Technische Fakultät
Albert-Ludwigs-Universität, Freiburg
Lehrstuhl für Kommunikationsysteme
Prof. Dr. Gerhard Schneider

Master thesis

Abstract Unattended Workflow Interactions

January 18, 2012

Supervisors
Dirk von Suchodoletz
Klaus Rechert

First Reviewer
Prof. Dr. Gerhard Schneider

Second Reviewer
Prof. Dr. Christian Schindelhauer

Alibek Kulzhabayev
Matr.-Nr.: 2950774

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum Unterschrift

Acknowledgment

I thank Prof. Dr. Gerhard Schneider for allowing me to write the thesis at Chair of Communications Systems. I am thankful to Dr. Dirk von Suchodoletz and Klaus Rechert for their guidance and feedback to better structure and improve the thesis. My thank goes to Isgandar Valizada for giving his valuable hints throughout the work. I am grateful to my friends Arman Janabayev, Sergio Christian Herrera Salazar and Nathan Muwereza for reviewing the thesis and helping me with the correction of grammar and vocabulary mistakes.

Abstract

Recent studies in the domain of digital preservation have demonstrated the principle feasibility of the migration-by-emulation approach. The migration-by-emulation approach is a method aimed to recreate original environments of the obsolete digital objects and using such environments, to automatically convert large amounts of digital artifacts, from obsolete formats into up-to-date formats. However, there are some issues hindering the creation of fast and reliable migration workflows.

We studied a large number of migration workflows, and we provide an abstract description of the migration workflow based on that study. In such a description the mechanism to automatically handle unsuccessful migration workflows is also provided.

Since we consider unattended migrations of a numerous digital objects, an approach of an execution of the migration worfklow for each digital object that is being migrated is not time-efficient. The abstract description of the migration workflow provided in this thesis, describes the process to repeat certain part of the migration workflow, according to the number of digital objects to be migrated. That allows to migrate large number of digital objects within one migration worfklow execution, which would increase the speed of such an execution. Moreover, we implemented a feature to create stages, which in this case serve as identifiers of the mentioned repeatable part of the migration workflow.

By using the abstract description of the migration workflow and the feature to create stages, a user can choose or rearrange certain parts of the multi-format migration workflow, allowing the user to select the output formats needed. Such a workflow migrates a digital object from one format to several different ones.

In this thesis, hindrances to create fast and reliable migration workflows are analyzed, and methods to optimize them are designed and implemented. About 3,35 times execution speed acceleration on 50 sample digital objects of such optimized migration workflows is observed in comparison with the ones that were created without our optimization method.

Zusammenfassung

Die jüngste Forschung im Bereich der digitale Archievirung hat die prinzipielle Realisierbarkeit des "migration-by-emulation" Ansatzes gezeigt. Dieser Ansatz ist eine emulationsbasierte Methode, die ermöglicht eine große Anzahl an digitalen Artifakten automatisch von alten Formaten in zeitgemäße Formate zu konvertieren. Allerdings gibt es einige Probleme, die eine Erstellung von verlässlichen und schnellen Migrationsarbeitsabläufen verhindern.

Wir haben eine große Anzahl von Migrationsarbeitsabläufen untersucht und liefern darauf basierend abstrakte Beschreibung von Migrationsarbeitsabläufen. In dieser derartigen Beschreibung wird auch der Mechanismus zur automatischen Behandlung fehlerhafter Migrationsarbeitsabläufe bereitgestellt.

Da wir eine unbeaufsichtige Migration vieler digitaler Objekte annehmen, ist der Ansatz eines Migrationsarbeitsablaufs pro digitalem Objekt das migriert wird, nicht zeiteffizient. Deswegen beschreibt die in dieser Arbeit vorgestellte abstrakte Beschreibung für Migrationsarbeitsabläufe ein Verfahren, um bestimmte Teile des Migrationsarbeitsablaufs, in Abhängigkeit von der Anzahl der zu migrierenden digitalen Objekte, zu iterieren. Zusätzlich, wurde ein Feature implementiert um Abschnitte zu erstellen, die in diesem Fall als Bezeichner für die weiter oben erwähnten wiederholbaren Teile der Migrationsarbeitsabläufe dienen. Das erlaubt die Migration einer großen Anzahl digitaler Objekte innerhalb einer Ausführung eines Migrationsarbeitsablaufs. Dadurch erhöht sich die Geschwindigkeit der Ausführung von Migrationsarbeitsabläufen.

Durch die Nutzung der abstrakten Beschreibung von Migrationsarbeitsabläufen und dem Feature um Abschnitte zu erstellen, kann der Benutzer bestimmte Teile des Migrationsarbeitsablaufs auswählen oder anders anordnen, um das jeweilige benötigte Ausgabeformat auszuwählen. Solche Migrationsarbeitsabläufe migrieren ein digitales Objekt von einem Format in verschiedene andere Formate in einer einzigen Ausführung des Migrationsarbeitsablaufs.

In dieser Arbeit, werden die Probleme bei der Erstellung von schnellen und verlässlichen Ausführungen von Migrationsarbeitsabläufen analysiert und Methoden entwickelt um solche Arbeitsabläufe zu optimieren. Ein Test mit 50 digitalen Objekten zeigt eine 3,35 fache Beschleunigung in der Ausführungszeit des optimierten Migrationsarbeitsablaufs im Vergleich zu dem nicht optimierten Arbeitsablauf.

Contents

1 Introduction

"Those who forget the past are condemned to reload it"
Nick Montfort, July 2000

Most digital artifacts were created by using interactive graphical applications, that were available at some point of time. Many national libraries, archives and organizations, whose main function is to preserve publications and records, have large amounts of such artifacts. They need a tool to automatically process them, in order to make these digital objects available to their users.

Recent studies (e.g., [1] [2]) in the digital preservation domain have shown the advantages and feasibility of the migration-by-emulation approach. The migration-by-emulation - is an approach to migrate digital objects (DOs), that is, to convert DOs from an obsolete format into a currently accessible one, within emulated original environments (hardware, operating system, drivers and other third-party libraries). Since DOs can be best rendered by the application where they were produced, providing the original environment is necessary to use those applications. As the original environment is not always available due to obsolescence, emulation of such kind of environment is crucial to render obsolete DOs.

Migrating DOs manually is tedious and an error-prone task. Additionally, the knowledge on how to use such applications is vanishing. Since most of DOs are being created by GUI (graphical user interface) applications, there exist the same sequence of interactions, that allows to migrate almost every DO of the same format. We call the mentioned sequence of interactions migration workflow. For that reason, those interactions in the migration workflow can be recorded once, and replayed, according to the number of DOs that are being migrated.

The Interactive Workflow Recorder (IWRec) is a tool for transferring human interactions to the emulated system and also registering them in a log-file - Interactive Workflow Description (IWD). User uses IWRec to migrate a sample object of an obsolete format to the format of interest. DO can be injected into the emulated system and extracted from it by using certain virtual storage device.

Subsequently, the Interactive Workflow Replayer (IWRep) can replay recorded user actions from IWD over any DO of the same format. As previously, DOs can be injected and extracted into the emulated system by utilizing certain virtual storage device. As soon as migration is performed, the DOs become recognizable by at least one currently available application.

Although, the migration of DOs is feasible, the execution of the migration workflows of the migration-by-emulation approach, in the following denoted as EMWs, is not sufficiently fast and reliable. The EMW is reliable if every interaction of the migration workflow (MW) is processed in the target system and an expected outcome of such interaction is obtained.

Several reasons why some EMWs fail are described in [3, ch. 3]. However, such reasons are not fully studied, hence neither their complete consequences on the migration of DOs. Most of the failures may be related to the following reasons: properties (e.g., event processing time) of the operating systems (OSs); properties (e.g., unexpected modal windows) of the applications that renders DOs; settings of the IWRep. Thereupon, we analyze the reasons of the failures and their consequences in the EMWs.

If large quantities of DOs have to be migrated, time of the EMWs becomes a crucial parameter. Since IWD is replayed with the same speed at which a user recorded it, delays from the user interaction are replayed too. Such delays are unnecessary to be replayed in the EMWs and they just increase the overall duration of the EMW. Therefore, the ways to eliminate them has to be considered.

Additionally, the MWs, that we study, consist of a numerous of pointer movements, but some of them are not necessary to be replayed within the EMWs, because they do not play a role in the migration. Therefore, they may be omitted. One approach that considers not to record pointer movements at all, is presented in [3]. However such an approach does not work in some OSs (e.g., Windows 95, Windows 98). Therefore, the properties of some OSs to be tested, that is, the ways that such OSs process pointer movements have to be analyzed in order to improve the EMWs.

If it is necessary to migrate DO not only to one format but to several ones, then it is easier and faster to make that in one MW, that produces several DOs of the required formats. In that case, it should be possible for the user to rearrange or delete some parts (stages) from an IWD in order to obtain the migration of the DO to preferred format(s). Those parts are the sequences of the descriptions of the interactions, where the conversion of a DO to the required formats takes place. Overall integrity of the MWs in any case, has to be preserved. With the current MWs it is not possible to perform the mentioned operations, because a user is not able to mark the stages of

the MWs. Hence, it is not possible to rearrange or delete them. Furthermore, there is no abstract description of the MW, which specifies how to modify the MWs.

Additionally, since a certain part of the MWs is related to migrate a DO, such a part may be repeated a given number of times, according to the number of DOs to be migrated (cf. [2, ch. 5.1.2]). Therefore, within one MW it could be possible to migrate a set of DOs. The parts of the MWs that can be repeated should be specified.

During the execution of some MWs, unexpected warning or error modal windows may appear. This kind of modal windows disrupt the consecutive order of interactions within the mentioned MWs. A user should create different additional MWs, which deal with different types of errors. In that case, the user has to be able to mark the part of the additional MWs, which corresponds to the interactions, performed to get rid of the unexpected modal windows. Moreover, a mechanism to handle such type of the errors should be designed.

2 Background

The chapter gives an overview on how and using which tools migration workflows are created.

2.1 Definition of Terms

In order to proceed further, the following concepts should be clear:

- Interactions – events such as pointer movements, pointer clicks, pointer drags, key strokes and synchronization conditions (e.g., SyncPoints), workflow structuring means (e.g., stages).
- SyncPoint – a screenshot serving as synchronization condition between screen states of a host system and an emulated system. It is used as precondition to invoke next interactions of the migration workflow.
- Workflow – an ordered list of interactions.
- Primary objects – an obsolete format digital objects of interest.
- Secondary objects – applications, helper programs and drivers [4], necessary to migrate primary objects to the digital objects of the currently accessible format.
- Migration Workflow – an ordered list of interactions, aimed to migrate particularly digital objects of an obsolete format to the format accessible by at least one current application.
- Interactive Workflow Description – a digital object containing the workflow with the description of each interaction, contributor had invoked. For example, for pointer click it could be a position of the pointer, button pressed, timestamp, modifier; for SyncPoint - position and color of each pixel.

- Interactive Workflow Recorder[1] – a tool to connect to the host system by using network address of the mentioned system and to transfer interactions to an emulated system through certain network port. Additionally, this tool registers interactions that are being sent into the interactive workflow description.
- Interactive Workflow Replayer[2] – a tool to read interactions from an interactive workflow description and to transmit them to the emulated system through certain network port and by using the network address of the host system.
- Container – virtual equivalent of a hard drive or other media that stores the content used by the emulator [5]. The content used in migration-by-emulation approach may be an operating system with secondary objects [4] or the obsolete digital objects and digital objects migrated to the format accessible by at least one current application.
- Workflow Registration Component[3] – a back-end component that is used for the contribution of the migration specific information [2].
- Migration Component – a front-end component that uses the migration specific information in order to perform data migration [2].
- Contributor – a user that utilizes workflow registration component, interactive workflow recorder to create certain interactive workflow description for further unattended migration of obsolete digital objects.
- End User – a user that works with the migration component or directly with the interactive workflow replayer and migrates obsolete digital objects. End user should specify the format of the obsolete digital object and the resulting migrated digital objects, upload the obsolete digital object and download the migrated digital object. Container in this case, serves as the media to save necessary operating system with secondary objects, and as a storage to inject and extract the digital objects in and from the mentioned operating system.
- Guest System – an emulated operating system with the secondary objects. Mainly saved in the container and run by the emulator.
- Host System – an operating system that starts an emulator with mainly two containers: one with operating system with secondary objects, and second with the digital objects to be migrated or already migrated ones. Additionally, emulator is passed parameters to enable network port, through which interactive workflow recorder or interactive workflow replayer can access a guest system.

