Secure Migration of Legacy Applications to the Web

Zisis Karampaglis, Anakreon Mentis, Fotios Rafailidis,
Paschalis Tsolakidis, Apostolos Ampatzoglou

Aristotle University of Thessaloniki, Thessaloniki, Greece
e-mails: anakreon@csd.auth.gr, apamp@csd.auth.gr

Abstract: In information technology, migration is the process of moving from one hardware or software platform to another. Nowadays, many desktop applications tend to migrate to the web or to the cloud. Desktop applications are not prepared to face the hostile execution environment of the web where applications frequently receive harmful data that attempt to exploit programming errors and vulnerabilities such as SQL injection and buffer overflows. We propose a migratory process for text based user interface desktop applications that mitigates the security concerns and enables them to perform safely in the web without modifications of the source code. Additionally, we describe an open source tool that facilitates the migratory process.

Keywords: software migration; web application; user interface; legacy application; safety

1 Introduction

Migration is the process of moving from one operating environment to another that is thought to be a more fitting one for the purpose of an application or a company. For example, migration could mean replacing a Windows-based environment with a Linux-based environment or vice versa. At this point it is necessary to clarify that term migration does not necessarily refers to operating systems or software, but to hardware, as well.

Nowadays however, the growing trends for providing SaaS (Software as a Service) and the globalization of modern economy, demand the transfer of many systems to the web or even to the cloud. Such a transfer, in addition to the obvious implementation cost, introduces security-related risks and costs. Web applications are exposed to a large and distributed user base that by accident or malice, can provide input that is harmful to the application execution and stored data.

In this paper we propose a migration process for *legacy* applications to the web, that protects the application from harmful user provided input. The proposed process uses an open-source tool developed by the authors for automatically adapting the text-based user interfaces to web-based UI.

In section 2, we discuss the current state of the art on how to modernize legacy applications and on data sanitization techniques. In section 3, we describe the proposed migration process, along with a qualitative evaluation. Furthermore, in

section 4 we present the developed tool. Section 5 examines threats to validity while section 6 proposes future extensions and the presented work are concluded in section 7.

2 Related Work

In this section of the paper we provide background information on the two basic concepts of this paper, i.e. modernization of legacy applications and data sanitization. More specifically, in section 2.1 we provide a systematic literature review of studies, which describe processes that are used for migrating legacy applications to more modern environments. Additionally, in section 2.2 we describe basic concepts of data sanitization and related work on tools that perform such tasks.

2.1 Modernization of Legacy Application

Bringing legacy applications at par to the latest technological standards has been the focus of many research efforts. They propose a wide variety of processes and methods to modernize the software components of the legacy applications as well as the hardware they perform on. Modification of the legacy code is an expensive and difficult task which most research efforts try to avoid. Instead, a middleware is introduced that provides the necessary stepping stone in the migration of the application into a modern execution environment.

Software modernization is achieved by enhancing application properties such as availability, usability, security, flexibility, interoperability, expandability and maintainability.

Modern operating systems prevent or limit direct access to system components that raise interoperability issues with some legacy applications. [32] propose using a Virtual Desktop Infrastructure (VMware VDI) that executes the legacy application in its own OS. In [20], a POSIX shell interpreter and an applet allow the execution of an application from different operating systems. In [22] the authors describe two obstacles that legacy applications must surpass to achieve web-integration: platform dependencies and quality requirements that the application should meet. The authors show that both obstacles are efficiently resolved by a middleware product. In [10], the authors provide guidelines to novice developers for migrating desktop applications to handheld devices. As a proof of concept, an existing application is migrated to the Palm operating system. In [18] the authors propose the Enterprise Integration (EI) architecture for legacy applications. The purpose is to increase the reusability of the applications that follow this architecture.

Security issues caused by incorrect sanitization of user-provided input are the focus of [26]. Cross-site scripting (XSS) exploits are prevented without any modifications to the application implementation. In [16] several case studies of test-driven methods are presented to define the flexibility of a legacy application in terms of security, by determining the degree of coupling between business logic and access control. Highly coupled applications are less flexible and require more modifications

in the implementation to enhance security. In [15] the author presents an Interface Adapter Layer to enable communication between a Common Object Request Broker Architecture (CORBA) platform and a legacy application. With this technique, the legacy application maintains its autonomy, enhances its availability and improves its security. In [17] the authors use the Memory Encryption and Transparent Aegis Library (METAL) to enhance the security of a legacy application from memory scanning attacks.

