobuf υποστηρίζει data streaming, επιτρέποντας στις εφαρμογές να επεξεργάζονται αποτελεσματικά μεγάλα σύνολα δεδομένων χωρίς να φορτώνουν όλα τα δεδομένα στη μνήμη ταυτόχρονα.

- **Υποστήριξη πολλαπλών γλωσσών**: Υπάρχει υποστήριξη για πολλές γλώσσες προγραμματισμού, όπως C++, Java, Python, C# και πολλές άλλες. Αυτό επιτρέπει τον ορισμό των δομών δεδομένων μία φορά στα προ αναφερθέντα αρχεία .proto και την χρήση τους σε διαφορετικές πλατφόρμες και γλώσσες.

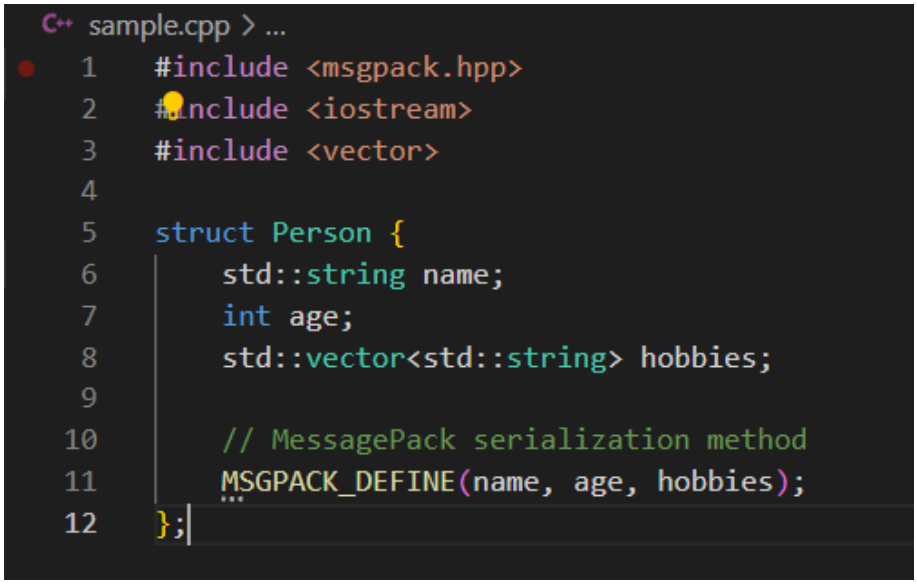
- **Δημιουργία κώδικα**: Το Protobuf χρησιμοποιεί μια δυναμική τεχνική δημιουργίας κώδικα για τη δημιουργία κλάσεων μίας συγκεκριμένης γλώσσας για το serialization και το deserialization των "messages" που το αφορούν. Αυτός ο παραγόμενος κώδικας είναι ιδιαίτερα βελτιστοποιημένος για απόδοση και παρέχει ασφάλεια στους τύπους των δεδομένων.

- **Επεκτασιμότητα**: Με την δυνατότητα επέκτασης των ήδη υπάρχοντών "messages", μπορούν εύκολα να προστεθούν πεδία σε αυτά χωρίς την τροποποίηση του κυρίους schema του αρχικού "message". Αυτό είναι κάτι ιδιαίτερα χρήσιμο σε συστήματα που χρησιμοποιούν πολλαπλούς τύπους δεδομένων ή όταν γίνεται ενσωμάτωση τρίτων πακέτων σε μία εφαρμογή, κάτι που συμβαίνει διαρκώς στον τομέα των βιντεοπαιχνιδιών.

Συνοψίζοντας, το Protobuf προσφέρει έναν γρήγορο, αποτελεσματικό και γλωσσικά ουδέτερο τρόπο serialization δομημένων δεδομένων, με υποστήριξη πολλαπλών γλωσσών προγραμματισμού, αποτελεσματική σειριοποίηση, backwards compatibility και ενσωμάτωση με RPC frameworks (ref). Αυτό τον καθιστά μία πολύ καλή επιλογή για τη δημιουργία κατανεμημένων συστημάτων και πρωτοκόλλων επικοινωνίας, ιδιαίτερα σε εφαρμογές με κρίσιμες επιδόσεις, όπως υπηρεσίες ιστού μεγάλης κλίμακας ή βιντεοπαιχνίδια.