[1] in [2] referred as Interactive Session Recorder
[2] in [2] referred as Interactive Session Replayer
[3] in [2] referred as Scenario Registration Component

- Remote System – an operating system that end user uses to migrate digital objects. Abstract notion that maybe identified as host system or other remote operating system that have migration component and/or interactive workflow recorder, that accesses the guest system through specified network port and certain network address of the host system.

2.2 Hardware Emulation

The term emulation is used in computer science to denote a range of techniques, which use some software in place of a different software and/or hardware to achieve the same effect as using the original. "The theory behind emulation is that the only way to ensure the authenticity and integrity of the record over the long term is to continue to provide access to it in its original environment..." [6].

One type of the emulators considered in the thesis are hardware emulators. This kind of emulators can emulate an environment, that is, OS and corresponding hardware. Hence, they can emulate obsolete OSs and implement the functionality of such devices as CPU, I/O devices, and memory components. Some examples of hardware emulators are: QEMU [7] and Dioscuri [8]. The mentioned type of emulators provide an access to the obsolete DOs by emulating certain OS with the secondary objects and corresponding hardware. The secondary objects have to contain appropriate application to access and migrate the DOs.

2.3 Interactive Workflow Recording

The contributor is a user that performs all tasks in the interactive workflow recording (recording) phase. This user creates a container both to inject and extract DOs and to install OS with secondary objects. Further on, the containers can be injected into the hardware emulator to migrate DOs and to run OS. The contributor also specifies a network port that the hardware emulator opens to access the guest system and network address of the host system.

IWRec connects to the guest system by using the network port and the network address specified by the contributor. After the guest system is loaded, the contributor can start migrating the sample DO of necessary format. IWRec transmits all interactions of the user to the guest system for further processing and also registers them in IWD.

Examples of the interactions that the contributor invokes are the pointer movements, clicks and drags, key strokes, SyncPoints, that is, conditions to synchronize screen states of the remote and the guest system. All these interactions with corresponding to them information such as e.g. type of the button, key code, pixel color is registered at the same pace as user invokes them into the IWD.

Of particular interest are the SyncPoints that synchronize a screen state of the guest system with a remote one. Mainly it is square area of specified size (in [9, pg.3] area of 10 by 10 pixels). IWRec registers to the IWD each pixel position and colour in that square. The SyncPoints serve as a synchronization condition and are very important for reliable EMWs. They stop the execution of the next interactions in MW until they match, which means until previously invoked interaction has an expected outcome in the guest system. In the next section, another type of the SyncPoints – pattern-matching SyncPoints will be considered.

The pointer movements within the EMW are also important to be taken into consideration, because during recording, offset between pointer in the remote and guest system take place, in the following denoted as pointer offset. This offset plays a role in reliable EMWs, because for an unskilled user, it may take days to record MW where all interactions will have an expected consequences in the guest system. The main reason why the pointer offset occurs is that the remote and the guest system are two different and independent OSs. Additionally, the current network protocol (the RFB protocol [10]) that is used to transmit interactions does not allow to control the pointer of the guest system, since it is OS-independent or solution is not found. In the future, possibility to improve the mentioned protocol or to use another one may be investigated. One approach [3] in order to eliminate the pointer offset was to identify pointer and its corresponding position in the guest system by using pattern-matching and move the pointer in the guest system to the position of the pointer in the remote system. Since, the reasons and their consequences of failing EMW are not fully analyzed and the implementation of the mentioned approach does not work in the replaying phase, the approach can be considered in the future work after the analysis, that will be made in our study.

Pointer acceleration (also called mouse acceleration) also results in the pointer offset [3]. However, the exact influence of the pointer acceleration and also existence of other reasons consequent to the failing EMWs were still not analyzed. Heavy load of the CPU or input and output requests to the primary drive of the host system probably cause interruptions in the work of guest system, hence some pointer shifts in the guest system may also take place. Those reasons and their consequences have to be identified by performing specific test cases. That information may allow to understand the causes of the failing EMW and ways (if any) to improve the reliability of them.

As soon as a sample object is migrated, the guest system is turned off and migrated DO is extracted from the container. The contributor can check the migrated DO and its content. If significant properties of the migrated DO corresponds to the sample DO, then the IWD is ready to be used for the unattended replaying.

The workflow registration component should be then supplied with the required information on input and output format of the primary object, container name containing preinstalled OS with secondary objects, name of the IWD. Other parameters as the name of the emulator, container name that is used for injecting and extracting primary object, network port and address are assumed to be entered.

After these operations, the recording phase finishes and primary objects can be migrated in the replaying phase.

2.4 Interactive Workflow Replaying

The user who is involved in replaying phase is called an end user. The end user can choose a migration component, special client application – UfcClient, or perform manual migration depending on the task the end user has to carry out.

The migration component is a front-end which allows the end user to choose input and output format of the primary object, upload the primary object and after the migration is finished to download the migrated DO. Based on the information that the contributor registered in the workflow registration component, the migration component automatically starts IWRep, hardware emulator and migrates the DO. After migration is finished, it responds either with the failure if the migration was unsuccessful, or with the migrated DO.

Another tool that allows to perform automatic migrations of DOs is UfcClient. This application was implemented based on the thesis described in [2]. UfcClient can be run from the command-line and with certain script, it is able to migrate large-quantities of DOs automatically. Moreover, (using the short script) it is possible to log failure and successful EMWs, their time and the DOs properties, corresponding to the MWs. EMWs can be evaluated for success and failure rates and such migration can help identify main error causes (if any) and error types.

The process when the end user directly uses IWRep is called a manual migration. In this case, the end user should first inject primary object to migrate in the appropriate container. Afterwards, the mentioned user runs the hardware emulator with the above-named container and the container with the necessary preinstalled OS and

secondary objects. Additionally, the end user indicates the network port to access the guest system. As the next step, IWRep is started with the network address of the host system and the network port that the hardware emulator opened to access the guest system. Which IWD to use should be specified when starting IWRep, too.

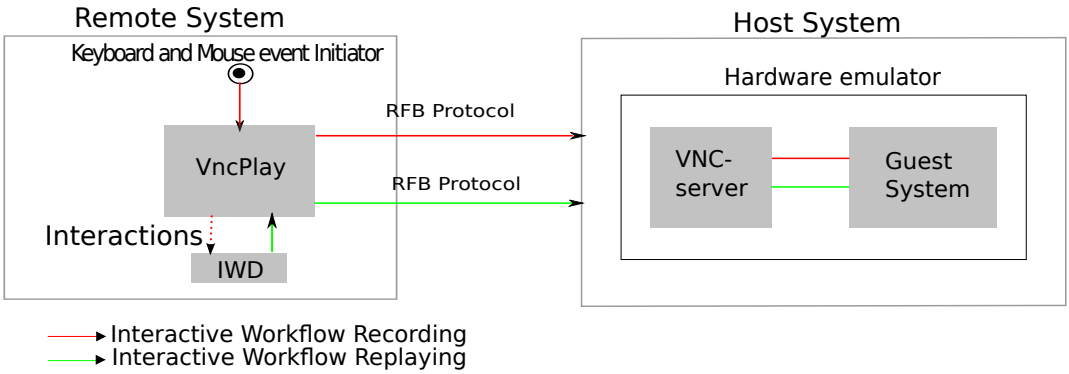


Figure 2.1: Interactive Workflow Recording and Replaying. Interactive Workflow Recorder and Replayer Tool Example – VncPlay. Hardware emulator has built-in VNC-server.

During replaying different behaviour of the interactions rather than in recording are noticed. Particularly, pointer movements resulted in different pointer offset during recording and within replaying. Since, the thesis is aimed to speedup and to improve if possible reliability of the EMWs, causes and effects of the shifts have to be analyzed and recommendations or solutions for improvement have to be devised.

Different types of SyncPoints – pattern-matching SyncPoints were proposed in the thesis [11]. Within the recording of the MW, SyncPoints registered in the IWD as before - pixel color and position. However, during replaying based on the pixel values from the guest system and IWD distances are calculated. Whenever two distances of the pictures are the same – result is zero, otherwise it is 1. In the thesis [11], the method is considered to be more efficient than the bitmap-matching one [9]. Hence, possibility to use either of the SyncPoint should be provided.

3 Interactive Workflows Analysis

In order to identify the reasons of why some EMWs fail, analysis of the different issues such as the causes of the offset between the pointers in the host and in the guest system, processing time and properties of the interactions, in this case, pointer movements, in the guest systems (sample OSs to be tested are Windows 3.11, Windows 95 and Windows 98) have to be identified.

Afterwards, duration of the EMWs has to be evaluated as well as the methods of its reduction. The ways to increase the speed of the EMWs have to be based on the analysis of the mentioned processing time and properties of the interactions.

The pointer offset is one of the reasons of the failing EMWs. Therefore, its causes and consequences should be analyzed thoroughly with the results both in the recording and in the replaying phases. Related test cases are to be performed : *Event Generation Time*, *Pointer Acceleration*, *Pointer Movement Distance*.

Furthermore, the heavy load of the primary disk and the CPU of the host system has to be made. Effect of such load on the interactions such as the pointer movements has to be identified. Related test cases are to be performed: *Pointer Movements Processor Load*, *Pointer Movements Disk I/O Load*.

Feasibility to eliminate delays made by the contributor during MWs recording, should be considered. Since, in the replaying phase how all types of the interactions were processed has to be analyzed. Related test cases are: *Pointer Movement Distance*, *Workflow Optimization*.

Results of such tests could give an understanding of how interactions in the guest system processed. Using such results feasibility to perform more reliable and fast EMWs should be examined.

Additionally, EMWs have to be evaluated for different parameters as time, success and failure rates, possible reasons invoking errors and type of errors. Based on that information, possibilities for further improvement have to be considered. Related test cases are: *Large-Scale Format Migrations*, *Failure Migrations Analysis*, *Multi-Format Migrations*.

3.1 Pointer Offset

Pointer offset is a hindrance to perform reliable EMWs. There are many different reasons that can cause this problem. The main reason, however, is that the remote and the guest systems are two different OSs and that the current network protocol is not able to control interactions or give the result of the interactions transmitted to the guest system. Methods to improve the network protocol or use of other one can be considered in the future works.

Other reasons causing the pointer offset and their consequences should be evaluated. Such reasons include: time to process the interactions (e.g., pointer movements) in the guest system; heavy load of the CPU and primary disk of the host system; turned on and off pointer acceleration options in the guest system; distance to which pointer can be sent when one pointer movement is generated. Depending on the last reason, test case to verify possible workflow optimization should be made.

3.1.1 Effect of Time Interval of Event Generation

Test case id: Event Generation Time

Unit to test: Verify whether pointer speed causes pointer offset. Results are to be classified by the operating systems – Windows 3.11, Windows 95 and Windows 98.

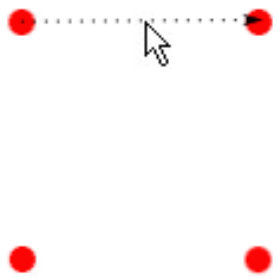
Prerequisites: The hardware emulator (QEMU) is started with VNC-enabled option. The wallpaper of the guest system contains positions of the pixels which are defined in test data. IWRec and IWRep tool is VNCplay. VNCplay is to be changed in order to automatically send pointer events to the designated positions on the screen within the specified time interval. Moreover, it should save pictures from the guest system for further analysis. Recording and replaying are to be extended with an option of saving pictures of the guest system's screen. After each loop there must be a pause to get updated screen snapshot from the guest system.

Test data: Time between each event (in ms) = {{0,5,15,30},{15,30, 50}}. There are four loops using which pointer moves on the square of size 100 pixels. Starting position is the center of the guest system screen. After each loop there is a pause of 3000 ms.

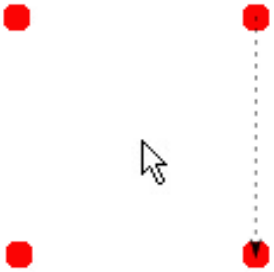
Steps to be executed:

1. Modify VNCplay to save pictures before and after each pointer move;

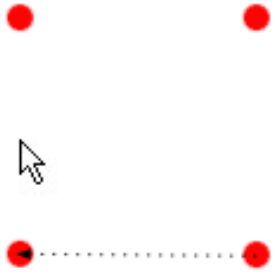
2. Change VNCplay to generate pointer movements to the designated positions and time intervals indicated in the test data ;
3. Start VNCplay;
4. Finish recording;
5. Evaluate pictures for pointer offset;
6. Repeat above-listed steps over each operating system.