Expandability towards modern technologies was often not top priority for legacy applications. The authors of [2] present a solution to the problem of continuous re-engineering that legacy applications have. With constant maintenance to keep an application up to date with the latest technological standards, its architectural structure becomes more difficult to expand. To mitigate the problem they suggest the use of ArchJava to ensure the integrity of the original architectural structure.

Approaches to increase flexibility appear in [31] where the authors propose a framework to enable Quality of Service (QoS) from a network layer mechanism, for legacy applications. The main constrain of introducing such a framework is the inflexibility to modification and recompilation that comes with legacy code. In order to surpass this obstacle the authors use a middleware component that bridges the gap between the application and the network QoS entities.

The authors of [4] use the Ubiquitous Web Application (UWA) framework to reengineer a legacy application into the web. The benefits of the method are good documentation and high usability/maintainability for the reengineered application as opposed to a production-oriented approach based on studies of past years. Another approach to usability comes from [11] where the authors suggest two proof-of-concept approaches to change the user interface of legacy applications into an innovative one. They separated the graphic output subsystem and the user interface from the application and modified them to support modern ways of interaction without changing the core of the application. In the end they claim that a full implementation of such an approach would be hard to exist within the scope of a research study. In [Paiano2011] the authors propose the use or Rich Internet Applications (RIA) to modernize legacy applications in terms of interaction and usability. They use the Rich-IDM method in a large-scale application and showed that it can extend the application into other conceptual tools.

In [27] the authors describe that the implicit information inside the legacy application is the first step to successful reengineering. The authors of [27] present a case study where by using SNAP, a CASE tool, they extracted the persistent data model and call-tree of an application and used them in the reengineering process, increasing the documentation and maintainability of the application. A common approach showing that understanding the application design is the first step towards modernizing the legacy application is introduced in [5]. They used Ubiquitous Web Applications Design Framework (UWA) and an extended version with Transaction Design Model (UWAT+). Through this method they leveraged legacy applications into rich Internet ones and due to the formality of the framework they minimized the time needed for the transition.

An upward trend exists that supports the transition of legacy applications towards a Service Oriented Architecture (SOA). Legacy applications are usually transformed into Grid services or services of cloud computing. In [Erradi2006] the authors in order

to achieve low integration cost, increased reusability and adaptability, present a survey of key approaches to modernize legacy applications together into SOA with a framework that will aid in deciding the optimal approach. In [1] the authors propose three approaches to transition legacy applications to Web Services through Service Oriented Architecture (SOA). The reason for this transition is the increased need for new features in organizations legacy IT infrastructure. The three approaches are: Session based approach that change only the Graphical User Interface (GUI) keeping the code of the legacy application unmodified, Transaction based approach to cover security issues that comes with the transition and Data based approach that expose the data of the application to the Web. Their work will be implemented, tested and verified in the future. In [28] the authors present an approach to migrate legacy applications into SOA by separating the business and the logic view of an application in order to redesign the legacy code.

In [34] the authors propose a method for migrating legacy applications into Grid Services. The otherwise standalone applications are reused and wrapped into the Grid technology through the use of Globus Toolkit. The main benefit of this approach is that without modifying the code of the application for Internet functionality, a multi-user access can be achieved. Similar to [34] the authors of [14], [19] and [9] present their techniques for migrating legacy code into Grid services without changing the source code of the application. In [12] the authors describe that the transition of an application to the Grid is made only for specific applications. In order for legacy application to effectively became part of the Grid they present three steps. Those steps are application partitioning, enablement and deployment. The authors also make a case study by using those steps to minimize manual alteration to the legacy application.

In [33] the authors provide a solution to migrate a legacy application to cloud. The Application Migration Solution (AMS) reconstructs the GUI of the application without changing its source code. The solution also supports the compilation of more than one application together in order to create a more powerful, in terms of features, application. In [6] the authors developed a prototype Cloud Customer Relationship Management (CloudCRM) from a legacy application, based on a three part architecture: a Web portal, a Software as a Service (SaaS) supermarket and a SaaS application development platform. The reason is the increase attention cloud computing attracts. In [13] the authors present a case study to increase the availability of legacy applications. They use a middleware (OpenSAF) to enhance a video streaming application. They suggest that in cloud computing, tradeoffs should be made between availability and resource cost when deciding the configurations of the legacy application. In [8] the authors present a model for automatic self-configuration protocol for legacy cloud applications. The decentralized protocol is used to improve the efficiency of the cloud deployment process.

