Appasami, G.; Suresh Joseph, K.; Annadurai, P.

Online Test Automation for new Generation of Silverlight Web Applications


Available in: http://www.redalyc.org/articulo.oa?id=265219625005
Online Test Automation for new Generation of Silverlight Web Applications

Appasami. G.¹, Suresh Joseph. K.², Annadurai. P.³

¹ Assistant Professor, Dept of CSE, Dr. Pauls Engineering College, Anna University of Technology - Chennai, Villupuram, Tamilnadu, India
appas_9g@yahoo.com

² Associate Professor, Department of computer science, Pondicherry University, Pondicherry, India
sureshjosephk@yahoo.co.in

³ Associate Professor, Department of computer science, Kanchi Mamunivar Centre for Post Graduate Studies, Pondicherry, India
annadurai_aps70@yahoo.co.in

Abstract. New Interactive, attractive and device independent web application’s Graphical User Interfaces (GUI) are developed by new technologies like Silverlight and Moonlight. Silverlight is Microsoft’s cross platform runtime and development technology for running Web based multimedia applications in windows platform. Moonlight is an open-source implementation of the Silverlight development platform for Linux and other operating systems. Manufacturing Execution systems (MES) is a framework which tries to address the problems in integration, deployment, Consistent GUI, etc. that exists in present manufacturing applications. Next generation Visualization technologies for the Manufacturing Execution systems will improve consistency, productivity and usability of the systems. Silverlight is one of the main UI technologies that are used in MES Framework. Testing of UI is always challenging and has been mostly manual till now. In the next generation of MES applications the testing of GUI needs to be automated as it will save lot of man-hours that is lost in manual testing also it will catch the defects early on in the cycle. GUI testing is a commonly known form of Usability Testing of software or website. It is also known as User Interface Testing. GUI testing is a performance related assessment of a software or website in terms of ease of use, versatility, friendliness with focus on the target audience, visual impact and the approach and time taken to progress into the specific purpose. Improving the quality of software is the main goal of UI Test Automation. Reducing testing time and cost are the goals of UI Test Automation. Reducing manual power is also one of the major objectives of UI Test Automation.

Keywords: Graphical User Interface, Test Automation, Online Web Applications Test Automation.

1 Introduction

Software testing plays a vital role in producing good quality best Software. Innovations in the field of software testing have improved the techniques of writing test scripts, generating test cases, and introducing test automation for unit testing, white box testing, black box testing, and other types of testing. Software testing has evolved over the years to incorporate tools for test automation. Mercury Interactive, Rational Software of IBM, and Segue are a few of the main tool vendors. The purpose of these tools is to speed up the software development life cycle, to find as many bugs as possible before releasing the products, to reduce the cost of software development by automating effective regression testing, and to increase application reliability.

The apparent goal of the software testing tools is to generate test scripts that simulate users operating application. Usually test engineers are trained by the tool manufacturers to write test scripts by hand or to use a capture/playback procedure to record test
Software verification and validation through testing is a major building block of the quality assurance process. GUI tests are vital because they are performed from the view of the end users of the application. Because the functionality of the application is invoked through the graphical user interface, GUI tests can cover the entire application when compared with non-GUI software testing, a GUI test infrastructure based on the current testing tools is also expensive with regard to purchasing the tools and the efforts that must be made to maintain the tests. Very often, only a part of the GUI components are tested automatically. The other part still requires manual tests. A GUI component can be recognized by its name, its position in hierarchy, its component class, the title of its parent window, and the programmer-assigned tag or ID. During each session of running an application, the GUI components are located with unique pairs of coordinates on the screen. However, GUI positions are often under modification by programmers, platforms, and screen resolution.

For many years, manufacturers have sought transparency into production information, processes, and resources. Since the first manufacturing resource planning (MRP) systems were introduced three decades ago, many enterprises have considered the factory floor a “black hole.” Very simply, work orders and materials entered the plant at one end and, much later, finished products exited at the other. As enterprises invested in enterprise resource planning (ERP) systems, the prevailing thought was that they would eventually improve visibility into manufacturing. The ability to unlock the operational data that would empower plant managers and executives to make informed, timely decisions has eluded many companies until relatively recently. However, a new generation of MES (manufacturing execution systems) and manufacturing intelligence systems is enabling the collection, presentation, and analysis of critical, real-time data to decision makers throughout the enterprise and the supply chain. Many technological innovations rely upon User Interface Design to elevate their technical complexity to a usable product. Technology alone may not win user acceptance and subsequent marketability. The User Experience, or how the user experiences the end product, is the key to acceptance. And that is where User Interface Design enters the design process. While product engineers focus on the technology, usability specialists focus on the user interface. For greatest efficiency and cost effectiveness, this working relationship should be maintained from the start of a project to its rollout. When applied to computer software, User Interface Design is also known as Human-Computer Interaction or HCI. While people often think of Interface Design in terms of computers, it also refers to many products where the user interacts with controls or displays. Military aircraft, vehicles, airports, audio equipment, and computer peripherals, are a few products that extensively apply User Interface Design. GUI Test Automation plays a major role in organizations, because it tries to reduce man power, cost and time. It also improves the quality of the software. UI test plays a major role in testing all combinations of data test cases.

1.1 Graphical User Interface

Graphical User interface (GUI) developed by Gene Mosher on the Atari ST computer and first installed in restaurants in 1986 is an early example of an application specific GUI that manifests all of the characteristics of direct manipulation. In 1995 the View-Touch GUI was developed into an X Window System window manager, extending the usefulness of the direct manipulation interface to users equipped with no other equipment than networked displays relying on the X network display protocol. This application is a practical and useful example of the benefit of the direct manipulation interface. Users are freed from the requirement of making use of keyboards, mice and even local computers themselves while they are simultaneously empowered to work in collaborative fashion with each other in worldwide virtual workgroups by merely interacting with the framework of graphical symbols on the networked touch-screen.