2.5.2 MsgPack C++

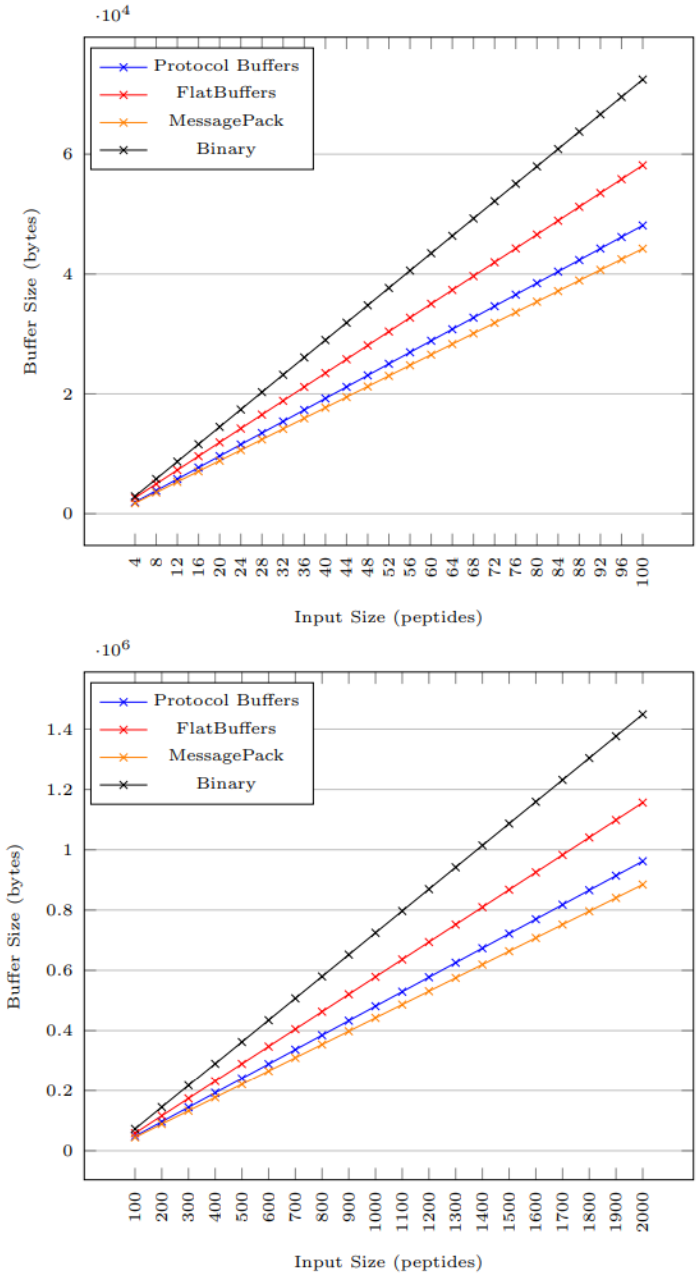
Το MessagePack είναι μία open-source binary μέθοδος serialization σχεδιασμένη με στόχο την αποδοτικότητα. Χρησιμοποιείται συχνά σε πρωτόκολλα επικοινωνίας και αποθήκευσης δεδομένων, όπου το μέγεθος και η ταχύτητα είναι κρίσιμα (ref thesis 2022). Και το MesssagePack χρησιμοποιεί structs ή κλάσεις γραμμένα στην εκάστοτε γλώσσα σαν data containers για το serialization των δεδομένων, όπως αυτό παρουσιάζεται στην Εικόνα 12.