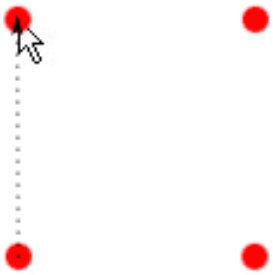
(a) Pointer movements sent from the position (400, 300) to (500, 300)



(b) Pointer movements sent from the stop place to the position (500, 400)



(c) Pointer movements sent from the stop place to the (400, 400)



(d) Pointer movements sent from the stop place to the (400, 300)

Figure 3.1: Example of pointer offset when pause between each event generation is 15 ms; distance between two successive points is 100 pixels (Windows 95); black dashed arrow line shows the expected result.

Expected result:

All the pointer movements specified in the test data should be processed correctly in

the guest system.

Actual result:

Windows 3.11 for Workstations:

This OS was evaluated with the time between each pointer movement sent (in ms): {0, 5, 15, 30}. Sending of pointer movements without a pause moved pointer to the wrong positions. First loop resulted in the correct pointer moves of approximately 10% but second loop moved the pointer up, nevertheless, movements was sent to shift pointer down. In the third loop, instead of going left, pointer moved down. During last loop pointer moved in right direction but only about 10% of events was processed. Nevertheless all events sent with time interval five ms and more were processed correctly.

Windows 95:

This OS was evaluated with the time between each pointer motion (in ms): {15, 30, 50}. 15 ms between each event resulted in processing of the half of the pointer movements. Result with 30 ms was better but about 5 pointer movements were not processed in each direction. With 50 ms delay, all events were delivered and processed properly.

Windows 98:

Handling of events in the test case of this OS did not differ noticeably from the result in Windows 95.

On the whole, all OS types processed events correctly with the time interval 50 ms between each event.

Comments:

Test cases of this section are executed on the computer – Intel(R) Pentium(R) 4 CPU 2.53GHz, 1 GB RAM, and 80GB HDD. CPU was overloaded with md5sum jobs and disk I/O heavy load was achieved by utilizing bonnie++ program [12].

3.1.2 Effect of Processor Load

Test case id: Pointer Movements Processor Load

Unit to test: Verify whether heavy CPU load causes pointer offset. Occurrences of so called lost interrupt calls has to be identified. Results are to be classified by the operating systems – Windows 3.11, Windows 95 and Windows 98.

Prerequisites: Hardware emulator (QEMU) is started with VNC-enabled option. The wallpaper of the guest system contains colorful positions of the pixels which are defined in the test data. The VNCplay is to be modified in order to automatically send pointer events to the designated positions on the screen within the specified time interval. Moreover, it should save pictures from the guest system for further analysis. Recording and replaying has to be extended with an option of saving pictures of the guest system screen.

Test data: Time between each event (in ms) = {{0,5,15,30},{15,30, 50}}. There are four loops using which pointer moves on the square area of 100 pixels. Starting position of the pointer is the center of the guest system screen. After each loop there is a pause of 3000 ms for getting a changed state of the screen of the guest system.

Steps to be executed:

1. Modify VNCplay to save pictures before and after each pointer move;
2. Change VNCplay to generate pointer movements to the designated positions and indicated in the test data time intervals;
3. Load CPU heavily;
4. Start VNCplay;
5. Finish recording;
6. Evaluate pictures for pointer offset;
7. Repeat above-listed steps over each of the three OSs.

Expected result:

Pointer movements may be delayed. Additional to the previous test case shifts may take place because of CPU loading.

Actual result:

Windows 3.11 for Workstations:

0 ms pause between each event generation caused that the guest system did not process events correctly. From the first loop like 20% of the pointer moves were handled. In the second loop, movements were sent to move the pointer down but result was opposite. Pointer moved approximately 3 times more distance that it supposed to in upper direction rather than down. Third loop was needed to generate events moving pointer to the left for 100 pixels. Instead of that, pointer was moved for

about 40 pixels down but also about 5-10 pixels to the left. Direction of the fourth loop was correct but about only 3-7 moves were processed. With 5 ms and more pause, pointer movements were properly processed.

Windows 95:

Pointer motions without any pause caused to move pointer in guest system to only 2-5%. 15 ms delay caused to move the pointer for about 50% of the distance. In each direction after pause between loops of events 3 sec, guest system processed exact number of pointer events. When pointer movements were sent every 30 ms, then percentage of correctly processed movements reached approximately 90 %.

Windows 98:

Pointer movements that were being sent without delay were processed not correctly in this guest system. Like 1-3 % were processed from the first loop, second loop which should move the pointer down, moved it to the right. Other loop events were also not correctly processed by the guest system . With 15 ms pause, only about 50 % of events were processed, others were ignored. 30 ms delay allowed guest system to interpret correctly like 80 - 90 %. Others were ignored or lost, as well. When pause became 50 ms guest system processed like 85 - 90 % of events from the first loop, other events were all processed.

Generally speaking, there was no much difference comparing to the previous test case result. To avoid all offset time should be increased for at least 10 more ms, which is 60 ms for all tested OSs.

3.1.3 Effect of Disk I/O Load

Test case id: Pointer Movements Disk I/O Load

Unit to test: Verify whether heavy Disk I/O load causes pointer offset. Occurrences of so called lost interrupt calls has to be identified. Results are to be classified by the operating systems – Windows 3.11, Windows 95 and Windows 98.

Prerequisites: The hardware emulator (QEMU) is started with VNC-enabled option. Wallpaper of the guest system contains colorful positions of the pixels which are defined in test data. VNCplay is to be changed in order to automatically send pointer events to the designated position on the screen within the specified time interval. Moreover, VNCplay should save pictures from the guest system for further analysis. Recording and replaying are to be extended with an option of saving pictures of the guest system screen.

Test data: Time between each event (in ms) = {{0,5,15,30},{15,30, 50}}. There are four loops using which pointer moves on the square of the size of 100 pixels. After each loop there is a pause of 3000 ms.

Steps to be executed:

1. Modify VNCplay to save pictures before and after each pointer move;
2. Change VNCplay to generate pointer movements to the designated positions and indicated in the test data time intervals;
3. Drastically load hard disk by large amount of I/O operations;
4. Start VNCplay;
5. Finish recording;
6. Evaluate pictures for pointer offset;
7. Repeat above-listed steps over each operating system.

Expected result:

Pointer movements may be delayed but should reach locations indicated in test data.

Actual result:

Windows 3.11 for Workstations:

Events that were generated without any pause were processed wrongly as in the previous test cases. The difference occurred when events were handled with 5 ms pause: about 92 -97 % of movements were processed. As it supposed to expect in this OS, when pauses between events were 15-30 ms - all movements were handled correctly.

Windows 95:

This time, results of this test case over this OS were different. In the first and second loop with 15 ms pause about 50 % were processed. But in the third one about 5 more events were handled which caused shift in the end of execution for 2-5 pixels to the left of the starting point. Pointer movements, sent with 30 ms delay in the first loop did not move pointer at all. OS could not react and just hanged. In the second loop about 80 % of events were processed. The third loop showed approximately the same situation. The fourth loop did not move the pointer. With 50 ms delay, all events were processed correctly.

Windows 98:

All loops with delay of 15 ms caused the guest system to handle only about 50 % of events. With 30 ms pause between each event, about 80 % of events were processed.

When pause was about 50 ms, three loops were processed correctly. Only in the fourth loop about 90 % of events were processed.

Generally speaking, small additional shift to few pixels in comparison to the previous test case has been noticed.

Results of these test cases show:

1. Fast pointer movements in all tested OS neither processed nor processed correctly.
2. Contrary to this, slower moves with 50 ms between each move to the next pixel handled by the guest system in all cases, when no load (CPU or I/O operations) applied, properly.
3. Windows 3.11 for Workstations OS processed events correctly when pause is 5 ms;
4. OSs as Windows 95 and 98 processed events properly when delay between each event was 50 ms;
5. When there was heavy load of CPU or I/O operations over primary disk of the host system, it was necessary to increase time of the pause between each event for about 10 ms;

Two last statements derive the following conclusion: OSs differ in their properties and particularly, how fast they process events. One of the reasons is the specific settings of the operating system as the interval time betweeen each event processing. Furthermore, IWD should be checked for the time between each event which could help identify whether the offset in EMWs take place because of the mentioned reason. Also, time interval in the IWRep with which interactions are extracted from the IWD also should be verified.

3.1.4 Effect of Pointer Acceleration

Test case id: Pointer Acceleration

Unit to test: Check whether pointer movements trigger pointer acceleration and identify possible offset of the pointer in the guest systems (Windows 3.11, Windows 95 and Windows 98), depending on the pointer acceleration level.

Prerequisites: Emulator is started with VNC-enabled option. Wallpaper of the guest system contains colorful positions of the pixels which are defined in Test data. Screen resolution is 800x600 pixels. If it is different, then positions in the test data

should be changed, according to the resolution. VNCplay is to be changed in order to automatically send pointer events to the designated position on the screen. Moreover, it should save pictures from the guest system for further analysis. Recording and replaying is to be extended with an option of saving pictures of the guest system screen after each pointer movement.

Test data: Pointer acceleration levels: No, Medium, High; there are four positions and five pointer events to be sent. Fist pointer movement is necessary to confirm that the pointer is in the center (400,300), and other four to move the pointer to positions: (500, 300), (500, 400), (400, 400) and (400, 300).

Steps to be executed:

1. Modify VNCplay to save pictures before and after each pointer move;
2. Change VNCplay to generate pointer movements to the designated positions;
3. Start VNCplay;
4. Finish recording;
5. Evaluate pictures for pointer offset;
6. Repeat above-listed steps over each operating system.

Expected result:

Pointer movements have to be exact and stop at marked locations.

Actual result:

Windows 3.11 for Workstations:

Absence of the acceleration in the guest system did not cause an offset in the pointer movements.
Medium acceleration moved pointer to the doubled distance. So in our case, movements were sent to 100 pixels and pointer moved to 200 pixels, hence offset 100 pixels. Full acceleration moved the pointer out of the screen, hence offset was at least 140 pixels. Note that the screen resolution in this OS was 640x480 pixels.

Windows 95:

Absence of the acceleration in the guest system did not cause an offset in the pointer movements;
Full acceleration moved the pointer out of the screen. One last movement was possible to measure – the distance of the offset was four times more than the actual pointer movement. So,the event was sent to move the pointer to 100 pixels, but final distance with the offset comprised 500 pixels.

Windows 98:

Results over this OS were identical with the previous one.

This test case proves and shows how exactly the acceleration in the guest system influence EMW. With the combination of the test case *Event Generation Time*, following statement can be derived: Pointer acceleration drastically influence pointer movement offset. To make EMW more reliable whether pointer acceleration has to be turned off and then movements to the various positions inside guest system with the interval 50 ms and more will be replayed correctly. Otherwise, if the switching the pointer acceleration off does not seem to be feasible, then pointer movements should be sent with 1 pixel distance and also with no less than 50 ms time interval, speaking generally.

Nevertheless, all three OS support easy switching off the pointer acceleration from the *Control Panel*. Of course, that is not a valid statement for all types of OSs (as only three Windows Operating systems were tested) but with the software implemented based on these test cases, it is possible to easily check other operating systems and come to further conclusions.

3.1.5 Effect of Pointer Movement Distance

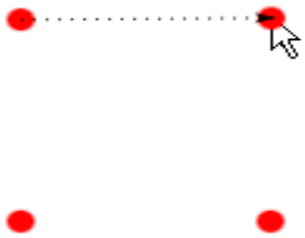
Test case id: Pointer Movement Distance

Unit to test: Actual MWs consists of large amount of pointer movements. It has to be verified whether it is possible to eliminate intermediate events. To do that *each pointer movement* has to be sent to the distances more than 100 (pixels).

Prerequisites: Pointer acceleration has to be turned off. Wallpaper of the guest system contains colorful positions of the pixels which are defined in the Test data. Screen resolution is 800x600 pixels. If it is different, then positions in the Test data should be changed, according to the resolution. VNCplay is to be changed in order to automatically send pointer event to the designated position on the screen within the specified time interval. Moreover, it should save pictures from the guest system for further analysis.