2.2 Data Sanitization

Sanitization takes user input and transforms it in order to eliminate potential threats for the data and operation of the application. The threats are caused by the use of characters known as *metacharacters* that have special meaning, when processed by

the various parts of a system. The transformation of user input is applied by removing characters based on two methods. These methods are distinguished by using a “white” or “black” list of characters, which contain characters that can or cannot be harmful for the application respectively.

For example, an SQL Injection attack consists of insertion or "injection" of a SQL query via the input data from the client to the application using meta-characters or special characters of SQL, such as “’”, “””, “AND”, “=”. An SQL injection attack can read sensitive data, modify data, and execute administration operations. The following code example taken from [25] uses a *Prepared Statement*. Consider the string *john* is the value for input parameter *user_name*, which is a valid input. The application will search a user with the *user_name john* and return all information for this user. On the opposite, if someone passes the string *1’ OR ‘1’=’1*, this leads to an attack because the query that the program submits to the database contains a tautology in the WHERE clause and gain access to sensitive information regarding database users.

```
String firstname = req.getParameter("firstname");
String query = "SELECT * FROM authors WHERE firstname = ? ";
PreparedStatement pstmt = connection.prepareStatement( query );
pstmt.setString( 1, firstname);
ResultSet results = pstmt.execute( );
```

Figure 1: Sample Code for SQL Injection

To prevent SQL Injection Attack, we apply data sanitization. The quotes “’”, “”" and “=*”* are removed from the string and the sanitized string will be *1OR11*, which cannot be a tautology. Several characters can be meta-characters depending on the language of the application. Data sanitization aims to prevent harmful input data to be inserted in an application. In [23] propose a technique of automatic query sanitization to prevent SQL Injection attacks. They use a combination of static analysis and program transformation to automatically instrument web applications with sanitization code.

There are three different ways for applying data sanitization to the user data. The first way is to use tools, which take user data, sanitize them and produce as output the sanitized user data to be sent to the web application. *Urlrewritefilter is based on* is a tool that rewrite *url* using certain filters and send modified url to the web application. Another tool for sanitizing html tags, attributes and values is *jsoop*. *Jetscripts* Data Sanitizer and XSS cleaner prevents SQL Injection and XSS attacks by cleaning or sanitizing user-submitted data. This tool is intended for users who write or modify scripts, or want an extra measure of protection against malicious users. It requires some knowledge in *php* scripting. The sanitizer can work in various modes such as numeric only, alphabetic only, alphanumeric only, alphanumeric with punctuation and email validation mode. Additionally to the above modes common command entities and *Javascript* specific entities are removed.

The second way is to use libraries that offer functions which sanitize input data. The ESAPI library by OWASP provides libraries for many programming languages such as Java, .net, ASP, PHP, PHP, ColdFusion, Python, JavaScript, Objective-C, Ruby, C, CPP and Perl.

The third way is to use embedded functions in various languages that sanitize input data depending on special character for each language. At php, function *mysql_real_escape_string()* sanitizes special character of mysql at a string, which is

proposed to be sent to the mysql database. Other functions are *filter_input()*, *escapeshellarg()* and *urlencode* for php, etc. Finally, tools that are used to statically analyze the vulnerabilities of code written in C are presented and compared in [3].

3 Migration Process

The main goals of the migration process are (a) transferring the application to the web without rewriting code and (b) secure the core of the application from harmful input that might be passed through the web interface.
In order to achieve these goals we propose a two-step process. First, the data are passed through a data sanitizer before reaching the core of the application and the user interaction of the application is automatically translated to a web-based interface. The architecture of legacy systems is depicted in Figure 2, whereas the architecture of the system after applying the proposed process is depicted in Figure 3.

