**XML Parser**

**Глава 1. Увод**

Основната идея на проекта е да чете и изпълнява прости операции с**XML** файлове. **XML** файловете са характерни с това, че в тях информацията е маркирана чрез тагове и по този начин се определят обектите и техните атрибути, наследници. На този етап проекта е под формата на конзолно приложение, на принципа на команден интерпретатор.

Необходими и задължителни изисквания при работата с XML файлове:

* Файлът трябва да е в **UTF-8** формат
* Всеки отварящ таг е нужно да бъде затворен
* Връзката между *Parent* и *Child* се осъществява чрез табулация

Зададени изисквания към проекта:

* Всеки елемент трябва да има уникален идентификатор
* Да се поддържат списъци с атрибути и техни стойности за всеки елемент
* Списък с вложени елементи (деца) или текст за всеки елемент
* С предимство са заложени във файла идентификатори

Интерпретаторът трябва да поддържа следните команди и функции:

* Подходящо принтиране на документа, спрямо изгледа характерен за файловия формат – **XML**
* Достъп до атрибут и неговата стойност, чрез зададено име на атрибута и идентификатор на елемента
  + Присвояване на нова стойност по зададена такава, име на атрибута и идентификатор на елемента
* Списък с атрибути на вложените елементи по даден идентификатор
* Достъп до <**n**>-тия наследник на елемент с даден идентификатор
* Достъп до текста на елемент с даден идентификатор
* Изтриване на даден атрибут по ключ и идентификатор
* Добавяне на нов наследник на елемент с празен списък с атрибути освен генериран от програмата идентификатор
* Изпълняване на **XPath** заявки по зададен <**xpath**>и идентификатор на елемент към, който да се изпълни

**Глава 2. Преглед на предметната област**

Основни дефиниции, които ще бъдат използвани са:

* *Deserialization* – процес по превръщане на серия от байтове (низ) в обект.
* *Serialization –* процес по превръщане на обект в серия от байтове.
* *Regex -* регулярни изрази, които намират съвпадения по предварително зададени шаблони и източник под формата на низ

Първият проблем при изпълнението на заданието, е локализирането, отварянето и прочитането на **XML** файла, без да се нарушава неговата цялост и той да остане годен ползване, след приключването на програмата. При подаден невалиден адрес на файла, се създава нов празен файл.

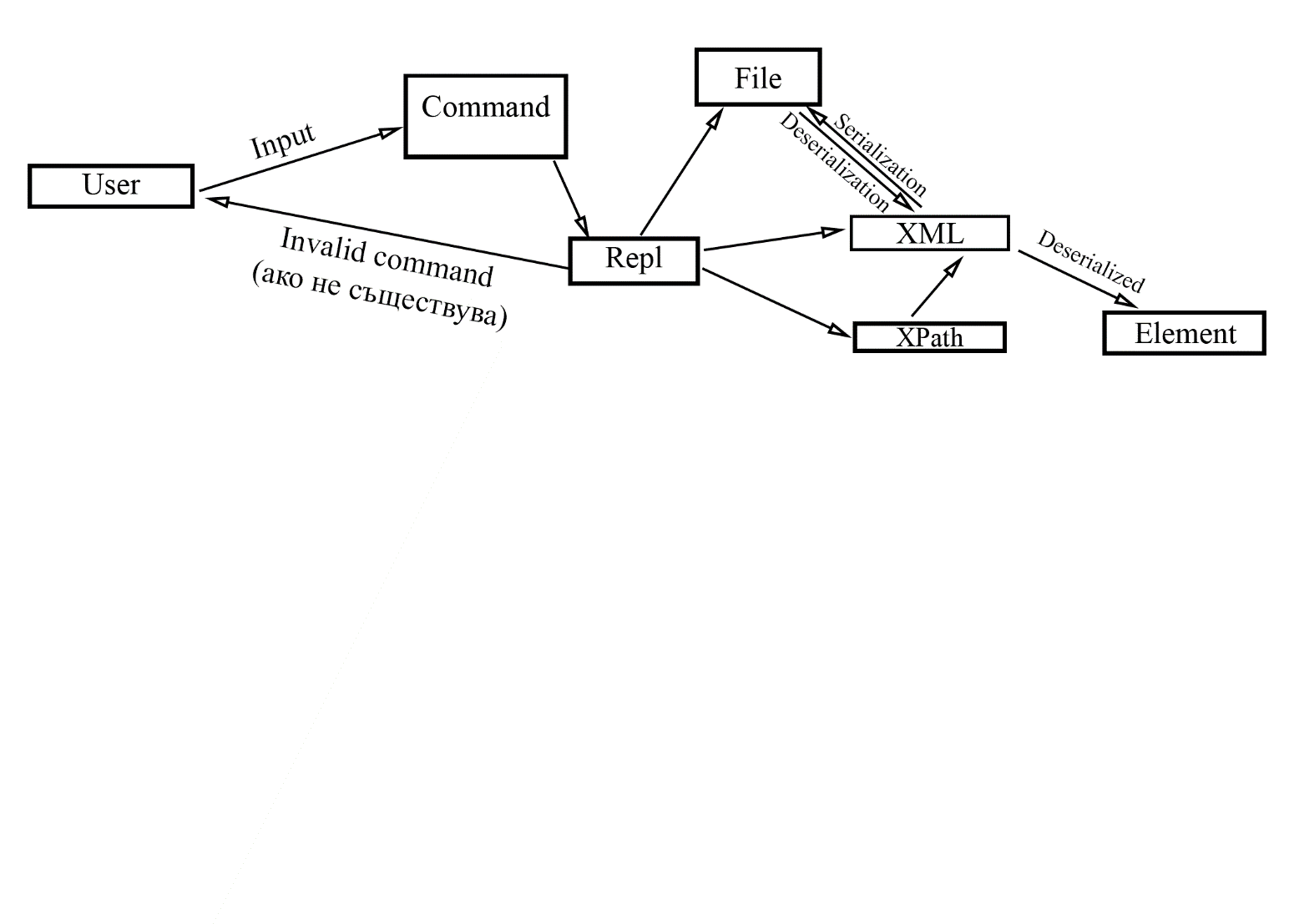
Друг възможен проблем при изпълнението на програмата е, опит за изпълняване на операции преди успешно да бъде отворен файл, и съдържанието да бъде прочетено и *Deserialized*.