Εικόνα 12: Δείγμα MessagePack struct προς serialization

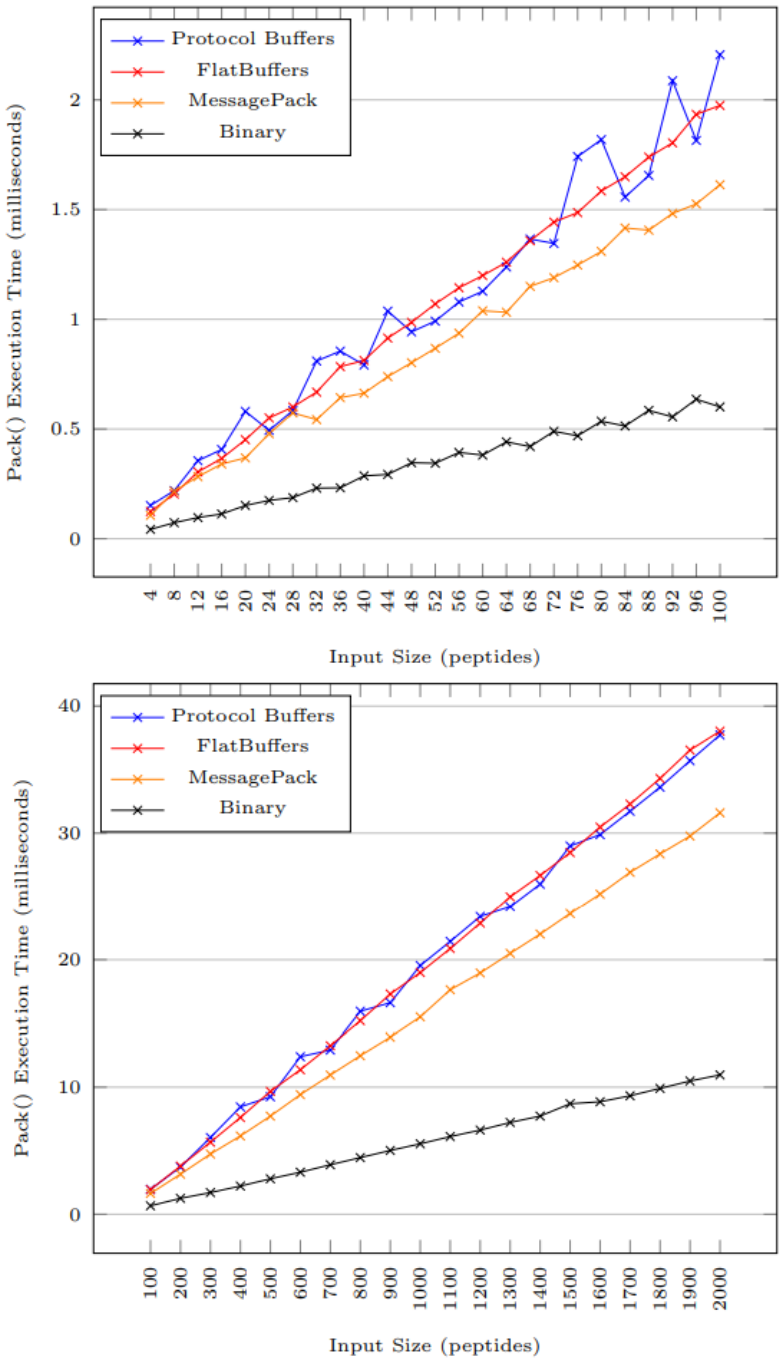
Τα βασικά χαρακτηριστικά που επιλέχθηκε το MessagePack προς ανάλυση είναι τα παρακάτω:

- **Συμπαγής binary μορφή**: Το MessagePack αναπαριστά τα δεδομένα σε δυαδική μορφή, που σύμφωνα με την επίσημη ιστοσελίδα του αλλά και μία έρευνα του University of Houston το 2022, φαίνεται πως είναι πιο συμπαγής σε σύγκριση με τις μορφές κειμένου όπως το JSON, XML και YAML ακόμα και σε σύγκριση με το Protobuf, όπως διακρίνεται στην Εικόνα 13 (ref). Αυτή η συμπαγής μορφή μειώνει το μέγεθος των μεταδιδόμενων δεδομένων, καθιστώντας τα αποδοτικά για επικοινωνία και αποθήκευση στο δίκτυο.