Test data:

Positions when the screen size is 800x600 pixels:



(a) Pointer Acceleration disabled, result is from the guest system – Windows 3.11, distance between two horizontal or vertical points is 100 pixels

(b) Pointer Acceleration enabled - medium level, result is from the guest system – Windows 3.11

(c) Effect of full pointer acceleration - pointer moved out of the screen, result is taken from the guest system – Windows 95;

Figure 3.2: Pictures show pointer acceleration effects depending on its level; dashed black arrow line shows the desirable result.

- ((400,300), (600,300), (600,500), (400, 500), (400,300)) – five pointer movements events are to be generated;
- ((400,300), (5,5), (795,5), (795,525), (5,525), (5,5)) – six pointer movements events are to be generated;
- ((400,300), (400, 5), (400,525), (5,5), (795,525), (795,5), (400,300)) – seven pointer movements events are to be generated;

Positions when the screen size is 640x480 pixels:

- ((320,240), (470,240), (470,390), (320,390), (320,240)) – five pointer movements events are to be generated;;
- ((320,240), (5,5), (635,5), (635,475), (5,475), (5,5)) – six pointer movements events are to be generated; ;
- ((320,240), (320,5), (320,475), (5,5), (635,475), (635,5), (320,240)) – seven pointer movements events are to be generated;;

Steps to be executed:

1. Modify VNCplay to save pictures before and after each pointer move;
2. Change VNCplay to generate pointer movements to the designated positions;
3. Start VNCplay;
4. Finish recording;
5. Evaluate pictures for pointer offset;
6. Repeat above-listed steps over each operating system.

Expected Result:

Pointer has to move to the designated positions since there is no pointer acceleration.

Actual Result:

Windows 3.11 for Workstations:

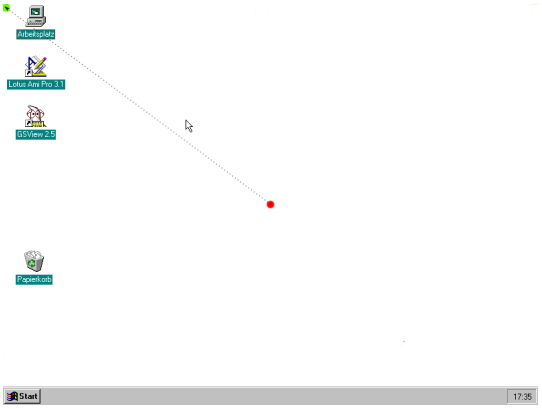
All positions were processed by the operating system correctly without any shift.

Windows 95:

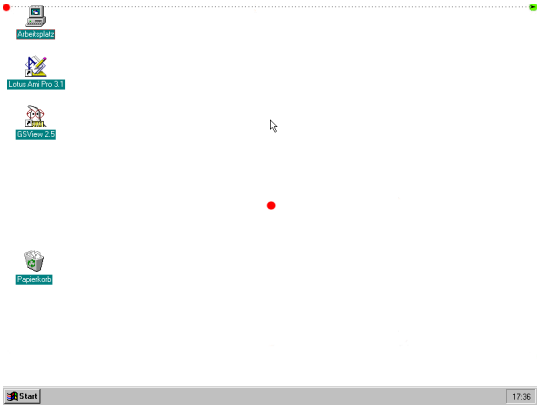
In the first list of positions specified in test data – pointer moved only to 126 pixels.
Using the second list in the test data – pointer moved only to 126 pixels.
Third time with the corresponding positions from the test data, the same result has obtained.

Windows 98:

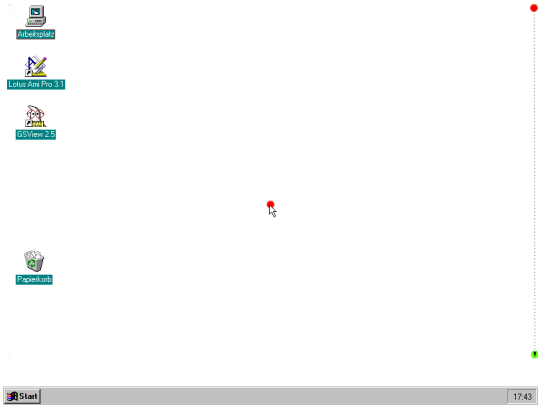
(a) Pointer movement to the position (400,300) to confirm that pointer is on the correct position



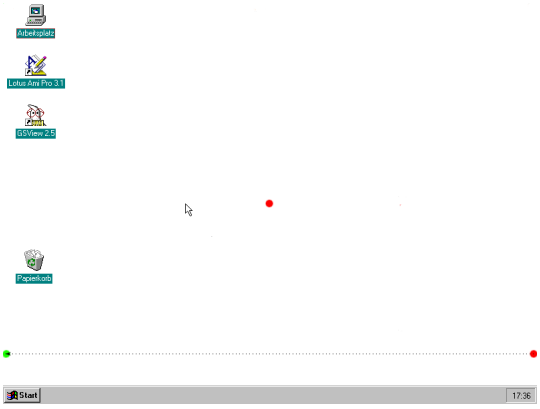
(b) Move pointer to the position (5,5)



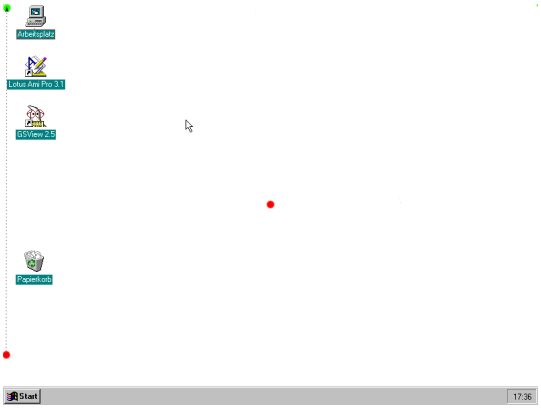
(c) Move pointer to the position (795,5)



(d) Move pointer to the position (795,525)



(e) Move pointer to the position (5,525)



(f) Move pointer to the position (5,5)

Figure 3.3: Pointer movements sent according to the second set of positions in the Test data with the screen resolution of 800x600 pixels. Screenshots taken from the OS – Windows 95.

First kind of test - pointer moved only to 127 pixels in all directions; Second type of test - pointer moved only to 126 and sometimes to 127 pixels; Third type of test - results are the same as from the second type of test with possible inaccuracy of one pixel.

The result of this test case is very important for the reduction of the duration of the EMWs. For example, approach in the thesis of Ruzzoli[3, pg.10] was to eliminate pointer movements which were between pointer clicks: "Es bietet sich also an, komplett auf die Aufnahme der Mausbewegungen zu verzichten und ausschließlich relevante Aktionen, also ein Betätigen der Maustasten, abzuspeichern". That could be time and event optimizing solution. But the test case shows that such an approach is not valid for all types of OSs. Although, in Windows 3.11 for Workstations it does work, in Windows 95 and 98 it does not. One pointer movement from one position to another cannot exceed 126 pixels. If it exceeds, then pointer just stops after reaching that threshold. Further solution to improve reliability and duration of EMWs should take into account this feature of the mentioned OSs.

3.2 Workflow Level

This type of the test cases are abstracted from the properties of the interactions and are aimed to register time, successful, failure rate and other characteristics of the EMWs. That is needed to reason on the reliability and time efficiency of the actual EMWs.

3.2.1 Large-Scale Format Migrations

Test case id: Large-Scale Format Migrations

Unit to test: Verify reliability, scalability and register time efficiency of the actual EMWs. Thus success and failure rates, total and singular runtime of the EMW has to be determined. Format migrations have to be made first and second time by using the same set of DOs and the same MW, third time to migrate different DOs but using the same MW. It is necessary to see how EMWs reliability depends on the chosen set of the DOs.

Prerequisites: Prepare software to make large-scale format migrations. Input format is DOC. Output format is RTF. The application that is used to migrate DOs is

MS Word 97 (8.0). The guest system is Windows 98.

Test data: 1000 DOs of one format.

Steps to be executed:

1. Create one MW with necessary number of SyncPoints to make EMW reliable as much as possible;
2. Migrate 1000 DOs from one format to another;
3. Migrate the same 1000 DOs using the same workflow;
4. Register necessary data;

Expected Result:

All 1000 DO should be migrated.

Actual Result:

Table 3.1 shows the results of the two migrations of one set of DOs migrated by the same MW. Table 3.2 presents the results of the migration of different DOs but using the same MW.

Criteria	**First Large-Scale Migration**	**Second Large-Scale Migration**
#of Input DOs	997	998
#of Output DOs	892, 89.47%	891, 89,28%
# of Failures	105, 10,5%	107, 10,7%
Avg time over all failures	00:15:00.89	00:14:52.49
Total time over all failures	26:16:32.95	26:31:36.20
Avg time over all successful migrations	00:03:58.57	00:03:59.14
Total time over all successful migrations	59:06:41.59	59:11:10.70
Avg time over all migrations	00:05:08.32	00:05:09.19
% Total time	85:23:14.54	85:42:46.90
Comments	Time format: [HH]:MM:SS.00	Time format: [HH]:MM:SS.00

Table 3.1: Results of the large-scale migrations of the same DOs with the same MW.

In the first large-scale format migration, three DOs were not sent to VncPlay – the possible reason ia the large size of the DOs 29-31 Mbytes. The mentioned issue is

Criteria	Third Large-Scale Migration
#of Input DOs	999
#of Output DOs	905, 90,6%
# of Failures	94, 9,4%
Avg time over all failures	00:14:19.20
Total time over all failures	22:26:04.33
Avg time over all successful migrations	00:04:04.01
Total time over all successful migrations	61:20:31.21
Avg time over all migrations	00:05:01.90
% Total time	83:46:35.54
Comments	Time format: [HH]:MM:SS.00

Table 3.2: Results obtained by using the same MW as in the first and second large-scale migrations but with different primary objects

related to SOAP messaging, because there is limit of the size of the message, in which DO is wrapped. In the second large-scale migration two DOs (size - 30, 33 (MB)) were not sent to VncPlay – the same reason as in the previous case. In the third migration one DO (size 29 (MB)) was not sent to VncPlay. This bug is neither related to the MWs nor to their execution, since its resolution will not be considered in this thesis.

Comments:

Test cases in this section are executed on the computer with the following characteristics: Intel(R) Core(TM)2 Duo CPU E7300 @ 2.66GHz, 4 GB RAM, and 250GB HDD.

3.2.2 Failure Migrations Analysis

Failed EMWs should be further analyzed to find out probable causes and failures. Therefore, failed DOs that are being migrated, should be run second time and error causes of the failing EMWs should be determined. Further on, recommendations and mechanism on how to eliminate the error causes are to be derived.

Test case id: Failure Analysis

Unit to test: Test failure EMWs from the first large-scale format migration for the possible causes and types of the errors.

Prerequisites: DOs that are not migrated in the first large-scale format migration. Use the same MW as in the tests – large-scale format migrations. Failures within EMWs have to be registered.

Test data: 105 DOs of one format

Steps to be executed:

1. Start large-scale format migrations;
2. Make 30 snapshots of the screen over 30 random DOs when possible failures of EMWs causes appear;
3. Analyze pictures for the possible failure causes of the EMWs;

Expected Result:

By using the pictures from the failure migrations, it should be possible to identify potential failure causes and error kinds.

Actual Result:

In failure cases, appeared three types of modal windows - first, notifying that there is no free space on the hard drive. Current size was 30 MB but for graphics conversions into specific format MS Word 97 needed more space. This type of the errors happened only when DO contained graphical content (see Fig. 3.4).

Second type of the error messages Fig. 3.5 appeared when primary object had different descriptors rather than the sample DO used to create IWD.

From failed in the first large-scale format migrations 105 DOs nine were migrated this time. Therefore number of not migrated DOs is 96 from 105.

Third type of the error or warning modal window in the Fig. 3.6 was related to macroses. User should have had invoked additional interactions to close this modal window.

