2012 IEEE Fifth International Conference on Software Testing, Verification and Validation

Combinatorial Testing on ID3v2 Tags of MP3 Files

Zhiqiang Zhang*, Xiaojian Liu†, Jian Zhang*
*State Key Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences, Beijing, China
{zhangzq,zj}@ios.ac.cn
†Institute of Automation, Shandong Academy of Sciences, Shandong, China
liuxiaojian@nwpu.edu.cn

***Abstract*—In this paper, we study the testing of ID3v2 tags for MP3 files. We construct two combinatorial testing (CT) models for two test goals. One is for testing the audio players' recognition and display of variously encoded text information, and the other is for testing its robustness against bad header and frame sizes. We have conducted experiments on an on-vehicle leisure and entertainment system and a portable MP3 player. We present some experimental results in this paper.**

***Keywords*-combinatorial testing, modeling, ID3v2 tags**

I. Introduction

MP3 (MPEG-1 or MPEG-2 Audio layer III) is an audio format designed by the Moving Picture Experts Group (MPEG), and is one of the most popular audio file formats. MP3 files usually use ID3v2 tags to store information such as title, artist and album. Common functions of MP3 players include playing MP3 files and displaying their audio information.

In this paper, we use combinatorial testing (CT) technique to test MP3 players with respect to the ID3v2 tags of MP3 files. We construct two CT models for two test goals, then generate and execute test cases from the models. Our experiments are performed on an on-vehicle leisure and entertainment system and a portable MP3 player.

CT is a popular software testing technique. In CT, the software under test (SUT) is modeled as a parameterized black-box. The SUT has k parameters $\{v_1, v_2, \ldots, v_k\}$, and the domain of parameter v_i is a finite set of discrete values $\{1, 2, \ldots, s_i\}$ (s_i is called the *level* of v_i). A test case for the SUT is an assignment to each parameter a specific value within its domain. And a test suite is a set of test cases.

In CT, the behavior of the SUT is affected by its k parameters, and failures are caused by the interaction of several parameters. The main objective of CT is to discover these *interaction faults*. Another one is to reduce the test suite size. To achieve these objectives, we usually use a covering array (CA) as the test suite. A $CA(k, t, \{s_1, s_2, \ldots, s_k\})$ is an array of k columns, where any t columns cover all value combinations of the corresponding parameters. Here t is called the strength of the covering array, and $s_1, s_2, \ldots, s_k$ are the parameter levels.

Table I shows an example CA of strength 2 for an online payment system. The SUT has 4 parameters of level 3. It is easy to check that any two of the four columns cover all 3×3 value combinations.

Table I
A Sample Covering Array

Client	Web Server	Payment	Database
Firefox	WebSphere	MasterCard	DB/2
Firefox	.NET	UnionPay	Oracle
Firefox	Apache	Visa	Access
IE	WebSphere	UnionPay	Access
IE	Apache	MasterCard	Oracle
IE	.NET	Visa	DB/2
Opera	WebSphere	Visa	Oracle
Opera	.NET	MasterCard	Access
Opera	Apache	UnionPay	DB/2

Usually, the parameters should satisfy some certain constraints. The constraints may have different representation forms. The constraints we use in this paper look like "`IF <IF-predicate> THEN <THEN-predicate>`". The terms of the IF and THEN predicates are equality ($=$) and inequality ($<>$) relations. The predicates can contain multiple terms joined by `NOT`, `AND` and `OR`.

Sometimes, some parameters may be invalid when several other parameters satisfy a certain condition. Ignoring this fact will cause coverage holes. Suppose we have a test case `(1,2,2,1)` in a CA, and the second parameter is invalid since $v_1 = 1$, then the combination "$v_2 = 2, v_3 = 2$" will not be tested by this test case. [1] proposes some methods to deal with this case. In this paper, we introduce an auxiliary parameter value "#" (representing the parameter is invalid), and add some additional constraints to eliminate coverage holes. More details are shown in our modeling process.

The rest of this paper is organized as follows: we introduce a subset of the ID3v2 tag format in Section II. Then we build parameterized models for the two test goals in Section III. The experiment setup is introduced in Section IV, and the test results are shown in Section V. Finally, we give the conclusions in Section VI.

II. ID3v2 Tag Format and MP3 Frames

In this paper, we focus on a *subset* of the ID3v2 tag format. The extended header, unsychronisation, padding are not

978-0-7695-4670-4/12 $26.00 © 2012 IEEE
DOI 10.1109/ICST.2012.145

taken into consideration. Here we give a brief introduction. More details can be found at [3].

ID3 is a metadata container to store audio file information such as title, artist, album, track number. It is usually used in MP3 files, and it has two unrelated versions: ID3v1 and ID3v2. We discuss the later one in this paper.