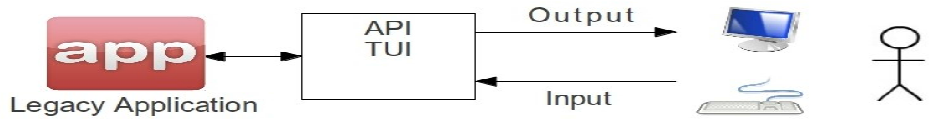

Figure 2: Legacy Application Architecture

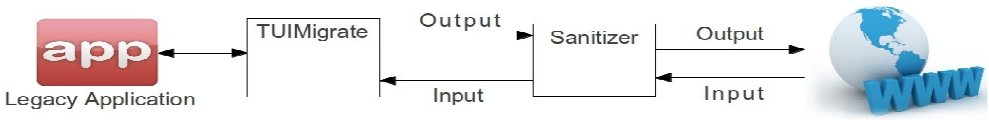

Figure 3: Application Architecture After Proposed Process Application

In typical web architecture, the data are handled through the HTTP protocol and move from client to server through POST variables. The data are processed in the server side and as a result an HTML output is passed to the Client. The HTML document is then handled / translated by the browsers. On the other hand, Legacy Applications retrieve data, process them and provide the output to the users. In the case of our study we focus on applications that interact with the user through a textual user interface. After the application of the process the web architecture is modified as follows. First, he API TUI is replaced with TUIMigrate which is automatically generated by a tool. TUIMigrate is responsible for translating any given user input to an API call. The created output is given from the application to the TUIMigrate Component that delegates it to the user. The other addition is the Sanitizer Component. Sanitizer is responsible to maintain the system secure. Sanitizer receives the data from the web and sanitize them so as to keep possible harmful from the core of the application.

The proposed process clearly addresses both migration goals. In order to validate the usefulness of our approach, we have conducted an interview with the project

manager of a CRM created in the early 80s for the DOS operating system. After three years of development, the CRM was migrated to the Windows OS. Currently, they are considering migrating the application to the web which they estimate it will require an additional five year period. The duration of migrating from Windows to the Web is estimated to be as long as the transition from DOS to Windows, due to the security threats in web applications.

The above case suggests that an automated process for a secure migration of legacy applications to the web would be useful for the software industry.

4. Tool Demonstration

This section of the paper describes the open-source tool that has been developed as part of the proposed process in order to automatically transform text-based user interfaces to web-based user interfaces. In our tool we use the Turbo Vision port to Linux and FreeBSD [19].

Turbo Vision (TV) is a console-based character-mode text user interface (TUI) framework developed by Borland. It enables a reach user interface with various components such as check boxes, radio buttons, and others. The library was placed in the public domain and is currently available for many operating systems. Figure 4 shows a TUI application.

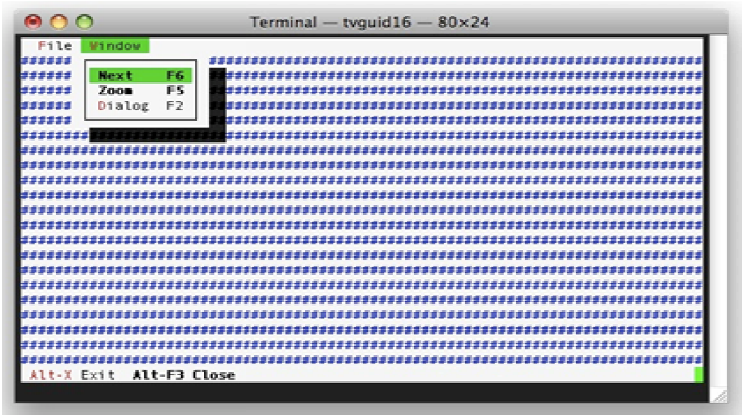

Figure 4: An example TUI application

4.1 Tool Overview

The general purpose of this tool is to enable C++ legacy applications that use Turbo Vision library to be used over the web, without rewriting or altering the application code. In order to achieve this, we have created a new library by rewriting some parts of the above mentioned TVision library. More specifically, we have modified some already defined classes and added some others, without changing the interface provided by TVision. Thus, applications that use the library can be recompiled with our new library without having any problems.

Our library, instead of mouse and keyboard, accepts input commands from a simple flat text file and produce the output to an xml file, instead of the terminal. In this way it can communicate with a web-application that will transform the xml file to html, resulting in the creation of a new Web User Interface. Currently our library is in a beta version and is available for UNIX distributions on sourceforge.net.

4.2 System Architecture

The architecture of the proposed tool consists of three main components. The first component is one modification of the TVision library which as an xml file output and a simple file input. The second component is a sanitizer and the third component is a web-application that will use these files in order to produce the Web User Interface. The proposed library has the same interface with the Turbo Vision library. So legacy applications that use Turbo Vision can use our library instead, without any problems. Unfortunately Turbo Vision is a static library and the legacy applications that uses it, must be re-compiled in order to work with our library.