Εικόνα 13: Μεγέθη serialized δεδομένων, με τα μικρά εισαγόμενα μεγέθη (πάνω) και τα μεγάλα (κάτω) (ref from Casey 2022)

- **Αποδοτικό serialization και deserialization**: Με βάση μία έρευνα του University of Houston το 2022 από τον Allen Michael Casey, το MessagePack έχει έναν πολύ αποτελεσματικό αλγόριθμο serialization και deserialization, όπως τα αποτελέσματα του φαίνονται στην Εικόνα 14. Αυτή η αποδοτικότητα είναι ιδιαίτερα επωφελής στη C++ όπου η απόδοση είναι κρίσιμη.



Εικόνα 14:Χρόνος εκτέλεσης της μεθόδους Pack() για όλους τους μηχανισμούς, με μικρότερες εισόδους (πάνω) και μεγαλύτερες εισόδους (κάτω)(ref from Casey 2022)

- **Language agnostic**: Αυτό σημαίνει ότι μπορεί να χρησιμοποιηθεί σε διάφορες γλώσσες προγραμματισμού χωρίς προβλήματα συμβατότητας. Αυτό το καθιστά κατάλληλο για ετερογενή περιβάλλοντα όπου χρησιμοποιούνται πολλές γλώσσες για την ανάπτυξη μίας εφαρμογής.

- **Υποστήριξη τύπων δεδομένων**: Υπάρχει υποστήριξη για integers, floating-point numbers, συμβολοσειρές, πίνακες και maps.

- **Data Streaming**: Το MessagePack υποστηρίζει data streaming, επιτρέποντας στις εφαρμογές να επεξεργάζονται αποτελεσματικά μεγάλα σύνολα δεδομένων χωρίς να φορτώνουν όλα τα δεδομένα στη μνήμη ταυτόχρονα.

- **Open Source**: Είναι open source (ref from site) κάτι που επιτρέπει στον εκάστοτε developer να σμιλέψει όλες τις διαδικασίες του όπως εκείνος επιθυμεί.

Συνοψίζοντας, το MessagePack προσφέρει μια συμπαγή, αποδοτική και γλωσσικά ανεξάρτητη μορφή serialization κατάλληλη για κρίσιμες σε απόδοση εφαρμογές σε C++. Η υποστήριξή του για διάφορους τύπους δεδομένων, το data streaming και η συμβατότητα μεταξύ πλατφορμών το καθιστούν μια ευέλικτη επιλογή για έργα που απαιτούν serialization και μετάδοση δεδομένων υψηλής απόδοσης, όπως για παράδειγμα τα βιντεοπαιχνίδια.