Failure causes:

- Different than in the sample DO file descriptors caused additional pop-up messages to appear;
- Additional content as macroses of obsolete format which may generate warning messages;

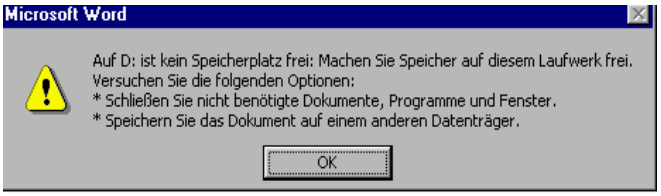


Figure 3.4: Example of the error, caused by not sufficient disk space

Figure 3.5: Example of the error, caused because of the different file descriptor (read-only)

- Heavy load of the host or guest system such that it could not process events;
- Not enough free space of the container used in the EMWs;

Recommendations to eliminate failure migrations:

1. File descriptors over all DOs to migrate and DO during recording has to be the same to prevent additional modal windows to appear;

2. To deal with modal windows such as e.g., macroses and other unexpected modal windows, MW should be described on an abstract level, such a description should contain ways to deal with the failing EMWs.

3. Increase the space of container from 30 (MB) to 500 (MB) where migration takes place;

4. Each interaction that result in the guest system screen change should be followed by SyncPoint in the place where such screen change takes place. One of the solution is to use only pointer events (movements, clicks) since the IWRec (actual used one is VNCplay) makes SyncPoints based on the pointer position. Second solution is to have a possibility during recording for each interaction that cause a change on the screen to make the SyncPoint on any position of the screen of the guest system with the feature to pause the recording.

3.2.3 Migration Workflow Optimization

Test case id: Workflow Optimization

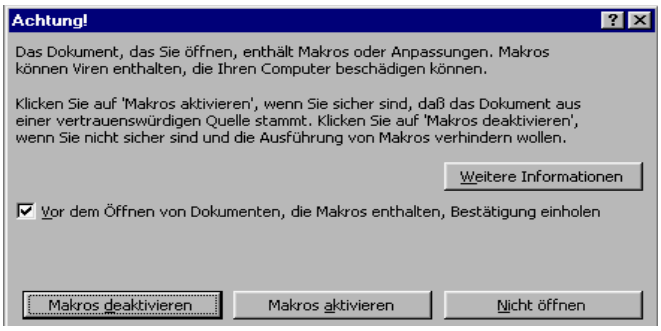


Figure 3.6: Example of the error, caused since DO contains macroses

Unit to test: Verify whether it is possible to remove intermediate interactions such as pointer movements from the IWD but not harm the integrity of the execution of the original MW.

Test data: distance = 50 (pixels); timer interval firing the event processing: 20 ms (actual value), 30 ms, 40 ms

Prerequisites: Pointer acceleration turned off.

Steps to be executed:

1. Create one MW with necessary number of the SyncPoints to make EMW reliable as much as possible;
2. Replay the MW;
3. Register pointer movements between two events such as key strokes or pointer click.
4. Delete all pointer movements that are between those two events which are less than 50 pixels.
5. Replay changed MW and compare the results with the original one.

Expected Result:

Pointer has to move to the same positions as in replaying the original IWD.

Actual Result:

- With the actual value of 20 ms, some events were not consumed in the replaying, resulting in offsets. Offset of about 3 pixels was noticed when replaying from the original IWD.
- With the value of 30 ms - offset of about 1-2 pixels is noticed when replaying from the original IWD.

- With the value of 40 ms - no offset was noticed. So replaying from original and modified (reduced in interactions) IWD did not differ in the result.

From this test case, it became evident that the actual time interval of the timer,that extracts interactions from the IWD is not sufficient for the replaying in the guest systems (Windows 95 and Windows 98) but sufficient for the Windows 3.11. Thus, time of the timer should be increased. Without doing it, optimization of the workflow by eliminating some kinds of interactions as pointer movements may lead to incorrect replayings, in comparison to the original IWD . This test case is much related to the first test case *Event Generation Time*. Its result may be abstracted on most OSs by checking the input processing characteristics of each particular OS, that was made and shown in this chapter.

3.2.4 Multi-Format Migrations

In this test case, one obsolete application has to be chosen and DOs have to be saved in the formats that the mentioned application provides. All format migrations have to follow one after the other. It has to be verified whether by using the actual MWs it is feasible to identify particular format migration(s). Moreover, possibility to rearrange or delete some parts of the MW in order to obtain the migration of the DO to preferred format should be checked.

Test case id: Multi-Format Migrations

Unit to test: Check whether it is possible to rearrange current multi-format MWs.

Prerequisites: Make one multi-format MW.

Test data: DO of certain format.

Steps to be executed:

1. Create multi-format MW;
2. Migrate the DO;
3. Analyze the MW;

Expected Result:

MW has to consist of the several format migrations which should be easily identified within the MW. It has to be possible to rearrange or delete some of the parts of the MW in order to get DO of necessary format(s).

Actual Result:

MW consists of overwhelmed number of registered interactions. They make the workflow not usable,that do not allow to reorder the execution. Without particular constraints and identification of the stages it is not possible to perform any rearranging. Because, firstly, such rearranging on the actual MWs may harm the integrity of the EMWs and secondly, the MW consists of more than two thousand of interactions, making it very difficult to identify which interaction where and at which format migration appeared.

3.3 Summary

In this section, conclusions over all test cases are to be derived. Related to the statement test case(s) is (are) to be given in parentheses.

It was shown that the pointer offset significantly influence the migration (*Event Generation Time, Pointer Acceleration, Pointer Movement Distance, Pointer Movements Processor Load, Pointer Movements Disk I/O Load*), that is, the reliability of the EMWs. The time with which user moves the mouse is not enough to the guest system to process interactions during the recording of the workflow (*Event Generation Time, Pointer Movements Processor Load, Pointer Movements Disk I/O Load*). Solution is to move the pointer slower or research in improving the existing network protocol between the guest and host system (if possible) but preserve independence of the mentioned protocol from the guest OS and emulated hardware.

In addition, pointer acceleration influences the pointer offset (*Pointer Acceleration*). In order to provide reliable recordings and replayings, it should be turned off in the guest system. All three tested OSs (Windows 3.11, Windows 95 and Windows 98) supported easy switching off the pointer acceleration.

The approach to eliminate all pointer movements as described in thesis [3] became obvious to be not working for at least two OSs (*Pointer Movement Distance*). Hence, it cannot be used as optimized solution. However, some interactions and time optimization of the EMWs can be derived after the results of test case: *Workflow Optimization.*

It was noticed that current time of the replaying interactions is not sufficient to correct replaying of them in the guest system. Hence, it has to be increased *Workflow Optimization.*

The pointer movements can be processed correctly if the distance of them do not exceed some threshold which also proves that all pointer movements cannot be eliminated - as it is described in [3]. But even if some intermediate interactions can be eliminated, to improve reliability of the EMWs, additional pointer movements before each pointer click should be registered into an IWD. It will confirm that the pointer movement and pointer click occur on the same position on the screen thus improving the reliability of the EMWs.

Additionally, the MW (that was tested in the test case *Multi-Format Migrations*) consist of more than two thousand interactions making it neither timely nor optimized by number of interactions. There is no feature of IWRec to structure the MW, which could allow rearranging of different format stages from the MW. Such a structure of the MW could also allow in the future combination of the stages from different MWs to guarantee automatic error-handling and cycling of some stages to migrate large-number of primary objects within one MW.

In order to determine actual state of the MWs, test case *Large-Scale Format Migrations* was needed. Moreover, we used result of the mentioned test case to identify main type of failures occurring within the EMWs (*Failure Analysis*). Solution to the appearance of the unexpected modal windows within EMW is to be provided in the next chapter.

3.4 Requirements

Based on the results from Summary section and objectives of the thesis, the following requirements on the design and the implementation of the software to improve MWs have to be defined.

1. Reliability
2. Time and Event Optimization
3. Integrity
4. Ability to rearrange
5. Usability

3.4.1 Reliability

EMWs is reliable if every interaction of the MWs is processed in the target system and an expected outcome of such interaction is obtained. If there will be reduction

of interactions in the MWs and speed increase in EMWs, then the reliability of the EMWs should be preserved, resulting in the same number or increased number of successful migrations.

3.4.2 Time and Interactions Optimization

The method to get rid of not necessary for the EMWs interactions has to be devised. It has to be based on the results of the analysis, made in this thesis. Delays depending on the user have to be excluded but EMWs has to be timely synchronized.

3.4.3 Integrity

The above-named reductions of the interactions and of the delays are to be made with some constraints. Without them EMWs may become unpredictable, error-prone and less deterministic, resulting in failure migrations. Conditions in which cases, what type of optimizations and when may be made are to be defined. Using that conditions (constraints) integrity of the MWs has to be preserved.

3.4.4 Ability to rearrange

There has to be a feature that allows a user to partition the MWs. The feature has to make it possible to rearrange and delete some parts of the workflow in e.g., multi-format MWs to choose necessary format migration. It should be possible to combine different parts of even different MWs and make a working MW. Certainly, in this case, previous requirement has to be taken into account.

3.4.5 Usability

The contributor should have possibility to think more to perform certain actions but that delays of the workflow should not be replayed within the EMWs. That could allow to create MWs resulting in more reliable execution. Hence, an approach implementing that has to be devised. Additionally, a feature to structure MWs has to be implemented, that could allow a user efficiently modify the MWs depending on the user's task.

4 Design

In this chapter, an approach to optimize the EMWs are presented. Furthermore, MWs described on an abstract level. That allowed to design an operation of repeating certain part of the MW to migrate set of DOs within one EMW. Additionally, after an abstract description of the MWs, it became possible to devise error-handling mechanism.

4.1 Migration Workflow Optimization

Reliability of the EMWs can be improved by increasing the time of the timer in IWRep at least in three tested OSs. The timer makes IWRep extract interactions one by one from the IWD after some particular time period. Current 20 ms is increased to 40 ms.

By removing excessive pointer movements from EMWs it is feasible to increase speed of the EMWs. The mentioned pointer movements are the pointer movements that are between key strokes or pointer clicks which does not change the screen state of a guest system. The pointer movements cannot be fully eliminated from the EMW because of the results of the test case Pointer movement distance. To recall, the test case states that some OSs (e.g., Windows 98, Windows 95) can move a pointer only to some specific distances by one event.

Our solution to that is :

1. To identify first and last pointer movements between pointer clicks;

2. Do not register in IWD all pointer movements which positions on the screen of the guest system are less than some threshold (currently chosen 100 (px));

3. To confirm that pointer click happens at the same position where last pointer movement appeared - generate and inject into IWD additional preceding to pointer click – pointer movement, which has the same position as pointer click;

4. To synchronize all interactions by time, threshold can be taken from the test cases Workflow Optimization, Pointer Movements Disk I/O Load, Pointer Movements Processor Load, Event Generation Time (currently chosen 40 (msec) for each interaction and applied for all interactions within an IWD.

This optimization drastically decreases the number of the interactions in the IWD. Reduced number of the interactions wintin an IWD make it possible to identify and edit the interactions within IWD. Stages to be described in the next section serving in this case as user-defined markers, will give possibility to exactly determine required interaction.

Besides, the optimization to be made to IWRec, registers in the IWD only necessary for the EMWs interactions, decreasing the overall time of the EMWs.

Additionally, delays when user thinks on the action to perform are ruled out. User has enough time to think which exact action necessary to commit.

Stress on both making reliable and fast EMWs is eliminated thus making recording more usable. After implementation of the feature it will be possible for a user to concentrate only on the task to make EMWs reliable not considering the pauses that the user makes.

4.2 Abstract Migration Workflow Description

In order to satisfy requirements reliability, integrity, ability to rearrange and usability, a MWs are to be described on an abstract level. Abstract description of the MWs should define conditions when the mentioned requirements will be satisfied.

Further on, feature to implement properties of the abstract MW is to devised.

On an abstract level, the MWs can be described as a list of stages. These stages have to be defined by the contributor during recording. As it is shown in the Fig. 4.2, there should exist mainly five stages to distinguish within each MW and two additional stages necessary to meet the requirements mentioned in this section:

1. *Start* – initial stage after guest system is loaded;

2. *LeftTop* – stage (additional), where contributor moves the pointer of the guest system to the position (0,0) and presses the mouse button to make sure, that the position is registered in the IWD and not deleted during the optimization.

This stage is needed to syncronize the pointers positions in the guest system and the remote one, allowing to make SyncPoints at the positions where both pointers located and after synchronizing the positions to control the pointer from the remote system so that no offset with the pointer of the guest system will take place;

3. *DOready* – stage, when DO to migrate is loaded into proprietary application and ready to be migrated;

4. *MigrateToFormatX* – stage, where actual migration to particular user-defined format take place, X is the extension of a format or multiple formats;

5. *CleanupL* – (or *CleanupM*) this stage is additional and necessary only when contributor wants to have some error-handling scenario within the EMWs; In order to use it, IWRep has to be extended to jump to that stage whenever conditions (e.g. time limit of SyncPoints mismatches) do not hold. L and M stand for any positive number and may be equal. These type of stages are processed when the stage name matches the specified in IWRep and in the IWD stage name, and if there is an error within the EMW is appeared.