The Text User Interface consists of several components of two types, namely visible and hidden. We are interested in visible components that can be drawn in the console. TVision library is written in C++, so each component is a different class. The modifications we made were to change some methods of the already defined classes / components, add some new methods and add a new class. There is a general class that is the base of all visible objects, named TView. The TView class exists to provide basic data and functionality for its derived classes. We added three empty methods that are derived in final classes and helped us to produce the output xml file and use the input file.

TGroup is a class that has for main purpose to group objects together in order to allow dynamically linked list of relevant views (including other groups) used as there were a single object. We used TGroup to recursively call some of the methods we created, in TView class, for every object that is included in the linked list. TWindow is a class that implements a window and it contains all the elements needed for the functionality of the window. Also one of its derivatives is the TDialog class, which implements the dialogs needed to interact with the user. Some components shown in table 1 can be used in TDialog. All the components in the table inherit the new methods we created in TView class.

Component	Description
TButton	Normal button
TCheckBoxes	Cluster of checkboxes
TFileDialog	File selector
TInputLine	Input text area for one line
TLabel	Tag for other components
TMemo	Input text area for multiple lines
TMenuView	Abstract base class for menus
TRadioButtons	Cluster of radio buttons

Table 1: Some TVision components

The TVision library uses some predefined events in order to make the components cooperate with each other. Also it uses these events to accept input from keyboard and mouse, as well as to produce the output to the console. Some of the modifications we made are to define new events and their handling. These events helped us to extend the input and the output capabilities of the library, beyond keyboard, mouse and terminal. When the web-application sends a command to the legacy application, a defined event is created by the application and is handled automatically.

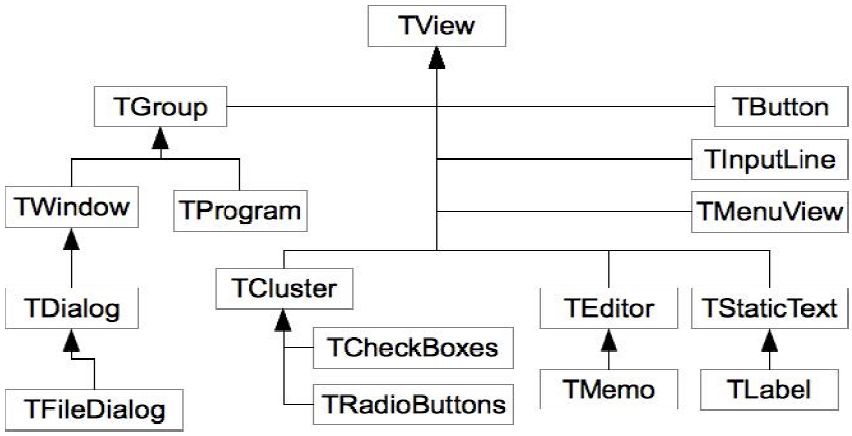

Figure 5: TVision classes and their hierarchy

4.3 Input / Output

The new class we created gave our library the ability to communicate through a specific UDP port with a client application and accepts two commands. One command produces an xml file which describes the currently accessible options from the legacy application e.g. forms, menus, buttons etc., and the other command makes the legacy application to accept a simple flat text file as input.

In this way when a user in the web-application submits something, the web-application creates the appropriate input file and then sends the command needed to our library. Our library accepts the file, does the actions indicated by the file and after that sends back the output xml file with the current state of the application.

The xml file that is being produced by the library has a specific schema so that can be easily read by a web-application that will transform it to html, in order to create a Web User Interface. The xml file is composed of three main parts, each one for different kind of components:

- *Menu part*. This part describes the menus and submenus of the legacy application. Each menu item has a name and an id, so that can be called whenever the end user wants.
- *Window parts*. This part describes each window that is open in the Text User Interface of the legacy Application. Every window has a title and a list of the "useful" components that is made of. The menu part and the window parts can

co-exist in the same file, since they can be both accessible by the user at the same time.