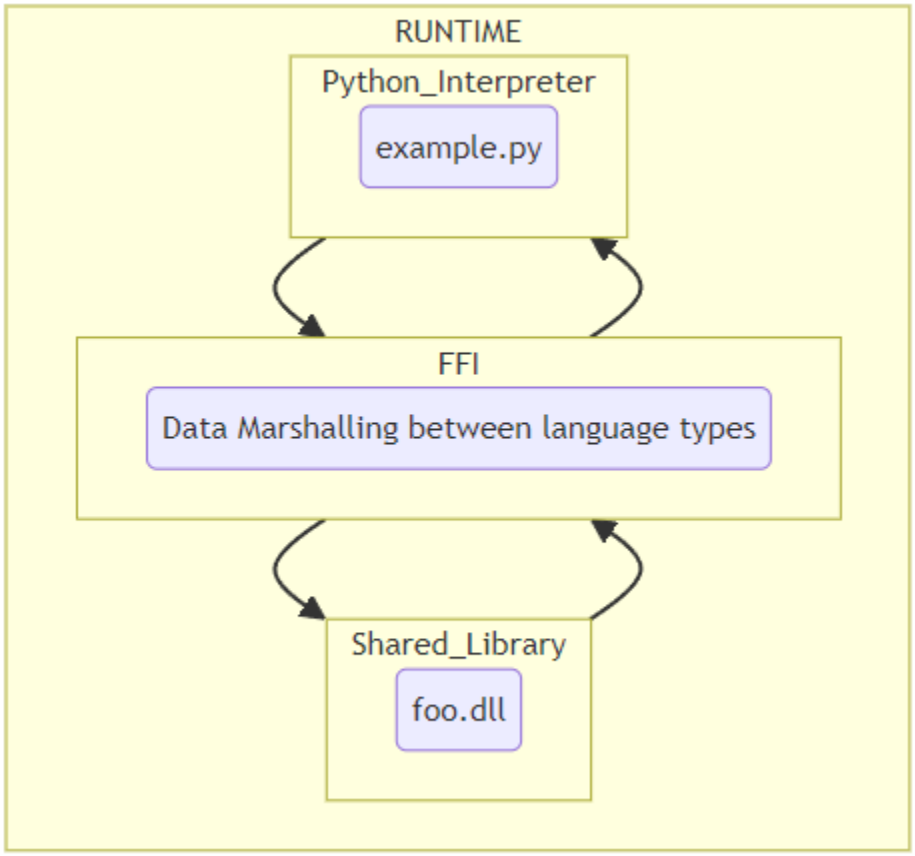
2.6 Foreign Function Interfaces

Στη ενότητα αυτή θα γίνει μία βασική ανάλυση των λειτουργιών των Foreign Function Interfaces (FFI) ξεκινώντας από τον ορισμό τους. Στη συνέχεια επεξηγούνται τρία βασικά σημεία που χρίζουν προσοχής κατά τη δημιουργία και τον σχεδιασμό ενός FFI με αυτά να είναι το λεγόμενο Data Marshalling, η διαχείριση των σφαλμάτων και τέλος η υπολογιστική βαρύτητα που επιφέρει το κάλεσμα μεθόδων μεταξύ διαφορετικών γλωσσών.

2.6.1 Ορισμός του Foreign Function Interface

Τα Foreign Function Interfaces (FFIs) αποτελούν έναν σημαντικό μηχανισμό που διευκολύνει τη δια λειτουργικότητα μεταξύ διαφορετικών γλωσσών προγραμματισμού, επιτρέποντας έτσι την απρόσκοπτη ενσωμάτωση διαφορετικών στοιχείων σε ένα γενικότερο περιβάλλον μίας εφαρμογής.

Στον τομέα ανάπτυξης λογισμικού, τα FFI χρησιμεύουν ως γέφυρες, επιτρέποντας στον κώδικα που είναι γραμμένος σε μια γλώσσα (host language) να έχει πρόσβαση σε συναρτήσεις και δομές δεδομένων που ορίζονται σε μια άλλη γλώσσα (guest language). Αυτή η ικανότητα είναι καθοριστική σε σενάρια που απαιτούν τη χρήση πολλαπλών γλωσσών σε ένα ενιαίο project, όπως η αξιοποίηση των πλεονεκτημάτων απόδοσης των βιβλιοθηκών low-level γλωσσών όπως η C ή C++ από γλώσσες high-level όπως η C#, Python ή η JavaScript (ref). Η διαδικασία αυτή διακρίνεται στην Εικόνα 15. Μέσω των FFI, οι προγραμματιστές μπορούν να ξεπεράσουν τα γλωσσικά εμπόδια, αξιοποιώντας τα πλεονεκτήματα των διαφόρων γλωσσών και εξασφαλίζοντας παράλληλα τη συμβατότητα και τη συνοχή σε ολόκληρη τη βάση του κώδικα. Ωστόσο, αυτή η αρχιτεκτονική λύση φέρει μαζί της και τα ανάλογα προβλήματα και εμπόδια, όπως είναι το data marshalling μεταξύ των δύο γλωσσών που θα πρέπει να διαχειριστεί ο εκάστοτε προγραμματιστής και το performance overhead που υπάρχει λόγω των εξωτερικών μεθόδων που καλούνται αλλά και λόγο του context switching που συμβαίνει εκείνη τη στιγμή στον επεξεργαστή. Τέλος, ένας τομέας των FFI που χρήζει σημαντικής προσοχής είναι το error handling μεταξύ των γλωσσών (ref).