Because of the difficulty of visualizing and manipulating various aspects of computer graphics, including geometry creation and editing, animation, layout of objects and cameras, light placement, and other effects, direct manipulation is an extremely important part of 3D computer graphics. There are standard direct manipulation widgets as well as many unique widgets that are developed either as a better solution to an old problem or as a solution for a new and/or unique problem. The widgets attempt to allow the user to modify an object in any possible direction while also providing easy guides or constraints to allow the user to easily modify an object in the most common directions, while also attempting to be as intuitive as to the function of the widget as possible. The three most ubiquitous transformation widgets are mostly standardized and are Translation, Rotation and scaling. Direct manipulation and UI design in general, for 3D computer graphics tasks, is still an active area of invention and innovation, as the process of generating CG images is generally not considered to be intuitive or easy in comparison to the difficulty of what the user wants to do, especially for complex tasks.
1.2 UI Test Automation

UI Test Automation is a combined process of UI Testing and UI Automation. Usability Testing is a mode of testing a particular product for its compatibility in terms of use. GUI testing is a commonly known form of Usability Testing of software or website. It is also known as User Interface Testing. GUI testing is a performance related assessment of a software or website in terms of ease of use, versatility, friendliness with focus on the target audience, visual impact and the approach and time taken to progress into the specific purpose.

The software or the website is released as a Beta version or as a pre release to a set of users to evaluate and assess the performance of the site or the software with focus on positive customer experience. Feedbacks from the users are taken as a survey based questionnaire or as a predefined test document of the functionality expected. The basic idea of going through this exercise is to allow for the opinions and the responses from the view of the cross section of the audience concerned. The test results are then evaluated and changes implemented in the respective software or website.

In the view of web applications, Usability testing is becoming popular as user centric design has become the focus of websites. This phenomenon has lead to a new genre of employable professionals called usability engineers whose responsibility includes positive focused friendly customer experiences. There are also many instances where sites like drugstore.com, altavista.com have been revamped as a result of evaluation and assessments of GUI Testing. GUI Testing can be classified into four stages – Low level, Application, Integration, Non-functional.

Testing Automation is a process of doing the testing of the application using appropriate tools and following various testing methodology. Different tools are available from various vendors with concentration on different type of testing. But the fact also remains that the blending of manual and automated testing methods is the best way to test any application. Testing Automation should concentrate on the following factors: Test process improvements, requirement definition, feasibility, interface testability, maintenance and Reusability.

1.3 Objectives of UI Test Automation

The Existing system requires a lot of manpower and UI Test Automation time is more. Silverlight is new Microsoft technology and there is no proper Software for UI Test Automation. This system is developed in such a way that the testers can do UI Test Automation in user – friendly manner. The main objectives of the proposed system are identified as:

- Online UI Test Automation
- Dot Net Silverlight Applications User Interface Testing
- Testing User Interface with Business logic
- More Secured page User Interface Testing
- Improves the quality of the Applications
- Permutation and combinational test cases testing and automation
- Manual testing, Cost and Time is reduced
- Three layers(UI/Business/Data) testing and automation

2 Existing Automation Systems

There is a need for this application as there is no system to do UI test Automation. This is the only system can perform UI Test Automation the system requirements are gathered from the concerned staff.

The proposed system aims at providing facilities for testers online. The Web- system contains three core subsystems namely the UI Accessibility, UI Test and UI Automation. It is proposed to build the system in a web-based model. Having the system online will increase the amount of Testers whom can use the system simultaneously compare to the limitation of a stand-alone application. More number of users can access the system simultaneously for online UI Test Automation.

There are several systems for Automation like Apriso, AspenTech, Camstar, InvensysWonderware, GE Fanuc, ICONICS, OSIsoft, Rockwell Automation, Siemens UGS PLM Software, Siemens A&D AS MES, Honeywell’s POMSnet, etc. has both advantages and disadvantages.
3 Proposed System Analyses

3.1 Problem Statement

The costly and massive effort required to implement and integrate information systems that share real-time manufacturing data throughout the organization is a significant impediment to creating more agile manufacturing businesses and industries. Typically, factory-floor information systems focus on the operation of production equipment and the control of processes. They communicate neither directly nor regularly with front-office information systems or with design and engineering systems. As a result, upstream information systems are unaware of important manufacturing details, such as the availability of appropriate tools, labor, and materials; maintenance schedules; records of past process performance; or the status of work in progress.

Middle-level information systems, known as manufacturing execution systems (MES), bridge this critical information gap between upstream and downstream activities. Today's MES solutions, however, are burdened by complexities that make them difficult to implement and integrate and, often, even more difficult to modify and upgrade.

MES applications track and manage all aspects of a job on the shop floor, at any point in the production cycle, in near real time. MES applications play essential supervisory and monitoring roles that link all levels of manufacturing and business operations. For example, they identify bottlenecks and material shortages on the shop floor, and they provide up-to-the-minute process performance results along with comparisons to past performance and to projected business results. While these tasks are important to the management and operation of all manufacturing businesses, only a small fraction can afford the cost of installing and maintaining an integrated MES solution, estimated to below one million. The sizable costs incurred for maintenance and systems integration work required to upgrade or otherwise change the system place MES further out of the reach of smaller manufacturers.

Across discrete manufacturing industries, adoption of MES has been slight and uneven. Large companies in industries with substantial regulatory and reporting requirements, such as the pharmaceutical, aircraft, and defense industries, are the most common users of MES, followed by semiconductor manufacturers, which manage complex, hard-to-control production and assembly processes.