Невалидно съдържание или оформление на **XML** файла, също може да доведе до редица проблеми и неточности по време на изпълнението на програмата.

Поради нуждата от запазване и работа с низове, като главен предмет на програмата, ще бъде използвано std::string за всички низове в програмата.

От потребителите се изисква да не добавят нови елементи, към затварящи тагове или към несъществуващи елементи. Също така имената на елементите , да не съдържат специални символи, табулации или разстояния.

**Глава 3. Проектиране**

*Примерна диаграма на последователностите*

Основните класове са:

* XMLParser
* Repl
* File
* XML
  + XPath
* Element

На графиката е представен основния процес и стандартното протичане на програмата. От потребителя се изисква въвеждането на команда, ако тя е валидна се изпълнява, ако не е се принтира подходящо съобщение на екрана и се очаква от него, нова команда.

За изпълнението на програмата е отговорен класът Repl и неговият метод Repl::REPL(). Класът има полета XML::xml и XPath::xpath, които осъществяват връзката между него и съдържанието на отвореният файл, чието съдържание е записано в xml.content, а списъкът с обекти в xml.Deserialized.

XML::XML(const vector<string> content) {

this->content = content;

this->Deserialized ();

}

След отварянето и прочитането на файла, съдържанието му се подава на конструктора на класа **XML** и в него се извиква методът Deserialization(), който има за цел да преведе всеки един ред от съдържанието на файла, в нов обект от тип Element и да го добави към колекцията от елементи.

**Глава 4. Реализация, тестване**

При стартирането на програмата, се създава обект от клас Repl, който съдържа полета xml и xpath, които преди отваряне на валиден файл, са с *default* стойности. След отваряне и прочитане на съдържанието на файла, то се записва в полето xml.content на обектът от тип XML, а методът Deserialized(), подготвя за употреба полето на обектът xml.Deserialized. Този метод е отговорен за извличането на информацията за всеки от елементите, като:

* Наследници
* Родители
* Текст
* Име на елемента
* Идентификатор
* Атрибути (ключ - стойност)

Полето xpath се копира от създаден XPath по подаден списък от елементи, който се взима от полето xml.Deserialized.

xpath = new XPath(xml->get\_Deserialized ());

Ако успешно бъде отворен файл, то променливата bool file\_Opened; променя своето състояние, което не позволява отново да се отвори друг файл, докато не се затвори текущият с командата Close.

Всяка въведена валидна команда с нужните аргументи, съответства на метод от класа XML или XPath. При невалиден идентификатор или друг аргумент, програмата извежда подходящо съобщение и продължава своето изпълнение.

За запазване на промените в текущият файл се използва командата save , а за запазване в нов файл – saveas. За принтиране на всичко поддържани команди и нужните аргументи, е нужно използването на командата help. Бъдещо подобрение би могло да бъде динамично добавяне към колекцията с поддържаните функции.

Важни сценарии, които трябва да бъдат тествани са:

* Празен файл
* Несъществуващ файл
* Добавяне на деца към несъществуващи родители
* Операции върху невалиден идентификатор
* Добавяне на деца към затварящи тагове (този случай се решава, като не се добавя идентификатор на тези елементи, и по този начин няма възможност за добавяне на деца или взимането на техни атрибути или текст)

Xpath person(address="USA")/name. Следниият XPath взима всички имена на хора, които имат адрес равен на "USA". В примера даден в заданието това е само едно име и то е John Smith.

Actual:

> xpath 1 person(address="USA")/name

- John Smith

**Глава 5. Заключение**

Проекта на този етап изпълнява всички поставени цели и функционалности от начален етап. Възможните подобрения за разширяването на функционалността на проекта, са много на брой, но на преден план биха били, подобрения свързани с добавянето на нови xpath заявки, към списъка с поддържаните такива.