- *Dialog part.* This part exists only if there is a dialog in the front of the Text User Interface. It describes all the components that the dialog contains and is necessary for the end user. All the components, including the window components, have an id, a tag that defines the type of the component, their content and, if exists, their name. If a dialog is in the front of the Text User interface, is the only thing that is accessible by the user. Even if there are menus and windows in the background they are not accessible, so the menu part and the window part cannot co-exist with the dialog part in the same xml file.

```
<xs:element name="dialog">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="label"/>
            <xs:element ref="text"/>
            <xs:element ref="button"/>
            <xs:element ref="checkbox"/>
            <xs:element ref="radiobutton"/>
        </xs:choice>
    </xs:complexType>
    <xs:key name="formItemKey">
        <xs:selector xpath="*"/>
        <xs:field xpath="@id"/>
    </xs:key>
    <xs:keyref name="formItemRef" refer="formItemKey">
        <xs:selector xpath="label"/>
        <xs:field xpath="@idref"/>
    </xs:keyref>
</xs:element>

<xs:element name="checkbox">
    <xs:complexType>
        <xs:sequence maxOccurs="unbounded">
            <xs:element name="item">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:string">
                            <xs:attribute name="selected" type="xs:boolean"/>
                            <xs:attribute name="id" type="xs:int" use="required"/>
                        </xs:extension>
                    </xs:simpleContent>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
        <xs:attribute ref="id" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="label">
```

Figure 6: A small part of the output XML schema

The input file that the library accepts for input is a simple text file. The data in the input file have flat structure. This means that every line contains one action for one component that has to be done by the legacy application, no matter in what order the actions are. Depending on the type of the component, there are up to four different commands:

- name :: value
- id <: value

- name :> index
- id <> index

All the above commands do the same action, but with different arguments. They indicate that the component with *id* or *name* (if the component has name) has to take the specified *value* (e.g. input line) or if the component has options to take the specified option with name *value* or id *index* (e.g. checkboxes).

Every component has its own different arguments that must be passed as values or index. For example, in order to distinguish when a menu item is called, before the id we put an *m* and as value we put the word *active*.

Next we present an example in order to demonstrate the use of our tool. Firstly in Figure 8, we present the XML code that corresponds to the TUI of Figure 4. Additionally, concerning a more elaborate menu, Figure 7 presents a TUI that is used as an example, whereas in Figure 9 we present the equivalent XML file that has been created with our tool.

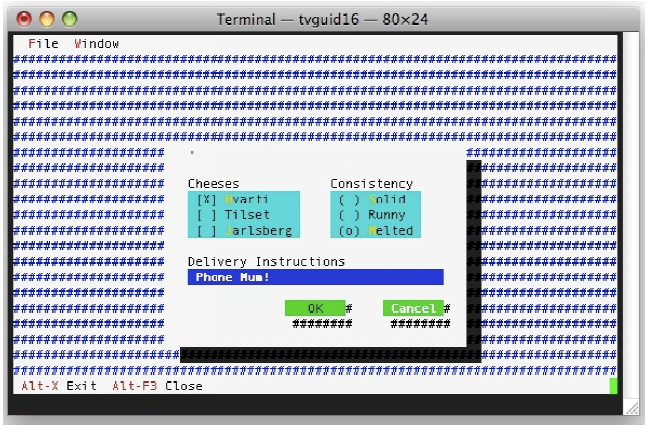

Figure 7: An example of a modal dialog

```
<?xml version="1.0" encoding="UTF-8"?>
<program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <menu>
                <item name = "File" />
                <menu>
                        <item name = "Open" id = "200"/>
                        <item name = "New" id = "201"/>
                        <item name = "Exit" id = "1"/>
                </menu>
                <item name = "Window" />
                <menu>
                        <item name = "Next" disabled = "true" id = "7"/>
```

```
                    <item name = "Zoom" disabled = "true" id = "5"/>
                    <item name = "Dialog" id = "202"/>
                </menu>
        </menu>
</program>
```

Figure 8: XML produced from TUI of Figure 4

```
<?xml version="1.0" encoding="UTF-8"?>
<program xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <dialog>
                <button id="1">Cancel</button>
                <button id="2">OK</button>
                <label id="3" idref="4">Delivery Instructions</label>
                <text id="4"><![CDATA[Phone Mum!]]></text>
                <label id="5" idref="6">Consistency</label>
                <radiobutton id="6">
                        <item selected="false" id="0">Solid</item>
                        <item selected="false" id="1">Runny</item>
                        <item selected="true" id="2">Melted</item>
                </radiobutton>
                <label id="7" idref="8">Cheeses</label>
                <checkbox id="8">
                        <item selected="true" id="0">Hvarti</item>
                        <item selected="false" id="1">Tilset</item>
                        <item selected="false" id="2">Jarlsberg</item>
                </checkbox>
        </dialog>
</program>
```

Figure 9: XML produced from TUI of Figure 8

5. Future Work

At the current preliminary stage of development, we have provided an alternative implementation of a TUI library API that enables legacy applications to operate in a Web environment. We are currently extending the sanitization component with functionality offered by the OSAP security library which detects and blocks user input

that cause SQL injection attacks. When all components of the architecture of the proposed migration approach are completed, we will empirically validate whether TUI application users are equally satisfied by the automatically generated web-based UI.