Business goals

New proposal will achieve the benefits of the three existing approaches. Like point solutions, an integrable MES can be assembled from piece-like and reusable components. And, like proprietary solutions and commercially available suites of applications, integrable MES provides comprehensive, system-wide monitoring and reporting. But, unlike existing approaches, integrable MES will be more comprehensive, and will accommodate rapid customization, incremental installation, as-needed reconfiguration, distributed monitoring and control, and enhanced information flow throughout the enterprise.

Anticipated outcomes of the capabilities enabled by integrable MES and to be realized by companies of all sizes across a wide range manufacturing industries include:

• Significant reductions in manufacturing costs and cycle time, enabling cost-effective production of both large and small lot sizes and customized products.
• Performance advantages offered by previously unaffordable automation tools
• Significant reductions in time-to-market for new products.
• Reductions in the time and cost of starting up new factories and of changing over.
• Reductions in inventory and higher levels of capacity and resource utilization.
• Reductions in the time and cost of establishing and operating tightly coupled yet flexible supply chains.

Technical goals

This program's technical goals are to research, develop, and validate the technologies, methods, and infrastructure needed to reduce the cost and time required to integrate manufacturing execution systems into a manufacturing enterprise. This may be accomplished through a combination of infrastructure and the development of easily integrable applications. This program intends to ease the flow of real-time manufacturing execution data among MES applications, as well as to make real-time data available upstream to executive information systems (EIS) and enterprise resource planning (ERP) systems, laterally to design systems, and downstream to equipment control systems.
The research will support a real-time manufacturing environment which is rapidly and easily reconfigurable, promoting increased agility, and exhibits the following characteristics:

- Integrability of manufacturing software across platforms and vendors' product lines.
- Configurability of a factory's manufacturing system architecture to allow for rapid introduction of new products, and rapid response to unanticipated changes in requirements.
- Adaptability of a factory's manufacturing system architecture to automatically adjust to changing work loads and conditions.
- Extensibility of solutions to complex manufacturing requirements within single factories, to other sites, and to other phases of the product life cycle.
- Reliability of distributed systems and integrity of shared data.

Technology Barriers

There are numerous technical barriers, some owing to the complex nature of real-time data, which must be overcome to achieve manufacturing execution system interoperability in a general and reusable manner. The following lists some of the unresolved issues:

- The automation of legacy system integration is not well understood, and requires techniques to map information and behavior across different technology generations and system architectures. The real-time nature of MES data further complicates the integration of manufacturing data within an enterprise.
- A certain degree of functional tailoring will always be necessary to meet the needs of a specific, changing manufacturing facility. No formal, widely applicable methodologies exist.
- Adaptable systems require new ground to be covered in the area of autonomous software systems. Research has not matured in areas such as agent technology, self-describing interfaces, support for multiple communication paradigms, knowledge representation, sharing and reuse, capture of knowledge and context, information mining, and self-modifying software entities.
- Technologies to support self-integrating applications are currently non-existent. This is similar in concept to installation programs for personal computer applications that query the characteristics of the computing environment to customize the installation.

Proposed Research

- This program will support research which addresses technical goals and barriers described above, including improved visualization and UI test automation.
- It tries to solve the problems in integration, deployment and consistent UI of Manufacturing Execution Systems.

3.2 Proposed Work

The proposed work must do UI Testing and Automation for new technology Silverlight web applications. Now a day's most of the MES Applications are developed in Silverlight. The new system must perform better UI Test Automation to reduce time, cost and man power. Proposed system will Test UI with bossiness logic and improves the quality of the software.

Enterprise Resource Planning (ERP) system is used to integrate all activities in an organization, but it does not satisfy the need of automated manufacturing organizations. So Manufacturing Execution Systems (MES) framework evolved, which tries to address the problems in , deployment, Consistent UI, etc. that exists in present manufacturing applications.

Graphical User interfaces (GUI) are recognized as a major improvement over character based user interfaces. However GUIs are inherently more complex to develop and test because of their unconventional programming style and multiple asynchronous input sources. In addition the graphical nature of the objects in a GUI adds a number of dimensions to the results which need to be verified. The flexibility and variety in look-and-feel provided by the GUI toolkits of today make their testing even harder.

Testing of UI is always challenging and has been mostly manual till now. In the next generation of MES applications the testing of GUI needs to be automated as it will save lot of man-hours that is lost in manual testing also it will catch the defects early on in the cycle. Try to solve the problems while integrating the MES applications, UI Test Automation, newer visual methods to depict data in business applications will be key goals of this project.
3.3 Technologies Used

Silverlight: Silverlight is Microsoft's new cross browser or delivering richer interactive applications to users over the web. Silverlight 2.0 is Microsoft's second release of the Silverlight platform. Silverlight is a Microsoft .NET new UI Language for attractive web applications. Silverlight is developed by Microsoft on Dot Net 3.0 Framework. So Silverlight provides security up to the level of Dot Net 3.0 Framework. But Silverlight controls are written by a language called XAML. Silverlight 2 ships with a "lightweight" version of the full .NET Framework, which features, among other classes, extensible controls, XML Web Services, networking components, and LINQ APIs.

LINQ: Language Integrated Query for Data accessing by service oriented method without Database connection.

XAML: eXtended Application Markup Language is an XML-based language of description of interface. It is at start intended by Microsoft to be a means of creating rich Internet applications. XAML is recognized by Internet Explorer, Firebox and mozilla with the advantage of a larger number of users.

4. Proposed System Design

4.1. Input Description

The system is web based UI Test Automation application, so the input to the system is supplied to web pages by programmatically. Web pages contain several controls to gather input. The sources for gathering input are text box, Button, dropdown list, list box, radio buttons, checkbox, date Grid component, tree view, form and mouse clicks. Some of the inputs are compulsory and some are optional. For the optional inputs the system assumes the default input values.