An ID3v2 tag is located at the beginning of the file. Typically, it consists of a *header* followed by several *frames* (note that the frames are not MPEG frames). The ID3v2 header format is as follows:

```
ID3v2/file identifier   "ID3"
ID3v2 version           $03 00
ID3v2 flags             %abc00000
ID3v2 size              4 * %0xxxxxxx
```

Here, "`$xx`" and "`%xxxxxxxx`" are respectively the hexadecimal and binary representation of a byte. In our subset of ID3v2, the flags byte is always set as `%00000000`. And the ID3v2 tag size is the size of the complete tag size excluding the header. The most significant bit (bit 7) of each byte of the tag size is set to zero, making 28 bits in total.

An ID3v2 frame header format is as follows:

```
Frame ID   $xx xx xx xx (four characters)
Size       $xx xx xx xx
Flags      $xx xx
```

Here, the frame ID specifies the type of the content of the frame. It is made up of four characters which fall into capital A-Z and 0-9. In our subset of ID3, there are six possible frame ID's: `TIT2` (title), `TPE1` (artist), `TALB` (album), `TRCK` (track number), `TCON` (genre) and `TYER` (year). The frame size indicates the size of the complete frame excluding the frame header. In our subset, the flags bytes are always set as `$00 00`.

In our subset of ID3, all the six frames are text information frames. And the frame content follows the frame header. The content should be like this:

```
Text encoding   $xx
Information     <text string>
```

In our study, we consider the character encodings commonly used in China. The text encoding byte indicates the encoding of the text string. It can be one of the following:

- `$00`: ASCII or GBK. GBK encoding uses 1 or 2 bytes for each character, and is ASCII-compatible.
- `$01`: Unicode (UTF-16). This encoding uses 2 or 4 bytes for each character. It could be little-endian (LE) or big-endian (BE). The text string should be proceeded by a Unicode byte order mark (BOM), which is `$FF FE` for LE and `$FE FF` for BE.
- `$02`: Unicode big-endian (without the Unicode BOM).
- `$03`: UTF-8. The encoding uses 1 to 4 bytes for each character, and is ASCII-compatible.

Besides, MPEG frame format is another thing to consider. Each frame has its own information. One MPEG frame begins with a four-byte header. The first 11 or 12 bits (for different MPEG versions) of the header are always set, so it can be used by the decoder for synchronization. Generally, the MPEG frames are independent of each other, and can be cut freely. However, for MP3 (MPEG Layer III), the frames are sometimes dependent on each other, and cannot be freely cut. In this paper, we use "MP3 frames" for MPEG Layer III frames.

Table II
MODEL FOR TESTING ENCODING RECOGNITION

TIT2	TPE1	TALB	TRCK	TCON	TYER
NONE	NONE	NONE	NONE	NONE	NONE
ASCII	ASCII	ASCII	ASCII	ASCII	ASCII
GBK	GBK	GBK			
ULE	ULE	ULE			
UBE	UBE	UBE			
UBE2	UBE2	UBE2			
UTF-8	UTF-8	UTF-8			

III. MODELING THE INPUT SPACE FOR CT

In audio players, the reading of ID3v2 tags and MP3 playing is performed by different modules. The later one is usually done separately by a MP3 decoder. In our paper, we are testing whether the SUT behaves correctly against different kinds of ID3v2 tags, not the correctness of the MP3 decoder. So we assume that the decoder will always play a sequence of MP3 frames correctly.

In this paper, we have two test goals: the first is to test whether the system can correctly recognize and display variously encoded text information, and the second goal is to test the robustness of the player when facing bad header and frame sizes. Now we build parameterized models for the two goals.

A. Goal I: Encoding Recognition

We have mentioned six encodings discussed in this paper: ASCII, GBK, Unicode LE (ULE), Unicode BE (UBE), Unicode BE with text encoding byte `$02` (UBE2) and UTF-8. It is possible that the frames in one ID3v2 tag are in different encodings. And in the implementations of audio players, it is possible that the encoding of some text information frame will affect the processing of other frames.

Among the six text information frames discussed in this paper, the `TIT2`, `TPE1` and `TALB` frames are often more important to the users. So they have all of the six encodings. We let the rest three frames encoded as ASCII text. Besides, it is possible that some frames are missing. So we build the model for encoding recognition as shown in Table II.

Here a test case is an MP3 file, which has test information frames indicated by the six parameter values. We first generate the six frames in different encodings, then assemble the ID3v2 tag according to the parameters. And finally, we attach a complete MP3 frame sequence.

