INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION
ISO/IEC JTC1/SC29/WG11
CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11
MPEG2003/M10015
October 2003, Brisbane, Australia

Source: **ENST**
Status: **For discussion at the 66th MPEG Meeting**
Title: **Issues on externProto coding**
Author: **Jean Le Feuvre, Cyril Concolato, Jean-Claude Dufourd**

Abstract

In this document, the externProto mechanism will be reviewed and it will be shown that its specification in MPEG-4 is not conceptually correct. The issues raised are not theoretical ones but have been serious bottlenecks in some of ENST's content design and creation works.

I Problematic

The externProto coding in BIFS has many restrictions compared to the original intent of the VRML externProto building block. This is mainly due to a bad binary coding of the externProto.

I.1 ExternProto in VRML

In VRML, a world description may refer to prototypes defined outside the file through the externProto element. An externProto consists in 3 major parts:

- the proto name as used in the world
- the proto interface needed to parse the proto instance.
- the location of the entity containing or implementing the proto thus defined.

The external entity providing the proto is not normative in VRML. The location of the externProto is very flexible and allows for explicit addressing. Let's have an example:

```
EXTERNPROTO MYPROTO [
        exposedField SFVec3f size 0 0 0
] "http://myserver/myprotolibs/lib1.wrl#box_proto"
```

This proto will be referenced as MYPROTO in the scene, while its name in the external proto library is box_proto. This allows defining several prototypes in a single file, using this file as a VRML library. With this construction, a scene may use several externProtos located in different libraries without having to repackage the libraries content.

I.2 ExternProto in MPEG-4

MPEG-4 usage of externProto is identical to the VRML usage, except that the externProto shall point to a valid MPEG-4 scene with protos (from now on, the 'proto lib'). Not that the scene is not relevant, only the list of protos is read. However MPEG-4 BIFS uses binary integers to identify prototypes, whether in the scene or in the proto lib. The following problems thus occur:

- the externProto in the scene and the proto in the proto lib shall have the same binary identifier.
- Two externProto in the scene may not have the same binary identifier.
- If two externProtos in the scene are pointing to two distinct proto libs, these proto libs shall not use the same proto identifiers.

We can see from the last item that packaging complex content with heavy usage of externProto is almost impossible since it usually requires repackaging the externProto libs so that proto IDs don't collide, which is not desirable or even not authorized nor feasible,. Moreover, if a proto lib uses an externProto, conflicts are very likely to happen.

Example: an author prepares a complex proto lib for media control using a simple proto lib for graphics nodes:

Lib2 (graphical items)
    ProtoID=1 graphical elements
Lib1 (media controlling):
    ProtoID=1 externproto "lib2"
    ProtoID=0 (media controlling part)

If another author decides to use this library, it MUST use an externProto with protoID=1. If he needs another building block from another proto lib (Lib3) and this proto lib was designed with a single proto ProtoID=0, he cannot use it.

It may also happens that the author wants to use a single proto of a proto lib with many protos, in which case it may need to use much more bits to encode proto IDs than needed.

This is a very important issue we've been facing while developing complex content with high reusability, and is a serious design flaw in the proto coding scheme.

II Solutions

Fixing the above problem can only be done by decorrelating the proto namespace used in proto instantiation and the proto namespace used in extern proto referencing. We propose two solutions to perform this, the first solution being a corrigendum on the existing specification, the second being an amendment.

II.1 Corrigendum Solution

Replace:

9.3.7.2.4.1 Syntax

```
class PROTOcode(isedNodeData protoData) {
bit(1) isExtern;
```

```
if (isExtern) {
MFUrl locations;
} else {
PROTOlist subProtos();
do {
SFNode node(SFWorldNodeType, protoData);
bit(1) moreNodes;
} while (moreNodes);
bit(1) hasROUTEs;
if (hasROUTEs) {
ROUTEs routes();
}
}
}
```

by

9.3.7.2.4.1 Syntax

```
class PROTOcode(isedNodeData protoData) {
bit(1) isExtern;
if (isExtern) {
bit(5) num_bits;
bit(num_bits) externProtoID;
MFUrl locations;
} else {
PROTOlist subProtos();
do {
SFNode node(SFWorldNodeType, protoData);
bit(1) moreNodes;
} while (moreNodes);
bit(1) hasROUTEs;
if (hasROUTEs) {
ROUTEs routes();
}
}
}
```

And update the semantics as follows:

`num_bits`: specifies the number of bits used to encode the externProtoID

`externProtoID`: the protoID of the externProto to use.

Replace

"The EXTERNPROTO code is found in the PROTO contained in this new scene with the same ID in both scenes."

By

"The EXTERNPROTO code is found in the PROTO contained in this new scene with an ID value of `externProtoID`"

Although we usually are reluctant to modifying bitstream syntax, we believe this is an important fix and shouldn't have any impact on existing products since the existing specification makes usage of the externProto quite impossible in real, commercial world.

II.1 Amendment Solution

This solution is roughly the same as above, except that declaration of the externProtoID is made explicit in the scene replace. To achieve this, one of the reserved bit of the BIFSScene structure is used:

Replace:

9.3.7.1.1 Syntax

```
class BIFSScene() {
bit(6) reserved;
bit(1) USENAMES;
PROTOlist protos();
SFNode nodes(SFTopNode);
bit(1) hasROUTEs;
if (hasROUTEs) {
ROUTEs routes();
}
}
```

By

9.3.7.1.1 Syntax

```
class BIFSScene() {
bit(5) reserved;
bit(1) useExternProtoID;
bit(1) USENAMES;
PROTOlist protos();
SFNode nodes(SFTopNode);
bit(1) hasROUTEs;
if (hasROUTEs) {
ROUTEs routes();
}
}
```

The useExternProtoID is then used to signal externProtoID in the externProtoCoding:

9.3.7.2.4.1 Syntax

```
class PROTOcode(isedNodeData protoData) {
bit(1) isExtern;
if (isExtern) {
      if (useExternProtoID) {
            bit(5) num_bits;
            bit(num_bits) externProtoID;
      }
      MFUrl locations;
} else {
      PROTOlist subProtos();
      do {
            SFNode node(SFWorldNodeType, protoData);
            bit(1) moreNodes;
      } while (moreNodes);
      bit(1) hasROUTEs;
      if (hasROUTEs) {
            ROUTEs routes();
      }
}
}
```

Although less elegant than the first solution, this keeps bitstream backward compatibility while allowing efficient externProto usage.

III Conclusion

We have shown in this document that externProto mechanism in MPEG-4 is too restrictive, if not broken, compared to the original intend of the externProto in VRML and have proposed an efficient solution to fix this. We strongly recommend updating the specification as distribution of prototype libraries is an extremely important segment of the MPEG-4 market.