- Text box: Text box controls are used to gather textual information. Some text boxes contains default values obtained from database or a predefined value, if the user wants to change the contents, Values are passed to text box by programmatically. Text boxes are associated with validation rules.
- Button: It is used to execute Click events and associated validation rules.
- Dropdown list: It allows UI Test Automation to choose one value from several values. Most of the dropdown list controls in the system are associated with DB.
- List box: List box allows UI Test Automation to choose more than one value from several values. Most of the list boxes controls are associated with DB.
- Date component: Date component is used to select a date into text box. Date component displays the date object, the user can navigate (i.e. move to required date) through the object and can choose a date.
- Grid view: Grid view control is very useful for Database record operations. UI Test Automation can supply record to Grid view by Program.
- Tree view: Tree view is very useful for tree operations. UI Test Automation can select node from the tree by program.
- Form: Form is container for the web controls used to transmit the data from client to server. When the form is submitted, in the client side, form is validated. UI Test Automation performs operations on page controls by program.

The system input is processed using the database UI Test Automation and if necessary the input information is maintained in the database.

Customer Entry Form

Customer Entry form is the web page used by the UI Test Automation system to automate the system. Table 1 shows Customer Entry form user interface details.
On clicking the button in the form, client side validation ensures that the entries are not null and not exceeding characters. On the server side, the information is checked to see whether it is a valid one. On error conditions the appropriate error message will be displayed in the output excel sheet.

### 4.1.2. Input Design

The input to the system is crucial to the working of the system. So the designers and developers should emphasize on the validation of the input. Most of the input submitted to the server page will be validated in the client side, some input will be validated in the server side. The input gathered through the web pages, it will be processed and the resultant action will be taken. The processing of the input includes validation, checking the input with the data in the database. When required, the input will be maintained in the database.

For example the input from the Customer entry form will be processed using the table “CustomerTable” as shown in table 2.

### Table 1. Customer Entry form UI details.

<table>
<thead>
<tr>
<th>Input Name</th>
<th>Input Type</th>
<th>Description</th>
<th>Validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Name</td>
<td>Text box</td>
<td>Input is the name of the customer.</td>
<td>The customer name entry should not be null, not exceed 25 characters. The entry should be valid user name of an existing customer name.</td>
</tr>
<tr>
<td>Address</td>
<td>Text box</td>
<td>Input is the address.</td>
<td>The customer address entry may have characters or empty.</td>
</tr>
<tr>
<td>City</td>
<td>Text box</td>
<td>Input is the city of the customer.</td>
<td>The customer city entry should not be null, not exceed 25 characters. The entry should be valid city of an existing customer name.</td>
</tr>
<tr>
<td>Country</td>
<td>Text box</td>
<td>Input is the country name of the customer.</td>
<td>The country name entry should not be null, not exceed 25 characters. The entry should be valid user name of an existing customer name.</td>
</tr>
<tr>
<td>Phone</td>
<td>Text box</td>
<td>Input is the phone number.</td>
<td>The phone entry should not be null, not exceed 20 digit numbers.</td>
</tr>
<tr>
<td>Add</td>
<td>Button</td>
<td>Add new record</td>
<td>Add button is invoked by UI Automation program to add a new record.</td>
</tr>
<tr>
<td>Update</td>
<td>Button</td>
<td>Update record</td>
<td>Add button is invoked by UI Automation program to Update a new record.</td>
</tr>
</tbody>
</table>

On submitting the customer entry form, the information will be verified. If the data are valid, then by invoking Add button click the information will be added to database.

### 4.1.3. Input Interfaces

The input interface in the system is very important, as it is where the customer gets satisfaction with the system. The input interface of the system is effective and user friendly. The system makes the user feel comfortable with the input. The system has dynamic behavior in the input interface.
1. Many input fields in the web pages of the system have default values. This allows the user to skip the fields. If the input is not the default value then the user can enter required values. For example, the UOPSSModel for model browser is the default value.

2. Some input fields behave dynamically. The fields are disabled and depending on the status of other inputs the field can be activated.

3. Some input fields will be depending on other inputs.

4. Some input field like email should contain ‘@’ symbol.

When the form is submitted the form will be validated in the client side to ensure that the required input entries are gathered and the values are valid entries. For the error conditions the system generates the appropriate error message.

4.2 Output Description

The system generates various types of outputs ranging from a small message to a big report. Most of the system outputs are generated using the database tables. The system generates different outputs based on the data passing via UI Test Automation.

Some of the system outputs are as follows:

1. Status report. Description: The report lists the successfully updated records in the table for a particular table.
2. Test report. Description: The report lists the test status done by UI test.

4.3 Procedural Description

Procedural description defines the process between received input data and Processed Output data. We can know the process status by status report. It is process of converting input data to output data.

4.4.5 Modules Diagram

The whole UI Test Automation can be split into several modules based on their functionality. They are UI Accessibility, UI Test, UI Automation and UI Test Automation.
The Silverlight Web applications user interface test automation is divided into several modules as shown in figure 4.7.

5. Implementation and Testing

5.1 Implementation Details

Microsoft released Silverlight 2, a new version of its solution for creation and delivery of Rich Internet Applications (RIA’s) through a Web browser. Silverlight 2 presented under the motto of creating “next-generation Web experiences” comes packed with a variety of new features and tools that enable designers and developers to better collaborate while creating more accessible and more secure user experiences. This release is designed to further Microsoft’s efforts to make Silverlight, Visual Studio and Microsoft Expression Studio the preeminent solutions for the creation and delivery of Rich Internet Applications and streaming media.