Table III
MODEL FOR TESTING BAD SIZES

TS	TSD	FS	FSD	FM	ATCHMP3
0KB	0KB	0B	0B	NONE	YES
4KB	1KB	256B	64B	SAFE	NO
64KB	-1KB	1KB	-64B	CUT	
				OVFL	

B. Goal II: Robustness Against Bad Header and Frame Sizes

In Section II, we have shown that ID3v2 headers and frames have `size` bytes. Audio players read these bytes, and then read a segment of bytes according to the size. The audio players may be vulnerable to bad sizes. So the second goal is to test the audio player's robustness against bad sizes. The model is as shown in Table III.

The first parameter `TS` is the ID3v2 tag size indicated by the ID3v2 header. And parameter `TSD` is the difference of the actual tag size compared with `TS`. Parameter `TS` is the frame size indicated by the frame header. And parameter `FSD` is the difference of the actual frame size compared with `FS`. The parameters should satisfy the following constraints: `IF TS=0KB THEN TSD<>-1KB; IF FS=0KB THEN FSD<>-64B`.

Here a test case is an MP3 file, which consists of an ID3v2 header and several frames. The ID3v2 tag and frame sizes follow parameter `TS`, `TSD`, `FS` and `FSD`. In our experiment, the frame ID is set as `TIT2`, and the frame content is filled by character 'A', encoded as ASCII text. The ID3v2 tag is filled by continuous frames.

In some cases, the frames may not be perfectly filled into the header. i.e., the remaining space is not enough to fill a frame. Then parameter `FM` indicates the filling mode for the remaining space. "NONE" means that the remaining space is discarded and will not be filled. "SAFE" means that the remaining space is filled by byte `$00`. "CUT" means that the remaining space will be filled by a partial frame, and the overflowing bytes will be cut and discarded. And "OVFL" means that the overflowing bytes will be kept.

Besides, when `TS=0KB and TSD=0KB`, then no frames will be filled, so `FS`, `FSD` and `FM` will be invalid. To avoid coverage holes, we introduce an invalid value "#" to these parameters, and add the following constraints:

- `IF TS=0KB AND TSD=0KB THEN FS=# AND FSD=# AND FM=#;`
- `IF TS<>0KB OR TSD<>0KB THEN FS<># AND FSD<># AND FM<>#.`

After the remaining space has been filled, parameter `ATCHMP3` indicates whether a sequence of MP3 frames is attached.

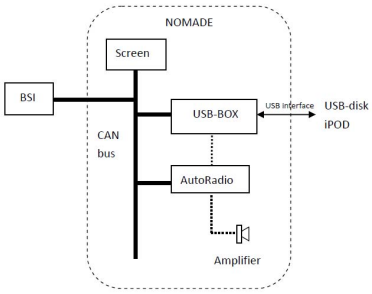

Figure 1. The Architecture of NOMADE

IV. EXPERIMENT SETUP

A. Test Subjects

We conducted experiments on two systems: an on-vehicle leisure and entertainment system and a portable MP3 player. Here we introduce the two systems. The companies' names are omitted in this paper.

1) Test Subject I: The first test subject is an on-vehicle leisure and entertainment system, called "NOMADE". It is developed for automobiles for some well-known companies. The main functionality of the test subject is identifying and playing the audio files stored in an external storage, such as USB disk or iPOD, when they are connected to the NOMADE. The permitted formats of these audios are MP3, WMA, AAC, OGG VORBIS and WAV.

The architecture of NOMADE consists of four parts: Screen, USB-BOX, AutoRadio and Amplifier. They are connected through CAN (Controller Area Network) bus, see Figure 1. When NOMADE is powered on, USB-BOX will read the audio files stored in the external storage connected to it via USB interface. If the stored files have the permitted formats and are valid audios, four kinds of lists (file list, artist list, genre list and playlist) will be constructed, and then these files will be played one-by-one through the Amplifier. The buttons on the AutoRadio can be used to browse, select and play the preferred music. When an audio file is being played, its title and artist information will be displayed on the Screen.

2) Test Subject II: The second test subject is a portable MP3 player. As it is a more professional music player, more music information can be displayed on the screen. This makes it easier for us to observe the test results.

B. Test Generation

We use Microsoft's PICT [2] to generate test suites. We input the models into the tool, and specify the strengths. We set the strength as 2 and 3 for the two test goals respectively. The second test goal has a higher strength since it is more critical and bad sizes are more likely to cause system failure.

The generated test suite size is 57 for test goal I, and 59 for test goal II. For each test goal, all test cases of the test suite are copied into the test subjects.