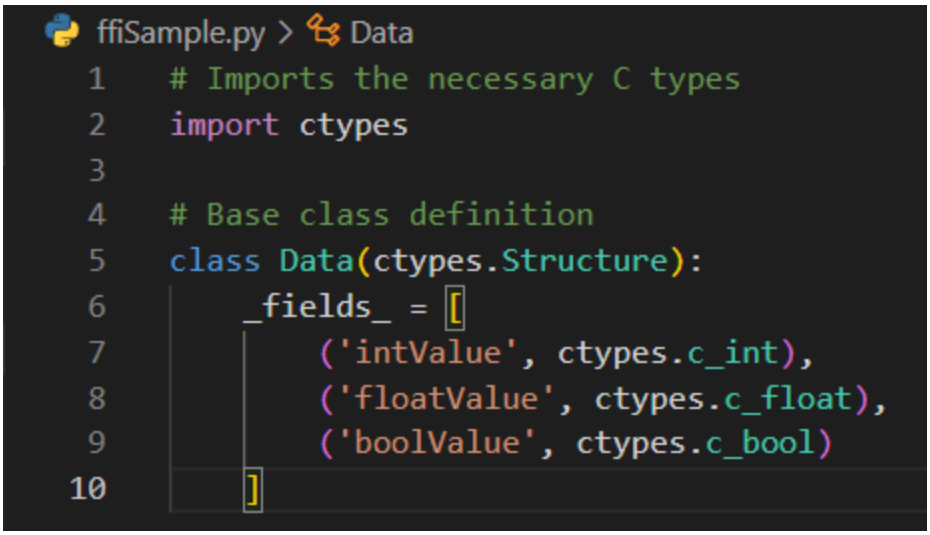


Εικόνα 15: FFI και Data Marshalling

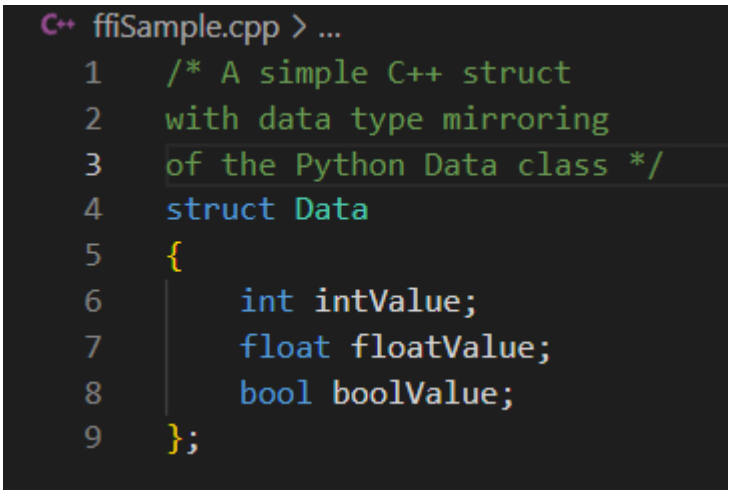
2.6.2 Data Marshalling στα Foreign Function Interfaces

Η «διακίνηση δεδομένων» ή αλλιώς Data Marshalling, επίσης γνωστό ως “packing”, περιλαμβάνει τη μετατροπή δεδομένων από μια αναπαράσταση σε μια άλλη, συνήθως με σκοπό τη μεταφορά τους μεταξύ διαφορετικών συστημάτων ή γλωσσών. Στο πλαίσιο των FFI, το data marshalling χρησιμοποιείται για τη μετατροπή δεδομένων μεταξύ των αναπαραστάσεων που χρησιμοποιούνται από την καλούσα γλώσσα και την καλούμενη γλώσσα, όπως αυτό φαίνεται στην Εικόνα 15. Για παράδειγμα εάν μια συνάρτηση σε μια βιβλιοθήκη C αναμένει μία συμβολοσειρά string και καλείται από την Python, η οποία συνήθως χρησιμοποιεί συμβολοσειρές Unicode, το FFI θα πρέπει να διαμορφώσει τη συμβολοσειρά Unicode της Python σε μια μορφή που μπορεί να κατανοήσει η συνάρτηση C, όπως ένα null-terminated ASCII string.