5.1.1 Silverlight

Silverlight Features for “Next-Generation Web Experiences”

Highlights of new Silverlight 2 features include:

- .NET framework support with a rich base class library; this is a compatible subset of the full .NET Framework.
- Built-in controls, including DataGrid, ListBox, Slider, ScrollViewer, Calendar controls and more.
- Skinning and templating support to make it easier to customize the look and feel of an application.
- Deep zoom to enable interactivity and navigation of ultrahigh resolution imagery.
- Networking support allows calling REST, WS*/SOAP, POX, RSS and standard HTTP services.
- Advanced content protection: Silverlight DRM, powered by PlayReady, offers robust content protection.
- Improved server scalability and expanded advertiser support: This includes new streaming and progressive download capabilities, search engine optimization techniques, and in-stream advertising support.
• Partner ecosystem: Visual Studio Industry Partners are providing products that further enhance developer capabilities when creating Silverlight applications in Visual Studio.
• Cross-platform and cross-browser support, including Mac, Windows and Linux in Firefox, Safari and Internet Explorer.
• Openness and Interoperability

Microsoft also announced further support of open source communities by funding advanced Silverlight development capabilities with the Eclipse Foundation’s integrated development environment (IDE) and by providing new controls to developers with the Silverlight Control Pack (SCP) under the Microsoft Permissive License. Microsoft funded Soyatec, a France-based IT solutions provider and Eclipse Foundation member, to lead the integration of advanced Silverlight development capabilities into the Eclipse IDE. Microsoft also plans to release the Silverlight Control Pack (SCP) and publish on MSDN the technical specification for the Silverlight Extensible Application Markup Language (XAML) vocabulary. The SCP will be released under the Microsoft Permissive License, an Open Source Initiative-approved license.

The Silverlight XAML vocabulary specification, released under the Open Specification Promise (OSP), will better enable third-party ISVs to create products that can read and write XAML for Silverlight.

Silverlight developers could certainly use all these initiatives to advance their learning of how advanced controls are authored – in Visual Studio 2008, Expression Studio 2 or Visual Web Developer 2008 Express Edition, which are all supported with Silverlight integrations.

Silverlight is going Strong

“We launched Silverlight just over a year ago, and already one in four consumers worldwide has access to a computer with Silverlight already installed,” said Scott Guthrie, corporate vice president of the .NET Developer Division at Microsoft. Silverlight adoption is growing rapidly, with penetration in some countries approaching 50%, according to Microsoft. The 2008 Olympics Games in Beijing were covered by many media companies, including NBCOlympics.com – the site powered by Silverlight – that received more than 50 million unique visitors, resulting in 1.3 billion page views, 70 million video streams and 600 million minutes of video watched.

The sheer popularity of the Olympic Games accounts in part for such impressive traffic numbers. But Silverlight certainly helped to deliver such a rich user experience.

Broadcasters in France (France Televisions SA), the Netherlands (NOS), Russia (Sportbox.ru) and Italy (RAI) also chose Silverlight to deliver Olympics coverage online.

Microsoft reports that many other organizations, such as CBS College Sports and Blockbuster Inc., are using Silverlight to build their next great offerings.

Silverlight Architecture

This topic describes the essential architecture and components of Microsoft Silverlight as shown in figure 3.
Silverlight is not only an appealing canvas for displaying rich and interactive Web and media content to end users. It is also a powerful yet lightweight platform for developing portable, cross-platform, networked applications that integrate data and services from many sources. Furthermore, Silverlight enables you to build user interfaces that will significantly enhance the typical end user experience compared with traditional Web applications.

While Silverlight as a client-side runtime environment seems simple and compact in size, the Silverlight development platform integrates a number of features and complex technologies, making them accessible for developers. To create effective Silverlight-based applications, developers need a working knowledge of the platform architecture.

There is a particular value in the combined set of tools, technologies, and services included in the Silverlight platform: They make it easier for developers to create rich, interactive, and networked applications. Although it is certainly possible to build such applications using today's Web tools and technologies, developers are hindered by many technical difficulties, including incompatible platforms, disparate file formats and protocols, and various Web browsers that render pages and handle scripts differently. A rich Web application that runs perfectly on one system and browser may work very differently on another system or browser, or may fail altogether. Using today's large array of tools, protocols, and technologies, it is a massive and often cost-prohibitive effort to build an application that can simultaneously provide the following advantages:

- Ability to create the same user experience across browsers and platforms, so that the application looks and performs the same everywhere.
- Integration of data and services from multiple networked locations into one application using familiar .NET Framework classes and functionality.
- A media-rich, compelling, and accessible user interface (UI).
- Silverlight makes it easier for developers to build such applications, because it overcomes many of the incompatibilities of current technologies, and provides within one platform the tools to create rich, cross-platform, integrated applications.

Core Presentation Components

The core presentation features of the Silverlight platform, shown in the previous section and illustration, are described in the following table.

---

**Figure 3. Architecture of Silverlight.**
Table 3. Features of the Silverlight platform.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Handles inputs from hardware devices such as the keyboard and mouse, drawing, or other input devices.</td>
</tr>
<tr>
<td>UI rendering</td>
<td>Renders vector and bitmap graphics, animations, and text.</td>
</tr>
<tr>
<td>Media</td>
<td>Features playback and management of various types of audio and video files, such as .WMP and .MP3 files.</td>
</tr>
<tr>
<td>Controls</td>
<td>Supports extensible controls that are customizable through styling and templating.</td>
</tr>
<tr>
<td>Layout</td>
<td>Enables dynamic positioning of UI elements.</td>
</tr>
<tr>
<td>Data binding</td>
<td>Enables linking of data objects and UI elements.</td>
</tr>
<tr>
<td>DRM</td>
<td>Enables digital rights management of media assets.</td>
</tr>
<tr>
<td>XAML</td>
<td>Provides a parser for XAML markup.</td>
</tr>
</tbody>
</table>

Developers can interact with this presentation framework by using XAML to specify presentation details. XAML is the primary point of interaction between the .NET Framework and the presentation layer. Developers can programmatically manipulate the presentation layer using managed code.