Table IV
SUMMARIZED RESULT OF TEST GOAL I FOR TEST SUBJECT I

Condition	Result
ALL	ASCII text is correctly displayed.
ALL	UBE2 text is incorrectly displayed.*
TIT2=NONE	Title is displayed as file name.
TPE1=NONE	Artist is displayed as folder name.
TPE1∈{ASCII,UBE2} AND TIT2∈{GBK,ULE,UBE,UTF-8}	Title is not displayed.*
TPE1∉{ASCII,UBE2} AND TIT2∈{GBK,ULE,UBE,UTF-8}	Title is displayed as file name.* Artist is displayed as folder name.*

Table V
SUMMARIZED RESULT OF TEST GOAL I FOR TEST SUBJECT II

Condition	Result
TIT2∈{UBE2,UTF-8}	Title is displayed as file name.*
(TIT2<>NONE AND TPE1∈{UBE2,UTF-8}	Title is displayed as file name.*
OTHERWISE	All information is correctly displayed.

V. TEST RESULTS

A. Test Goal I

1) Test Subject I: After executing the test suite, only two files' information can be correctly displayed. We manually analyze the results and the summarized results are shown in Table IV. In this table, a "*" indicates that the displayed content information is not displayed as expected (i.e. a failure).

Please NOTE that some test cases satisfy multiple conditions in the table. If the corresponding result has conflicts, then those in upper rows have higher priorities.

From the results we can see that the test subject can only display ASCII text correctly. The developers informed us that the test subject was designed to recognize and display Chinese text, however their implementation only support ASCII text.

2) Test Subject II: For the Test Subject II, 24 out of the 57 test cases failed. Again, we analyze the results and the summarized results are shown in Table V.

B. Test Goal II

1) Test Subject I: We first copy all the 59 audio files into the test subject, then an error occurs and all files cannot be played. So we suspect that some files caused this failure. Then we copy one file at a time into the test subject, and see which files can cause the failure. And finally we confirmed that 7 of the 59 test cases can trigger this failure. We analyze the results and the summarized results are shown in Table VI.

2) Test Subject II: The results for Test Subject II are shown in Table VII. We can see that, none of the 59 test cases caused the MP3 player crash. This shows that the MP3 player is robust when facing bad tag and frame sizes.

However, during the testing procedure, we occasionally discovered a defect. When the subject is playing a file with ATCHMP3=NO, it displays that the file is damaged for a few seconds, and then proceeds to the next file. Just between this two moments, it is not possible to play back to the previous file. This defect is not discovered by CT, it's just an occasional discovery.

Table VI
SUMMARIZED RESULT OF TEST GOAL II FOR TEST SUBJECT I

Condition	Result
TS=0KB AND TSD=1KB AND FS<>0KB AND ATCHMP3=NO	The subject encounters a failure, and all audio files copied into the subject cannot be played.*
ATCHMP3=YES	The file can be successfully played.
ATCHMP3=NO	The file is skipped.

Table VII
SUMMARIZED RESULT OF TEST GOAL II FOR TEST SUBJECT II

Condition	Result
ATCHMP3=YES	The file can be successfully played.
ATCHMP3=NO	The player shows that the file is damaged and proceeds to the next file.

C. Benefit of Using CT

In our work, we generate a CA for each of the two models. The test suite sizes are acceptable. For Test Goal I, the exhaustive test suite has $7^3 \times 2^3 = 2744$ test cases, while the CA has only 57 test cases. For Test Goal II, the exhaustive test suite has $3^4 \times 4 \times 2 = 648$ test cases (including test cases that do not satisfy the constraints), while the CA has only 59 test cases.

Our experimental results show that we detected interaction defects using CT. For Test Goal I, Test Subject I&II, the failures are caused by interaction of 1 or 2 parameters. For Test Goal II, Test Subject I, the failures are caused by interaction of 4 parameters.

VI. CONCLUSION

In this paper, we study a subset of ID3v2 tags for MP3 files. We build CT models for testing encoding recognition and robustness against bad tag and frame sizes. We generate test suites from the models, and perform experiments on two subjects. The test results show that some interaction defects are detected. The modeling method used in this paper can be used on other metadata formats having similar structures.

ACKNOWLEDGMENTS

This work is supported in part by the National Natural Science Foundation of China (under grant No. 61070039, 60903049 and 61061130541). The authors would like to thank Xiaoyi Zhang for his participation in performing some experiments.

REFERENCES

[1] Baiqiang Chen, Jun Yan and Jian Zhang. Combinatorial Testing with Shielding Parameters, *Proc. APSEC'10*, 280–289.

[2] J. Czerwonka. Pairwise testing in realworld: Practical extensions to test case generator. *Proc. PNSQC'06*, 419–430.

[3] Official site for ID3, http://www.id3.org/