Μια παρόμοια διαδικασία μπορεί να εννοηθεί και μεταξύ των Εικόνων 16 και 17, όπου ο προγραμματιστής θα έπρεπε να είχε δημιουργήσει και τα δυο data types για να μπορέσει να «μεταφέρει» την πληροφορία και τα δεδομένα από την πλευρά της Python στη βιβλιοθήκη της C++.



Εικόνα 16: Data Container δείγμα σε Python προς μεταφορά σε C++ library



Εικόνα 17: Data Container δείγμα σε C++ έτοιμο να δεχτεί δεδομένα από μία κλάση Python μέσω του FFI

2.6.3 Διαχείριση των Exceptions μεταξύ των γλωσσών

Όταν καλείται μία συνάρτηση σε διαφορετικές γλώσσες, είναι σημαντικό να εξετάζεται ο τρόπος διάδοσης και χειρισμού των σφαλμάτων μεταξύ των διαφορετικών γλωσσών. Αυτό μπορεί να περιλαμβάνει τη μετατροπή των αναπαραστάσεων των σφαλμάτων μεταξύ των γλωσσών ή την παροχή μηχανισμών για τη μετάφραση των σφαλμάτων που ανακύπτουν σε μια γλώσσα σε exceptions ή κωδικούς σφαλμάτων σε μια άλλη γλώσσα (ref). Μερικοί ενδεικτικοί τρόποι σωστής διαχείρισης των σφαλμάτων μεταξύ διαφορετικών γλωσσών είναι οι ακόλουθοι:

- **Error Propagation**: Σε πολλές περιπτώσεις, τα σφάλματα που προκαλούνται από συναρτήσεις που καλούνται μέσω ενός FFI μεταδίδονται πίσω στον κώδικα που καλεί την συνάρτηση. Αυτό σημαίνει ότι εάν μια συνάρτηση που καλείται μέσω του FFI αντιμετωπίσει ένα σφάλμα, όπως ένα exception

χρόνου εκτέλεσης ή έναν κωδικό σφάλματος, μπορεί να κάνει throw ένα exception ή να επιστρέψει μια ένδειξη σφάλματος στον καλούντα κώδικα.

- **Κωδικοί Σφαλμάτων**: Τα FFI μπορούν να χρησιμοποιήσουν κωδικούς σφαλμάτων ή ειδικές τιμές επιστροφής για να υποδεικνύουν σφάλματα. Για παράδειγμα, μια συνάρτηση που καλείται μέσω ενός FFI μπορεί να επιστρέψει μια αρνητική τιμή ή έναν δείκτη NULL για να σηματοδοτήσει μια κατάσταση σφάλματος.

2.6.4 Performance Overhead στα Foreign Function Interfaces

Η χρήση των FFI συνεπάγει μεταξύ άλλων και ζητήματα επιδόσεων, που αφορούν κυρίως τη μετατροπή δεδομένων, την επιβάρυνση κλήσης συναρτήσεων και τη διαχείριση μνήμης. Η μετατροπή δεδομένων μεταξύ διαφορετικών αναπαραστάσεων καθώς διασχίζει τα όρια των γλωσσών μπορεί να εισάγει μία υπολογιστική επιβάρυνση, ιδίως για μεγάλα σύνολα δεδομένων, καθώς συχνά περιλαμβάνει λειτουργίες marshalling και unmarshalling, όπως αυτές εξηγήθηκαν στην ενότητα 2.6.2. Παρομοίως, η πράξη της κλήσης συναρτήσεων μεταξύ διαφορετικών γλωσσών προσθέτει στο φορτίο επεξεργασίας λόγω βημάτων όπως η μεταφόρτωση παραμέτρων και η εναλλαγή περιβάλλοντος μεταξύ γλωσσών, το λεγόμενο context switch (ref). Επιπλέον, οι αποκλίσεις στους μηχανισμούς διαχείρισης μνήμης μεταξύ των γλωσσών μπορεί να επηρεάσουν την απόδοση, γεγονός που απαιτεί προσεκτική εξέταση, ιδίως σε σενάρια που περιλαμβάνουν συχνές λειτουργίες μνήμης όπως ένα library με cache και δομές δεδομένων.