5.1.2 UI Test Automation

Basic guidelines for making UI available

The ability to uniquely identify and locate any control within the UI provides the basis for automated test applications to operate on that UI. Programmatic access to UI elements requires that all UI elements are labeled, property values are exposed, and appropriate events are raised. For standard WPF controls, most of this work is already done through the AutomationPeer class. Custom controls require additional work to make sure that programmatic access is correctly implemented.

Programmatic access to all UI elements and text

User interface (UI) elements should be configured to enable programmatic access. If you are working with a standard WPF control, support for programmatic access is built into the control. If the control is a custom control—a control that has been derived from an existing control or a control that has been derived directly from the Control class—you must check the related AutomationPeer implementation for areas that might need modification. To improve testability, make sure that every control in the application has been assigned an AutomationId value (one of the key Microsoft UI Automation properties) that is unique and language neutral. An AutomationId value that is consistent from build to build makes it easy to identify the control in the visual tree, compared to searching for the control by another method.

Adding names, titles, and descriptions to UI objects

Assistive technologies, especially screen readers, use the title to identify the location of the frame, object, or page in the navigation scheme. Therefore, the title must be descriptive. Similarly, for WPF controls, the NameProperty and HelpTextProperty values are important for assistive technology devices and for automated testing. This is especially important for WPF ItemsControl objects (TreeView, ListBox, ComboBox, etc.), because the individual item's AutomationId value might be reused in a different subtree under the shared parent.

When an ItemsControl instance is bound to an XML data source, the assistive tool (such as the Narrator application) uses the ToString method to get the value of each item in the ItemsControl instance. This value is simply the string "System.Xml.XmlElement". To provide a meaningful value to the Narrator application, you can bind the AutomationProperties.Name property to the data source's property that is displayed in the ItemsControl instance.

Programmatic events are triggered by all UI activities

If a control is derived from a standard control or from the Control class, you must check the related AutomationPeer class for areas that might need modifications. You must also expose related events for the new control as needed. By following these practices, you enable assistive tools to be notified of changes in the UI and to notify the user about these changes. For more information about how to create a custom control with AutomationPeer, see the section Custom Control Authoring and
Discovery of UI elements

This section focuses on how to use Microsoft UI Automation for Automated Testing. It provides a brief introduction to the UI Automation Object Model, outlines the steps for implementing UI Automation in a WPF application, lists best practices and several different approaches to allocating UI elements, and then provides code examples for these approaches.

The UI Automation API object model

Every UI element, such as a window, a button, and so on, is represented by the AutomationElement derived class in the System.Windows.Automation namespace of the UIAutomationClient assembly. An AutomationElement instance corresponds to a UI element regardless of the underlying UI Framework (WPF or Win32). All Automation elements are part of a tree, in which the root element represents the Windows desktop. Through the AutomationElement.RootElement static property you can obtain a reference to the Windows desktop element and from there find any child UI element.

AutomationElement objects expose control patterns that provide properties and events specific to common control types (such as windows, buttons, check boxes, and so on). Control patterns in turn expose methods that enable clients to obtain additional information about the element and to provide input to the element.

Best practices for obtaining UI Automation elements

The following are best practices for obtaining UI Automation elements:

- As a rule, obtain only direct children of the RootElement object. A search for descendants might iterate through hundreds or even thousands of elements, possibly causing a stack overflow. If you are trying to obtain a specific element at a lower level, you should start your search from the application window or from a container at a lower level.
- To find a known element (identified by its Name property, AutomationId property, or other property or combination of properties), it is easiest to use the FindFirst method. If the element to find is an application window, the starting point of the search can be the RootElement object.
- To find all elements that meet specific criteria that are related to a known element, use the FindAll method. For example, you can use this method to retrieve list items or menu items from a list or menu, or to identify all controls in a dialog box.
- If you have no prior knowledge of the applications that your client might be used with, you can construct a subtree of all elements that you are looking for by using the TreeWalker class. Your application might do this in response to a focus-changed event. That is, when an application or control receives input focus, the UI Automation client examines children and perhaps all descendants of the element that has received the focus.
- After you find the supported patterns for a given UI element, we strongly recommend that you do not call GetSupportedPatterns. Performance can be severely affected because this method calls GetCurrentPattern internally for each existing control pattern. If you can, call GetCurrentPattern only for the patterns that you need.

Ways of finding UI elements by using UI Automation

Developers and testers can use the following ways to find an element by using UI Automation:

- Search for an element's AutomationId value. Note that the AutomationId value could be reused in the descendants. For more information, see Use the AutomationID property on the MSDN Web site.
- Search based on the localized control name.
- Search based on a control type.
- Search based on a PropertyCondition value.
- Obtain a control reference in an event handler.
- Search a ListItem object.

- Search based on a ClassName property value.
- In a multi-threaded apartment (MTA) application, create a single-threaded apartment (STA) thread for accessing UI that might appear broken.
- Use the WPF Dispatcher object to automate the AutomationElement object on the UI thread

If your client application might attempt to find elements in its own user interface, developers and testers must make all UI Automation calls on a separate thread.

Sample codes for locating UI elements

This section includes examples that show how to locate UI elements. The following code examples are excerpts from the code that is available in the sample test project listed under Custom Control Authoring and Testing Sample later in this document.