6. Conclusions

This work proposes a process for automatically migrating legacy applications to the web. It is a general method in the sense that it places no constraints on the design and implementation of a legacy TUI application. As far as the application is concerned, nothing is changed. It still invokes the same methods provided by the TUI API although the user input is now provided by data sent remotely with HTTP requests and the produced output is now transformed into HTML instead of being displayed on the user's screen. Also, the received data is cleared of special characters that may threaten the integrity of the application data or the security of the application. Common cases of errors such as SQL injection attacks are handled by the data sanitizer. In order to demonstrate the feasibility of the approach, we have implemented the API of a TUI library and a tool that automatically transforms TUIs (textual user interfaces) to HTML based UIs. The proposed process, reduces the migration cost and enables secure access to previously locally executed applications in the demanding web environment.

References

1. Al Belushi, W.; Baghdadi, Y. : An Approach to Wrap Legacy Applications into Web Services. In: Service Systems and Service Management, 2007 International Conference, p.p. 1 – 6, 2007.
2. Abi-Antoun M.; Coelho W.: A Case Study in Incremental Architecture-Based Re-engineering of a Legacy Application. In: Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference, p.p. 159 - 168, 2005.
3. Chatzieleftheriou, G.; Katsaros, P. Test driving static analysis tools in search of C code vulnerabilities, In Proceedings of the 35th IEEE Computer Software and Applications Conference Workshops (COMPSACW), Munich, Germany, IEEE Computer Society, 96-103, 2011
4. Distante, D.; Perrone, V.; Bochicchio, M.A.: Migrating to the Web legacy application: the Sinfor project. In: Web Site Evolution, 2002. Proceedings. Fourth International Workshop, p.p. 85 – 88, 2002.
5. Distante, D.; Tilley, S.; Canfora, G.: Towards a holistic approach to redesigning legacy applications for the Web with UWAT+. In: Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference, p.p. 5-10, 2006.
6. Dunhui Yu; Jian Wang; Bo Hu; Jianxiao Liu; Xiuwei Zhang; Keqing He; Liang-Jie Zhang: A Practical Architecture of Cloudification of Legacy Applications. In: Services (SERVICES), 2011 IEEE World Congress, p.p. 17-24, 2011.
7. Erradi, A.; Anand, S.; Kulkarni, N.: Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture. In: Services Computing, 2006. SCC '06. IEEE International Conference, p.p. 257 – 260, 2006.