6. *CloseApp* – Closing the application;

7. *End* – Stop of the replaying phase;

If a user wants to create multi-format MW, then it is necessary to create multiple *MigrateToFormatX* stages, where X will notate particular format or number in increasing order, corresponding to the number or extension of the format to be migrated. Moreover, not all stages are necessary to make format migrations. For example, stages *LeftTop* and *Cleanup(L or M)* are additional and necessary to improve the reliability of the EMW.

Decisions (diamonds) in the flowchart diagram Fig. 4.2 may check the following:

- SyncPoints match
- Time threshold to check SyncPoints
- Number of times to check one condition
- Unique title of the stage within the IWD

To fulfill Ability to rearrange and Integrity requirement pointer has to be moved to the position (0,0) after each stage (except the stages: *Start*, *CloseApp*, *Cleanup(L or M)*, and *End*).

When there exists MW containing multiple format migrations, then there is a possibility to the user to choose necessary stages of the MW, that migrate DO to certain

format. That could be made by the identity of the starting and ending state of all *MigrateToFormatX* stages. In order to identify the parts of the MW and make it more usable, stages should be marked, accordingly to the Fig. 4.2.

To maintain the integrity of the MW and implement the ability to rearrange its stages, the constraints on the starting and ending status of that stages have to be fulfilled.

The state of the stage is the screen state, pointer position, and stage title.

4.2.1 Mechanism of Stages Repetition

Stages *DOready*,*MigrateToFormatX*, *CloseApp* can be repeated, according to the number of the DOs that are being migrated. Therefore, the stage titles should be specified in the IWRep and then during the recording of the MW. Afterwards, number of DOs to migrate has to be read from the container. The stage titles should correspond to the stage titles specified in the IWRep. In the end, depending on the number of primary objects, repetitions of the mentioned stages within an IWD should be performed by the IWRep.

To open the next DO in the folder additional interactions may be necessary. Interactions that do choose particular DO (e.g. key strokes), can be marked using the stages. Afterwards, they can be repeated number of times multiplied by the number of the primary objects.

Another issue is the name of the DOs. Since in the recording phase one sample DO is used, then other all DOs may have the same name thus rewriting each other in the folder. That can be avoided by identifying interactions when name of the DO is typed. Also in this case, by using the stages that part of the interactions should be marked and for the next after first DOs increasing number (by simulation of pressing the key) should be appended. For example, if the name of the first migrated DO is MOB.pdf, then then interactions before typing M and after typing B should be marked. Furthermore, IWRep should have already some counter by using which for the next DO specific key event will be generated. This will allow to save DOs with different names in one folder.

In such a way, based on the stages, cycles to migrate set of DOs within one EMW can be implemented. This approach can decrease the time of the EMWs, since the time to repeat such operations as emulator run, operating system boot, shutdown of the operating system will not needed to be repeated for each primary object.

4.2.2 Error-handling Mechanism

EMWs should have a mechanism reacting on failures automatically, after a user identified such failure migrations and created IWD corresponding to each error case. To do that, a feature to create stages is crucial.

The following description of the mechanism to deal with failing EMWs can be considered:

Assume user makes large-scale EMWs. The user creates a script to log all information concerning migration into some file. After the EMWs are finished, the user can identify which DOs are migrated and which are not. Further on, the user deals with the DOs that were not migrated.

User makes recording over one of the failed DOs. Whenever error appears (in most cases, as test case Failure analysis shows, errors are unexpected modal windows) the user creates stage *CleanupL*, where L is the error appeared with EMW. The user makes screenshot of the error appeared, serving later on as a SyncPoint, so that it will be possible to the IWRep during migrations using the stage title and exact screenshot of the error to identify the failure. Afterwards, user executes actions to get rid of the error and continues recording; After error is eliminated, user moves the pointer to position (0,0) and creates new stage to identify the end of the *CleanupL* stage.

In the next step,the user imports that stage (marked list of interactions) into the original IWD in corresponding place within the IWD where actually error can occur. IWRep functionality should be changed to jump to the section in IWD to the stage *CleanupL* whenever error occurs. If SyncPoint match in that stage, then actions to get rid of the error will be taken automatically. If SyncPoint in *CleanupL* stage does not match, then IWRep search for the next occurrence of the *CleanupL*. Also in this case as in previous SyncPoint is checked. If no SyncPoint match in the *CleanupL* stage(s), then IWRep goes to the stage *End*, right after the stage CloseApp, and finishes the EMW. User will check how many DOs are migrated time time and if there is some more failure EMWs, then user can repeat the scenario for the new errors. In the end original MW will consist of the number of *CleanupL* stages that correspond to the number of different failures within EMWs and IWRep will automatically deal with that type of errors in the next migrations.

Also use of manual-area SyncPoints where a user can exactly specify the area of the SyncPoint during recording of the MW is preferable. In the future, possibilities of creating them has to be investigated.

This approach is scalable depending on the number of errors. It can definitely improve reliablity of EMWs since it guarantee an expected outcome of interactions from the original MW. Stages in this case, as in the previous section are essential to perform such kind of error-handling.

4.3 Software Design

This section considers the design of the particular features in the IWRec and IWRep (see Chapter 2.1) to achieve the goals described in Introduction. The design meets the requirements from Chapter 3.

4.3.1 Use Cases

Use case diagram modeling functionality of the system in terms of the following actors, and their use cases shown in Fig. 4.1.

The Interactive Workflow Recorder supports two use cases:

1. Input: Pointer movements between two input pointer clicks or SyncPoints;
   Output: Inject optimized pointer movement events;

2. Input: Request to create the stage with the specified name;
   Output: Inject stage specific information;

The Interactive Workflow Replayer supports two use cases, too:

1. Input: Stage specific information;
   Output: Processed stage specific information;

2. Input: Interactions;
   Output: Progress information;

4.3.2 Actors

The potential actors accessing both components are:
Contributor - user that utilizes workflow registration component and IWRec to create IWD for further unattended migration of obsolete DO;

End User - user that works with the migration component or directly with the IWRep and migrates obsolete DOs. End user should specify format of the DO and of the resulting migrated DO, upload and download the DO.

Both components interact with the actor:
Interactive Workflow Description - DO containing a MW, with the description of each interaction that contributor had invoked.

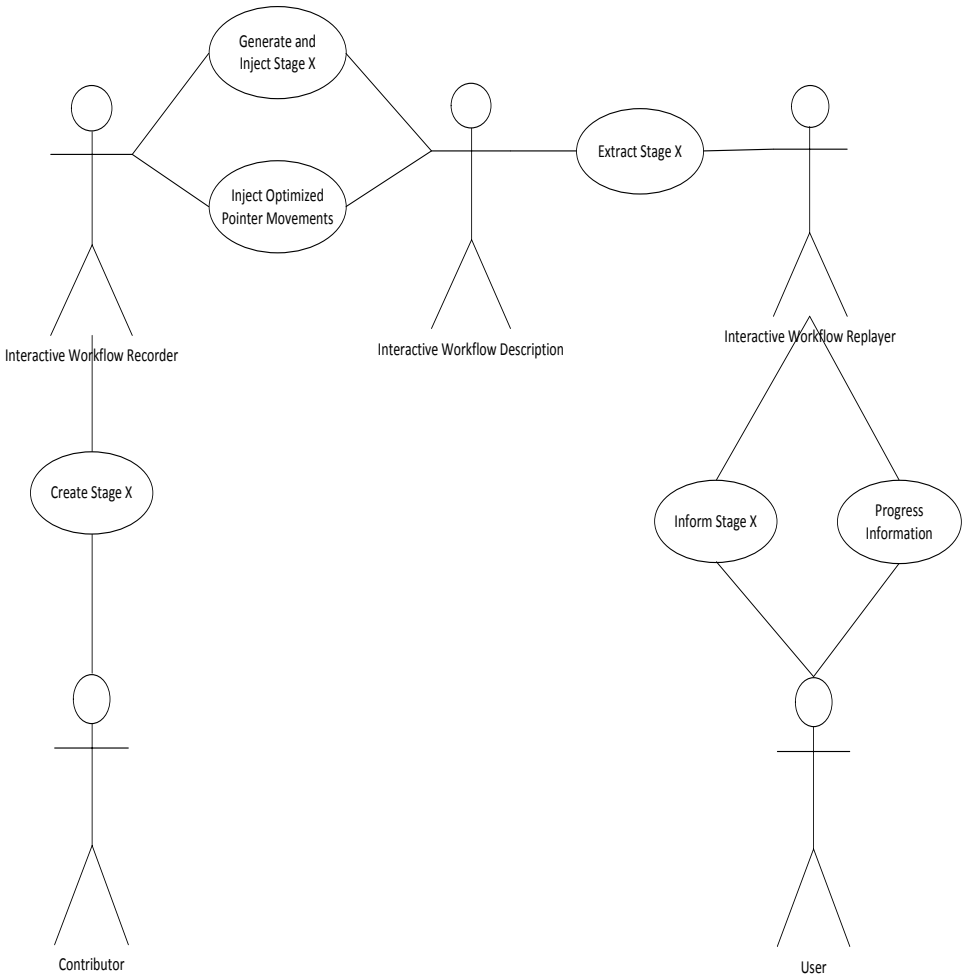


Figure 4.1: Use Case Diagram

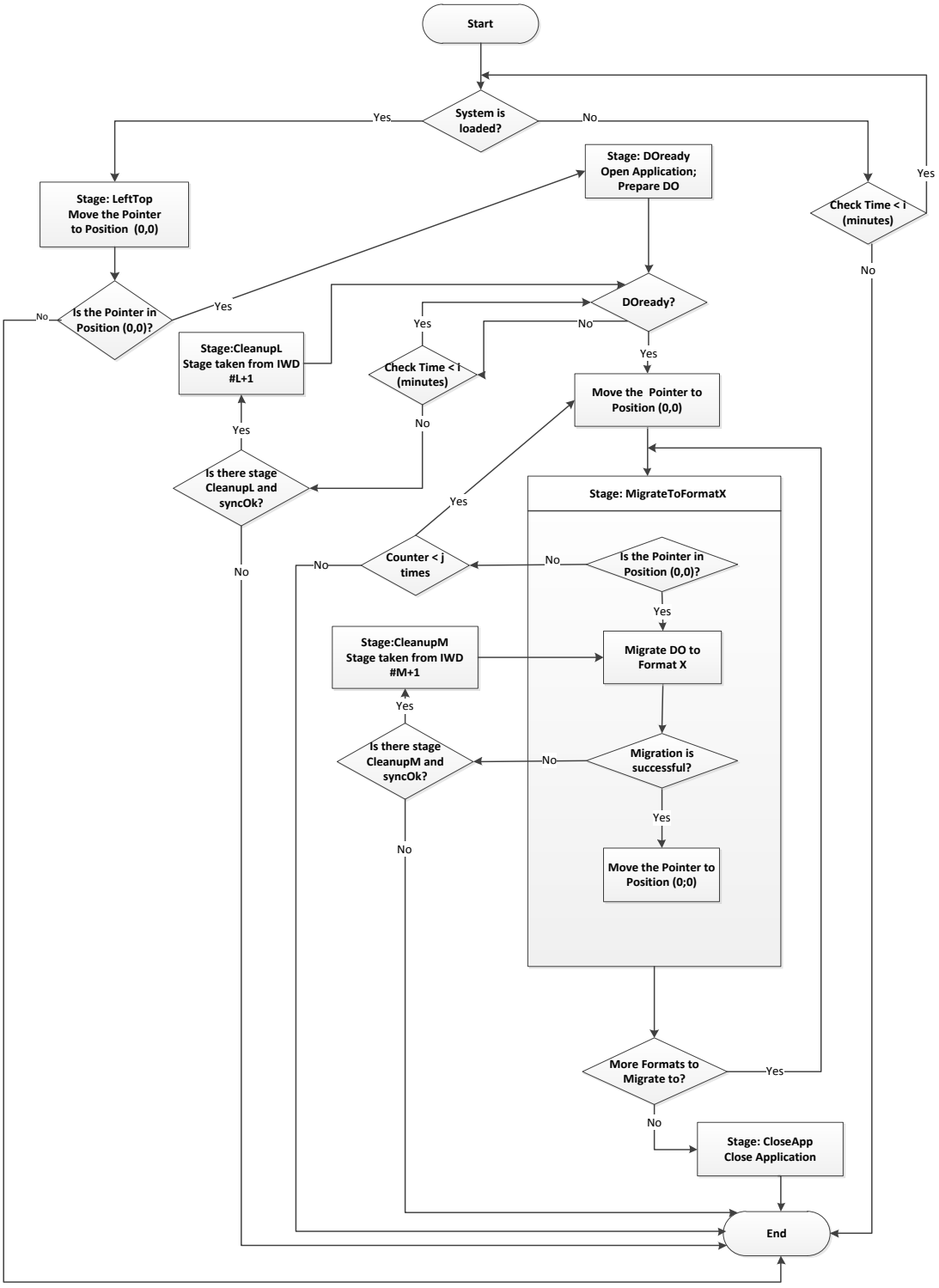


Figure 4.2: Modified Flowchart diagram describing a migration workflow on an abstract level. Combined process – Stage: *MigrateToFormatX* includes other processes and decisions.