Example 1. This example shows how to find a button that has the AutomationID value “button1” in a WPF application, and then click it.

```csharp
/// Finds a UI Automation child element by AutomationID.
/// </summary>
/// <param name="automationID">AutomationID of the control, such as "button1".</param>
/// <param name="rootElement">Parent element, such as an application window, or AutomationElement.RootElement object when searching for the application window.</param>
/// <returns>The UI Automation element.</returns>
private AutomationElement FindElementByID(String automationID, AutomationElement rootElement)
{
    if ((automationID == null) || (rootElement == null))
    {
        throw new ArgumentException("Argument cannot be null or empty.");
    }
    // Set a property condition that will be used to find the control.
    Condition c = new PropertyCondition(AutomationElement.AutomationIdProperty, automationID, PropertyConditionFlags.IgnoreCase);
    // Find the element.
    return rootElement.FindFirst(TreeScope.Element | TreeScope.Children, c);
}
```

Example 2. This example shows how to find a control by control name.

```csharp
/// Finds a UI Automation child element by name.
/// </summary>
/// <param name="controlName">Name of the control, such as "button1".</param>
/// <param name="rootElement">Parent element, such as an application window, or AutomationElement.RootElement object when searching for the application window.</param>
/// <returns>The UI Automation element.</returns>
private AutomationElement FindElementByName(String controlName, AutomationElement rootElement)
{
    if ((controlName == null) || (rootElement == null))
    {
        throw new ArgumentException("Argument cannot be null or empty.");
    }
    // Set a property condition that will be used to find the control.
    Condition c = new PropertyCondition(AutomationElement.NameProperty, controlName, PropertyConditionFlags.IgnoreCase);
    // Find the element.
    return rootElement.FindFirst(TreeScope.Element | TreeScope.Children, c);
}
```
5.2 UI Automation with API

In order to use the UI Automation API, we need to add references to these assemblies provided by Microsoft:
- UIAutomationClient.dll
- UIAutomationTypes.dll

For this simple implementation, I used the base class AutomationElement, which, as I said earlier, keeps the reference to a UI element from desktop applications. The idea is that we need to start from the desktop element, which is the root element, AutomationElement.RootElement as shown in figure 4, and we will search through all the child objects of RootElement for a test application having the title: "UI Automation Test Window". After getting a valid reference to the AutomationElement of the test application, we can then interact with the different contained controls.

![Automation Elements Hierarchy chart.](image)

In this way, the controller application sets some values to two TextBox controls as shown in figure 4.

```csharp
AutomationElement rootElement = AutomationElement.RootElement;
if (rootElement != null)
{
    Automation.Condition condition = new PropertyCondition(AutomationElement.NameProperty, "UI Automation Test Window");
    AutomationElement appElement = rootElement.FindFirst(TreeScope.Children, condition);
    if (appElement != null)
    {
        AutomationElement txtElementA = GetTextElement(appElement, "txtA");
        if (txtElementA != null)
        {
            ValuePattern valuePatternA = txtElementA.GetCurrentPattern(ValuePattern.Pattern) as ValuePattern;
            valuePatternA.SetValue("10");
        }
        AutomationElement txtElementB = GetTextElement(appElement, "txtB");
        if (txtElementA != null)
        {
            ValuePattern valuePatternB = txtElementB.GetCurrentPattern(ValuePattern.Pattern) as ValuePattern;
            valuePatternB.SetValue("5");
        }
    }
}
```

Here is the GetTextElement function:

```csharp
Private AutomationElement GetTextElement(AutomationElement parentElement, string value)
{
    Automation.Condition condition = new PropertyCondition(AutomationElement.AutomationIdProperty, value);
    AutomationElement txtElement = ParentElement.FindFirst(TreeScope.Descendants, condition);
    return txtElement;
}
```
As we can see in the above code, we are using a control pattern, ValuePattern. UIA (UI Automation) uses control patterns to represent common controls. Control patterns define the specific functionality that is available in a control by providing methods, events, and properties. The methods declared in a control pattern allow the UIA clients to manipulate the control, for example, the ValuePattern.SetValue method. Besides ValuePattern which represents a control that stores a string value, as a different example of control pattern, we can take the InvokePattern control, InvokePattern, which represents the controls that are invokable, controls like buttons. In order to use a control pattern, we need to query the object to see what interfaces are supported, and only after getting a valid control pattern can we interact with it by using its methods, events, and properties. The following list shows the most common control patterns:

- SelectionPattern - used for controls that support selection, controls like ListBox, ComboBox
- TextPattern - used for edit controls
- ValuePattern - used to get/set a value on a control that does not support multiple values
- InvokePattern - used for invokable controls
- ScrollPattern - used for controls that have scrollbars
- RangeValuePattern - used for controls that have a range of values

The following example shows how to use a InvokePattern; in other words, it will click a button contained in parentElement:

```csharp
Automation.Condition condition = new PropertyCondition(AutomationElement.AutomationIdProperty, "Button1");
AutomationElement btnElement = parentElement.FindFirst(TreeScope.Descendants, condition);
InvokePattern btnPattern = btnElement.GetCurrentPattern(InvokePattern.Pattern) as InvokePattern;
btnPattern.Invoke();
```

5.4 Testing

5.4.1 Test Cases

The test cases are designed separately for Manufacturing Execution System (MES). Test cases for MES Site Specific configure system are as follows in figure 5 and 6.

Figure 5. Test cases for MES Site Specific configure page.
5.5.2. Test Strategy

The test plan corresponds to the processes that are used to test an application. For a web-based application, the plan followed is, first to test the screens that relate to the user interface which would be done with the help of the unit test cases that are prepared before the actual coding of the JSP pages is done. These test cases would cover all the possible situations that are expected. After the unit testing is done, the integration testing is done within the various modules of the application and then by actually hosting on the testing server of the UnitedPro intranet. This exhaustive testing procedure ensures the quality of the system. The system is tested by the team comprises of the testing professionals as well as system developers.

6. Results and Discussions

Performance analysis is used to test the run time performance of software. A process that analyzes an application’s performance for the potential problems is necessary to ensure that the software meets or exceeds design expectations. This process is known as performance analysis. It involves examining an application to ensure that each component works efficiently and according to design, paying close attention to input/output.