8. Etchevers, X.; Coupaye, T.; Boyer, F.; de Palma, N.; Salaun, G.: Automated Configuration of Legacy Applications in the Cloud. In: Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference, p.p. 170 – 177, 2011.
9. Fakai Lu; Hao Huang; Zhuoqun Xu; Huashan Yu: A Middleware for legacy application wrapper. In: Semantics, Knowledge and Grid, 2005. SKG '05. First International Conference, p.p. 47, 2005.
10. Foss, A.; Wong, K.: On migrating a legacy application to the palm platform. In: Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop, p.p. 231 – 235, 2004.
11. Guillaume Besacier, Frédéric Vernier: Toward user interface virtualization: legacy applications and innovative interaction systems. In: EICS '09: Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems, p.p. 57-166, New York, 2009.
12. Iyer, P.; Nagargadde, A.; Gopalan, S.; Sridhar, V.: Two phase approach for grid enablement of legacy applications. In: Granular Computing, IEEE International Conference, p.p. 194-199, 2006.
13. Kanso, A.; Khendek, F.; Mishra, A.; Toeroe, M.: Integrating Legacy Applications for High Availability: A Case Study. In: High-Assurance Systems Engineering (HASE), 2011 IEEE 13th International Symposium, p.p. 83 – 90, 2011.
14. Kacsuk, P.; Goyeneche, A.; Delaitre, T.; Kiss, T.; Farkas, Z.; Boczko, T.: High-Level Grid Application Environment to Use Legacy Codes as OGSA Grid Services. In: GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, p.p. 428-435, Washington, 2004.
15. Konstantas, D.: Migration of legacy applications to a CORBA platform: a case study. In: Distributed Platforms: Client/Server and Beyond: DCE, CORBA, ODsanitP and Advanced Distributed Applications, Proceedings of the IFIP/IEEE International Conference, p.p. 100-112, 1996.
16. Le Traon, Y.; Mouelhi, T.; Pretschner, A.; Baudry, B.:Test-Driven Assessment of Access Control in Legacy Applications. In: Software Testing, Verification, and Validation, 2008 1st International Conference, p.p. 238 – 247, 2008.
17. Levy, J.; Khan, B.: Hiding Your Wares: Transparently Retrofitting Memory Confidentiality into Legacy Applications. In: Communications, 2007. ICC '07. IEEE International Conference, p.p. 1368 – 1372, 2007.
18. Li, S.; Yang, H.; Zhou, H.: Building a dependable Enterprise Service Assembly Line (ESAL) for legacy application integration. In: Cyberworlds, 2004 International Conference, p.p. 170 – 175, 2004.
19. Liping Zhu; Matsunaga, A.; Sanjeepan, V.; Lam, H.; Fortes, J.A.B.: Application modeling and representation for automatic grid-enabling of legacy applications. In: e-Science and Grid Computing, First International Conference, p.p. 8-31, 2005.
20. Marosi, A.C.; Balaton, Z.; Kacsuk, P.; GenWrapper: A generic wrapper for running legacy applications on desktop grids. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium, p.p. 1 – 6, 2009.
21. McLean, G.: Validation, Pro WPF and Silverlight MVVM, p.p. 129-144, 2011
22. Mondal Sakib Abdul, Kingshuk Das Gupta: Choosing a middleware for web-integration of a legacy application. In: SIGSOFT Software Engineering Notes , Volume 25 Issue 3, p.p. 50 – 53, New York, 2000.
23. Mui, R.; Frankl, P.: Preventing SQL Injection through Automatic Query Sanitization with ASSIST, Fourth International Workshop on Testing, Analysis and Verification of Web Software, p.p. 27-38 EPTCS 35, Antwerp (2010).
24. Owasp, https://www.owasp.org/

25. Paiano, R.; Pandurino, A.; Mainetti, L.: Two phase approach for grid enablement of legacy applications. In: Web Systems Evolution (WSE), 13th IEEE International Symposium, p.p. 81-85, 2011.
26. Saxena Prateek, Molnar David, Livshits Benjamin: SCRIPTGARD: automatic context-sensitive sanitization for large-scale legacy web applications. In: CCS '11: Proceedings of the 18th ACM conference on Computer and communications security, p.p. 601-614, New York, 2011.
27. Sarmiento, C.; Takahashi, S.: Understanding CASE generated legacy applications: a case study. In: Program Comprehension, Proceedings. 12th IEEE International Workshop, p.p. 111-119, 2004.
28. Sheikh, M.A.A.; Aboalsamh, H.A.; Albarrak, A.: Migration of legacy applications and services to Service-Oriented Architecture (SOA). In: Current Trends in Information Technology (CTIT), 2011 International Conference and Workshop, p.p. 137 – 142, 2011.
29. Sigala Turbo Vision, http://www.sigala.it/sergio/tvision/index.html
30. C. Snyder, T. Myer, M. Southwell, Validating and sanitizing user input, Pro PHP Security, Part 2, p.p. 15-32, 2010
31. Tsetsekas, C.; Maniatis, S.; Venieris, I. S.: Supporting QoS for Legacy Applications. In: Lecture Notes in Computer Science, Volume 2094, Networking — ICN 2001, p.p. 108-116, 2001.
32. Wong, D.: Kickin' it old school!: dealing with legacy applications. In: SIGUCCS '08: Proceedings of the 36th annual ACM SIGUCCS fall conference: moving mountains, blazing trails, p.p. 55-58, New York, 2008.
33. Xin Meng; Jingwei Shi; Xiaowei Liu; Huifeng Liu; Lian Wang: Legacy Application Migration to Cloud. In: Cloud Computing (CLOUD), 2011 IEEE International Conference, p.p. 750 – 751, 2011.
34. Xiong Yu; Daizhong Su: Wrapping Legacy Applications into Grid Services: A Case Study of a Three Services Approach. In: Lecture Notes in Computer Science, Volume 4402, Computer Supported Cooperative Work in Design III, p.p. 520-529, 2007.