5 Implementation

This chapter describes the implementation of the functionality described in the Chapter 4. Furthermore, it describes how well the requirements from the Chapter 3 are met.

5.1 Create Stage Use-Case

The implementation of the use-cases Create Stage X, Generate and Inject Stage X, Extract Stage X, Inform Stage X, and Progress Information meets the requirements: Reliability, Ability to rearrange, Usability, and Integrity. Requirement reliability is met since feature to partition the MWs into the stages along with the abstract description of the MWs made in the previous chapter serves as a base for the error-handling mechanism. Such an error-handling mechanism is necessary to guarantee an expected outcome of each interaction from the original MW. Abstract description of the MWs allows to delete, edit and rearrange certain stages of the MWs by preserving overall integrity of EMWs.

The first use-case provides contributor with specific interface to create stages. Title (name) of the stages should be specified by the contributor depending on the part of the MWs. New stage has unique id - stageID, starting from one and incrementing automatically, as new stage is added.

In order to implement this use-case, two java classes were used - *ButtonPanel.java* and *VncInputRecorder.java*. All classes except described in section SyncPoints Abstraction, are taken from the modified VNCplay version[1].

First class creates interface of the top panel of the VncPlay with several buttons. On that panel – button NewStage is created. Second class "listens" to the button action and whenever it is clicked, invokes pop-up input window to enter the name of the stage. If nothing entered in the input window then no stages are created. The same

[1]VNCplay modifications based on the theses of [3][11][13]

action undertaken when input window is cancelled. Inputted by the contributor text and other parameters as stageID transmitted as the parameters to other method *newStage*, which is considered in the next use-case.

Pseudo code based on the code snippet that is added to the *ButtonPanel.java* class:

```
1 Create newStage button
2 Create stageTitle textField
3 if(recording is true)
4   {
5   display newStage;
6   Add action listener to the newStage
7   create variable stageID equal to 1
8   }
```

Following pseudo code based on the code that is added to the existing *VncInputRecorder.java* class:

```
1  if(there exist events and event equal to newStage)
2   {
3   Block input of events
4   Show input Dialog stageTitle with message("Enter stage title():"
       )
5   if(stageTitle equal to null)
6   skip
7   else if (stageTitle.length greater than 0)
8   {
9   call procedure newStage with arguments stageID, stageTitle
10  Increment stageID
11  }
12  Enable input of events
13 }
```

5.2 Generate and Inject Stage Use-Case

After getting the name of the *Stage*, appropriate to IWD information is generated. That information can be saved in a hashmap to fit IWD formatting and structure.

Timestamp of the stage is calculated by getting first, the last interaction timestamp and adding to that timestamp 40 ms.

Pseudo code to do that is the following:

```
1  procedure newStage(stageID, stageTitle)
2      Convert stageID to String
3      Create HashMap m of generic type String, String
4      Put into m strings "SID" and "stageID"
5      Put into m strings "Title" and "stageTitle"
6      Put into m strings "type" and string "stage"
7
8      Increase the timestamp of previous event by 40
9      Assign it to ts;
10     Register into the output file ts and m
```

5.3 Inject Optimized Pointer Movements Use-Case

Actual MWs contain a great number of pointer movements. Such movements are saved to each few pixels increasing the time of the EMWs and overall load to the guest system. Moreover, they make the MWs unreadable and less editable.

Finally, contributor spends time on thinking which action to undertake in particular situation. These delays are also saved to the MWs making them time-inefficient.

Next implementation of the optimized MWs does not register in the IWD excessive to EMWs pointer movements. Pointer movement is only saved when: it is the first interaction after SyncPoint; when it is the first interaction before and after some mouse click; when this pointer movement is more than some threshold (actual 100 pixels) in comparison to the previous registered pointer movement. Above-mentioned threshold is derived from the test cases Pointer movement distance, Mouse Acceleration, Event Generation Time and Migration Workflow Optimization. It is valid until now on all three OSs tested, but can be easily abstracted by using test cases from the Chapter 3.

This use-case meets the requirements Time and Interactions Optimization, Integrity, Usability and and preserve the Reliability of the EMWs. Not necessary to the EMWs pointer movements will be not registered within the MWs increasing the speed of the EMWs. The user has more time to think on which action to undertake in the MWs

since all delays will be eliminated. User will think more on how to make EMW more reliable (e.g., locating SyncPoints in the positions where screen state changes after each key stroke or mouse click) than the speed with which to record. Time of all interactions are synchronized and is no more user-dependent.

Overall integrity of the MWs is preserved, meaning that the actions the contributor has done during recording will be exactly replayed.

Reliability of the EMWs also increased since the time with which each interaction replayed is now enough to the guest systems (at least, three tested) for processing. Before, IWD contained interactions with the time difference less than the time, necessary to the guest system to process interactions.

Class *VncCanvas.java* "listens" to the interactions invoked by the contributor and send them to the *VncInputRecorder.java* class in order to register them in the IWD. Thus, all analysis and further optimization of the pointer interactions are made in this class.

Following pseudo code describes the optimization that was added to the class *VncCanvas.java*:

```
1  if(listener is not null)
2  {
3   if(list OptimizedMovements is Empty)
4   {
5   add evt into OptimizedMovements
6   Listener process evt as MouseEvent
7   }
8   else if(getButton from evt equal to button1 or to button2 or to
       button3)
9   {add evt into optimizedMovements
10  Listener process evt as MouseEvent
11  }
12  else if(getButton from evt equal to NOBUTTON)
13  Create MouseEvent previousEvent
14  Make previousEvent equal to get Last event from
       OptimizedMovements
15    if(getButton from previousEvent equal to button1 or to button2
          or to button3)
16    {
17    add evt into OptimizedMovements
18    Listener process evt as MouseEvent
19    }
20    else
```

```
21    { if(evt.getX() minus previousEvent.getX() greater than
         OPTIMIZATION_DISTANCE) or (evt.getY() minus previousEvent.
         getY()) greater than OPTIMIZATION_DISTANCE) or (
         previousEvent.getX() minus evt.getX()) greater than
         OPTIMIZATION_DISTANCE)  or (previousEvent.getY() minus evt.
         getY() greater than OPTIMIZATION_DISTANCE))
22      {add evt into OptimizedMovements
23      Listener process evt as MouseEvent
24      }
25    }
26 }
```

VncInputRecorder.java class synchronizes the time of all interactions. Additionally, in the mentioned class pointer movement which corresponds the positions of the mouse click is generated. It is needed to synchronize the last pointer movement position and the position of the next mouse click.

Pseudo code based on the code implementing that, partially shown here:

```
1 procedure mouseEvent(MouseEvent e)
2  {
3  ...
4  if (Recording isPaused) return
5  Make int id equal to getID from e
6  Make int button equal to getButton from e
7  Make boolean clickRelease equal to false
8   ...
9  //MouseMove before each mouse pressed in order to synchronize
      position of the moved mouse and following mouse click
10 if (id equal to MOUSE_PRESSED)
11  {
12   Make  MouseEvent me equal to new MouseEvent(Canvas,
        MOUSE_MOVED, e.getWhen() , 0, e.getX(), e.getY(), 0, false,
         NOBUTTON)
13   Create Map<String, String> em2
14   Make em2 equal to Create HashMap - logEvent(me, "x", me.getX()
        , "y", me.getY(), "button", me.getButton())
15   Make Ts equal to LasEventTimeStamp plus timeIncrease
16   Make LastEventTimeStamp equal to ts
17   em2.put("when", "" plus ts)
18   Log into IWD - log(em2, ts)
19  }
```

```
20 Create Map<String, String> em
21 Make em  equal to logEvent(e, "x", e.getX(), "y", e.getY(), "
      button", e.getButton());
22 //automatic time increase
23 Make ts equal to LastEventTimeStamp  plus timeIncrease;
24 Make LastEventTimeStamp equal to ts;
25 Put into em - ("when", "" plus ts);
26 Log into IWD - log(em, ts);
27 }
```

5.4 Extract Stage Use-Case

This use-case extracts the stage properties from the IWD within the EMWs and displayed to the end user. That is needed for the user to identify at which stage EMW is.

Pseudo code look as following:

```
1 procedure doEvent(Map<String, String> m)
2 {
3   Make String type equal to m.get("type");
4    ...
5   else if(type.equals("stage"))
6   {
7    Make String title equal to m.get("title");
8    print out ("=========stage "concatenate title concatenate "
        starts ==========");
9   }
10 }
```

5.5 Progress Information Use-Case

In order to make the EMWs more user-friendly, progress of the EMWs is to be computed. It is based on all interactions of the MW. When next interaction is processed in the EMW, then the percentage of the EMW increases.

Following pseudo code is based on the code snippet added to *VncInputPlayback.java* class:

```
1  procedure float getTotalRows() throws IOException
2  {
3   Make float totalRowsequal to0
4   Make _fr2  equal to new FileReader(new File(IWD))
5   Make _br2 equal to new BufferedReader(_fr2)
6   while(_br2 is not null )
7   {
8    Increment totalRows
9   }
10  Close _br2
11  Close _fr2
12  return totalRows
13 }
```

5.6 SyncPoints Abstraction

In thesis [11] pattern-matching SyncPoints were mentioned. In our thesis, three classes dealing with the two types of SyncPoints are implemented: abstract class - *SyncPoint.java*, and extending it *PatternSync.java* and BitmapSync.java classes. Functionality of *PatternSync.java* class was taken from the code that was implemented based on [11, ch.3]. Bitmap-matching SyncPoints were extracted and put into BitmapSync.java class.

Such a structure allows to choose one of the SyncPoints during the runtime by the following example command:

```
java VncViewer PORT localhost PORT 5900 autorecord yes
SyncPoint [pattern|bitmap]
```

Pattern-matching SyncPoints complicate comparisons of the SyncPoints by making much more calculations in the replaying phase. However, this class maybe extended later to provide comparisons of patterns such as windows, panels or objects of different sizes, but not distances of two snapshots. That could be used to handle error situations when, for instance, the size of the SyncPoint is needed to be dependent from the objects shapes on the screen.

5.7 Evaluation of Optimized Migration Workflows

In order to conduct an evaluation of our approach, we created two MWs, one of which is recorded by using a not optimized IWRec, that was prior to our studies, and the second one is a optimized IWRec, that were extended, based on this thesis. Both of the MWs consisted of the same number of SyncPoints that were made almost at the same time and positions on the screen of the guest system – Windows 98.

The IWRep version is modified, based on our study, where the value of the instance of the java class Timer is increased from 20 ms to 40 ms. This time increase enables to process all interactions in the guest system (it is valid on all three OSs, tested in Chapter 3) thus making the MW more reliable. Additionally, after appearance of the wallpaper of the guest system, there is additional 10 sec delay, that is made in IWRep to give OS time to load all services and components. In some cases, when there is no such delay, OS was not ready to process interactions even if the wallpaper was displayed on the screen. This delay increases the reliability of the EMWs, since the interactions, that were sent in the beginning of the EMWs will be most probably processed by the guest system. Furthermore, time of the unsuccessful SyncPoint matches, in the following *mismatch threshold*, in both sets are made to be 25 minutes. The container size in both sets of DOs is 500 Mbyte.