The proposed system has been developed according to the requirements of the system. The system has been implemented in the Internet. The system has been tested using various test cases designed by the development team as well as testing team by a team comprising of testing professionals as well as developers. The system is functioning as proposed for the different test cases developed. It also generates the appropriate error messages when required.

The system is capable of handling several users simultaneously. The system is functioning efficiently in the Internet as proposed. When the user consults with domain experts, they can chat as they are sitting together.

The system is preventing the unauthorized access to the system. It ensures the security of the data. It also prevents unauthorized access of the web pages of the system in the Internet. The system responds to the client requests within 1 second. The system’s interface is effective. The results generated by the system are effective and are highly satisfactory to the organization.

The performance analysis is done in terms of security, reliability, portability, Integrity and processing time. We maintained records in various databases like Text, Excel, XML, MS-SQL, Oracle. In each database we maintain the same records for this performance analysis. The UI Test Automation is initially started with 5000 records, then 10000, 15000 and 20000. Finally the average case is taken for our analysis.
Table 4. Different DB records and their UI Automation Processing time.

<table>
<thead>
<tr>
<th>Databases</th>
<th>Time</th>
<th>Time for UI Automation (NS)</th>
<th>AVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Records</td>
<td>input</td>
<td>5000 10000 15000 20000</td>
<td>12500</td>
</tr>
<tr>
<td>Text (Time in ns)</td>
<td>250 750 1300 1750</td>
<td>2250</td>
<td>1512.5</td>
</tr>
<tr>
<td>Excel (Time in ns)</td>
<td>300 800 1100 1800</td>
<td>2300</td>
<td>1500.0</td>
</tr>
<tr>
<td>XML (Time in ns)</td>
<td>100 600 1400 1600</td>
<td>2100</td>
<td>1425.0</td>
</tr>
<tr>
<td>MSSQL (Time in ns)</td>
<td>400 900 1450 1900</td>
<td>2400</td>
<td>1662.5</td>
</tr>
<tr>
<td>Oracle (Time in ns)</td>
<td>450 950 1000 1950</td>
<td>2450</td>
<td>1587.5</td>
</tr>
</tbody>
</table>

The graph is plotted for the average case. In this Experiment the average number of records is taken as 12500 and the same records are maintained in all data files like text file, Excel sheet, XML Data, MS-SQL and Oracle table. The graph shows that the UI Test Automation time Using XML is very less when compared with others. The overall UI Test Automation processing time for each database is shown on the top of each column as shown in the bar figure 7 (Bar chart).

The Approximately estimated time in nano Seconds for 5000, 10000, 15000 and 20000 records is given as shown in the table 4.

Table 5. Performance comparison table for various DBs.

<table>
<thead>
<tr>
<th>DBs</th>
<th>Data Entry</th>
<th>Space for data</th>
<th>Security</th>
<th>Input Time</th>
<th>Report Output</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Easy</td>
<td>Small</td>
<td>No</td>
<td>Less</td>
<td>Fast</td>
<td>Better</td>
</tr>
<tr>
<td>Excel</td>
<td>Easy</td>
<td>Small</td>
<td>Less</td>
<td>Less</td>
<td>Fast</td>
<td>Better</td>
</tr>
<tr>
<td>XML</td>
<td>Moderate</td>
<td>Small</td>
<td>Less</td>
<td>Fast</td>
<td>Best</td>
<td></td>
</tr>
<tr>
<td>MSSQL</td>
<td>Difficult</td>
<td>Large</td>
<td>More</td>
<td>More</td>
<td>Slow</td>
<td>Good</td>
</tr>
<tr>
<td>Oracle</td>
<td>Difficult</td>
<td>Large</td>
<td>More</td>
<td>More</td>
<td>Slow</td>
<td>Good</td>
</tr>
</tbody>
</table>

In this performance analysis, we considered the Easy of data entry, Space required for data, input process and Output report generation time for UI Test Automation. Even though databases like MS-SQL and Oracle provides more security, XML data set is best for UI Test Automation. Usually managing text files are very easy. So entering data in text files and excel sheet are easy, but these kinds of data does not provide security. Developer, tester and User can choose Database according to their needs. But for UI Test automation tester XML is best because of its good performance. The performance comparisons are shown in Table 5.

There are several methods for UI automation, but AutomationPeer method is best for Silverlight applications. MS-SQL and Oracle Database provide more security, but it takes more access time when compared with XML data. Developer can choose any database according to their needs, but XML is best input data for Silverlight UI Automation because entering data is easy.


References

7 Conclusion

In the present situation where the technology is the buzzword and has revolutionized the way we work and live, we would be the losers if we do not keep up with the changing world. Moreover, it makes a world of difference and a whole of sense to break-up from the age old work culture and embrace the effective, cost, and time saving ways of looking and working at things. This system is useful to both the user and organization to do UI Test Automation. This System supports and improves many of the core functionality of the web pages i.e. It helps in quick and easy UI Test Automation for testers, Testers can do UI Test Automation in easier way. Such a system helps the organization in minimizing the time and cost to perform UI Automation.

Hence the Automation peer UI Test Automation plays a dominant role of modern world to do UI Test Automation for applications like Silverlight. There are several methods for UI automation, but AutomationPeer method is best for Silverlight applications. Developer can choose any database according to their needs, but XML is best input data for Silverlight UI Automation. To generate output report, Excel is best one when compared with others like Text, XML, MS-SQL and Oracle.

8 Future Enhancements

UI Accessibility and AutomationPeer Testing are extended to do test automation of moonlight Applications in Linux environment. In future this AutomationPeer Testing can be used for web Applications developed by JAVAFX and FLEX. In future UI Accessibility, Testing, Automation and Imputation of missing data can be done for moonlight applications.


Exterals Links