In order to check the time optimization of the execution of the optimized (created using optimized IWRec – implemented in our study) and the non-optimized MWs (the MWs that were created with IWRec version, previous to our study) – 50 sample DOs were migrated. In the Fig. 5.1, we can notice the optimization that was made. From 50 DOs all were migrated using both MWs. Time of the one execution of MWs was not mentioned in the Fig. 5.1 because it took more than 20 minutes and could excessevily increase the scale of the time in the figure. Using the non-optimized MW only one execution took 29 min 38 sec 91 ms and another execution using the optimized MW 25 min 52 sec 71 ms. Even in this case, when threshold of mismatches elapsed, the time of the execution of the optimized MW was less than the time of the execution of the non-optimized MW.

In order to understand why both last mentioned executions took significantly more time in comparison with others, the execution of both MWs was repeated with the same DOs that triggered SyncPoint mismatches. This time the DOs that caused SyncPoints mismatches were migrated without any issues. The execution of non-optimized MW over the mentioned DO took 3 min 58 sec 89ms while the execution of the optimized MW took 0 min 33 sec 15 ms. Therefore, the reason of threshold mismatch may be related to the overload of the guest system so that it could not process some interactions or it processed pointer movements wrongly, causing the

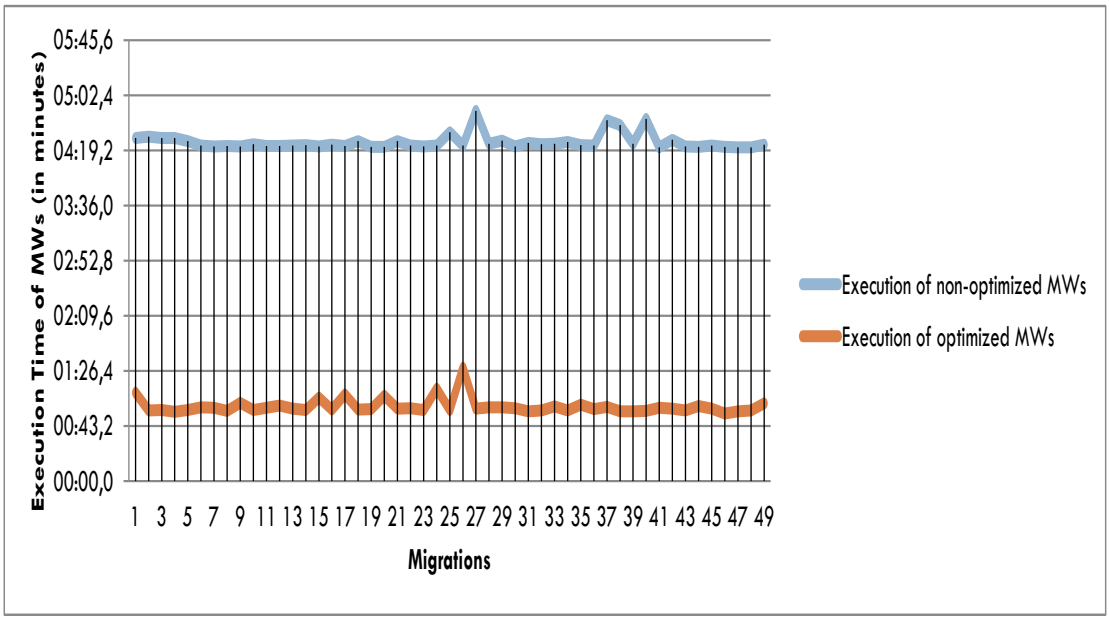


Figure 5.1: Diagram showing comparison of the execution time of the optimized and the non-optimized MWs.

offset. Such an offset was already demonstrated in the test cases *Pointer Movements Processor Load* and *Pointer Movements Disk I/O Load*.

Overall execution time of the non-optimized MWs took 4 hours 6 min 45 sec, while the overall execution time of the optimized MWs took 1 hour 13 min 42 sec. Hence, the execution time minimization rate of the optimized MWs, in the described case, is about 3.35 times.

6 Conclusion

Based on the aim of the thesis, we analyzed the hindrances to create reliable MWs. We identified the set of reasons and their consequences on the reliability of the MWs. The results of this analysis play a significant role to understand the failing EMWs causes and subsequently, allows a user to create more reliable MWs.

A user recording MWs needs an additional time to make the workflow reliable, e.g, to place SyncPoints at necessary positions, to make a decision whether to create the SyncPoint at a particular point in the MW. Our approach does not consider such delays and fully dependent on the number of interactions, that the user invoked. That enables the user to focus more on the reliability of the MWs rather than both characteristics: the recording speed and the reliability.

After the test cases, described in Chapter 3, we were able to optimize MWs. The optimization, resulted in the significant increase of the speed of the EMWs while preserving the reliability of the MWs.

During the analysis, we registered failing EMWs. The main reason was identified and we devised an error-handling mechanism. The mentioned mechanism only can work on all MWs when they are described on an abstract level. Hence, we designed an abstract description of the MW, which shows the functionality how and when to handle error cases.

We examined the work related to our study, and discovered the information on the time reduction of the migration of numerous DOs. Based on that information and the analysis of large number of MWs, using the abstract description of the MW, we designed an approach to migrate set of DOs within a single EMW.

Abstract description of the MW was also used to specify certain stages of the multi-format MWs. These stages upon meeting the conditions, shown in the abstract description can be rearranged or modified so that a user can obtain different format migrations from the single MW.

Since, the abstract description of the MW is crucial for the above-described functionalities, we designed and implemented a feature to create this kind of description

within each MW. This feature allows a user to create stages within MW, so that it can be structured in order to fit the mentioned description.

6.1 Future Work

Basic feature on partitioning the MWs based on the abstract description, was implemented, but supplementary work can be done in the implementation and the evaluation of the designed error-handling mechanism. Particularly, functionality to make jumps of the MW execution to the necessary stage has to be implemented. That can be done by the IWRep. The IWRep should reopen the IWD, and whenever a SyncPoint mismatch takes more than some time threshold (in our case, e.g., 1 min), to find corresponding for error-handling stage in the IWD (we described them as CleanupL(M) stage). If SyncPoint after the mentioned stage does not match, then to search for the next CleanupL stages. If SyncPoint of some of the CleanupL stages matches then proceed with the new opened IWD. In case none of SyncPoints match, the execution should start from the point before it reopened the IWD. Hence, information on the last extracted interaction should be saved e.g., in a hashmap. Continuing the execution that was before the jump is necessary because some DOs may need more time to be migrated, since SyncPoints will not match but the EMW may be correct and after some pause can proceed further. In this case, when the execution jumps back to the last interaction, threshold of the SyncPoint mismatch should be increased in the IWRep automatically (e.g., from 1 min to 60 min).

Moreover, repetition of a certain part of the MW in order to increase the speed of the EMWs, can also be implemented in the future works, based on the design from Chapter 4 and by using already implemented feature to create stages in the MWs. IWRep should get a functionality to get the quantity of DOs that are being migrated. It should be also extended whenever the number of DOs more than one to repeat the stages specified in the Chapter 4. Further mechanism is described in the Chapter 4.

Additional work may be done in several areas to improve usability of the workflows, to make it more interactive even after recording the IWD. For that reason, an editor tool might be developed. The mentioned tool could load original IWD which has machine-readable interactions. Based on the loaded IWD, editor would create simple interactions that user will comprehend.

Following is the snippet of the actual IWD consisting of the different interactions.

```
1 20 id=503 button=0 when=1323771558075 modifiers=0 type=java.awt.
     event.MouseEvent y=482 x=297
```

```
2 40 px306y500=-1 px306y502=-1 px306y501=-1 px306y503=-1 px306y504
     =-1 px306y505=-1 ...
3 -20 id=501 button=1 when=1323771558095 modifiers=16 type=java.awt
     .event.MouseEvent y=482 x=297
4 60 id=502 button=1 when=1323771558155 modifiers=16 type=java.awt.
     event.MouseEvent y=482 x=297
5 40 id=401 when=1323771558275 keycode=40 keychar=65535 modifiers=0
      type=java.awt.event.KeyEvent
6 40 id=402 when=1323771558315 keycode=40 keychar=65535 modifiers=0
      type=java.awt.event.KeyEvent
7 ........
```

Each line can be parsed by the editor and corresponding interactions could be derived. Some events as mouse press and mouse released, key press and key release could be combined to even one event. Derived IWD may look something like the following:

```
1 Pointer Move to (297, 482)
2 SyncPoint of the area of 20 pixels: 1) px306y500=-1 px306y502=-1
     px306y501=-1 ...
3 Mouse Click on (297, 482)
4 Key Stroke - Down Arrow
5 .......
```

Using the software implemented, based on this paper, time difference between interactions and timestamps of them may not be registered in IWD. They could be automatically added during replay.

Such descriptive structure of the MW could ease understanding of the MW by the user and user will be able to make some necessary modifications directly to the MW.

Editor further might be used to generate new IWD from the modified IWD. That will allow the user to create new MWs based on the old ones without recording new MWs.

Bibliography

1 Dirk von Suchodoletz, Klaus Rechert, and Isgandar Valizada. Remote emulation for migration services in a distributed preservation framework. In *Proceedings of the 8th International Conference on Preservation of Digital Objects (iPres 2011)*, pages 158–166, 2011.

2 Isgandar Valizada. Large-scale, transparent format migration system. Master's thesis, Albert-Ludwigs Universität, Freiburg, 2011.

3 Felix Ruzzoli. Ein framework für die zustandbasierte fehlererkennung und -behandlung von interaktiven arbeitabläufen. Master's thesis, Albert-Ludwigs Universität, Freiburg, 2009.

4 Klaus Rechert, Dirk von Suchodoletz, Randolph Welte, Maurice van den Dobbelsteen, Bill Roberts, Jeffrey van der Hoeven, and Jasper Schroder. Novel workflows for abstract handling of complex interaction processes in digital preservation. In *Proceedings of the Sixth International Conference on Preservation of Digital Objects (iPRES09)*, 2009.

5 Volker Uhrig. View-path realizations for obsolete digital objects. Master's thesis, Albert-Ludwigs Universität, Freiburg, 2011.

6 Jacqueline Slats. Emulation: Context and current status, 2003. URL `www.digitaleduurzaamheid.nl`.

7 QEMU. Quick emulator, 2006. URL `http://wiki.qemu.org/Main_Page`.

8 Jeffrey van der Hoeven. Dioscuri: emulator for digital preservation. *D-Lib Magazine*, 13(11/12), 2007. ISSN 1082-9873. URL `http://www.dlib.org/dlib/november07/11inbrief.html#VANDERHOEVEN`.

9 Nickolai Zeldovich and Ramesh Chandra. Interactive performance measurement with vncplay. In *ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 54–64. USENIX Association, Berkeley, CA, USA, 2005.

10 Tristan Richardson. The rfb protocol, 2009. URL http://www.realvnc.com/docs/rfbproto.pdf.

11 Achille Nana Tchayep. Emulatoren-testing für die digitale langzeitarchivierung. Master's thesis, Albert-Ludwigs Universität, Freiburg, 2011.

12 Russell Coker. bonnie++(8) - linux man page, 1999. URL http://www.coker.com.au/bonnie++/readme.html.

13 Mario Philipps. Entwurf und implementierung eines softwarearchivs für die digitale langzeitarchivierung. Master's thesis, Albert-Ludwigs Universität, Freiburg, 2010.

14 Remco Verdegem and Jeffrey van der Hoeven. Emulation: To be or not to be. In *IS&T Conference on Archiving 2006, Ottawa, Canada, May 23-26*, pages 55–60, 2006.

15 Klaus Rechert, Dirk von Suchodoletz, Randolph Welte, Felix Ruzzoli, and Isgandar Valizada. Reliable preservation of interactive environments and workflows. In Mounia Lalmas, Joemon M. Jose, Andreas Rauber, Fabrizio Sebastiani, and Ingo Frommholz, editors, *Research and Advanced Technology for Digital Libraries, 14th European Conference, ECDL 2010, Glasgow, UK, September 6-10, 2010. Proceedings*, volume 6273 of *Lecture Notes in Computer Science*, pages 494–497. Springer, 2010.

16 IDC. International data corporation, 2011. URL http://www.idc.com/.

17 PLANETS. Open planets foundation, 2011. URL http://www.planets-project.eu.

18 Microsoft. Pointer ballistics for windows xp, 2002. URL http://msdn.microsoft.com/en-us/windows/hardware/gg463319.aspx.