A Generic Extension Mechanism for X3D to Define, Implement and Integrate New First-Class Nodes, Components, and Profiles

Enrico Rukzio
Dresden University of Technology, Department of Computer Science
Heinz-Nixdorf Endowed Chair for Multimedia Technology
01062 Dresden, Germany
enrico.rukzio@inf.tu-dresden.de

Abstract

The current Extensible **3D** (X3D) specification [5] defines a set of nodes, which are grouped in components and profiles. The extension mechanism of X3D allows only the spontaneous creation of new second-class nodes by prototype statements. We think that it is useful to create new first-class nodes on demand, which might be organized into proprietary unregistered components or profiles to extend functionality and to meet specific application needs. This position statement sketches an X3D extension mechanism, which allows the easy Ad Hoc definition, implementation and integration of new first-class nodes, new components and new profiles without a registration process.

1. Existing X3D Extension Mechanisms

The X3D specification defines a rich set of built-in nodes, which are grouped in 24 components and 5 profiles. A component is typically a set of X3D nodes, which may be organized into levels. Each level is defined by a set of nodes. A profile consists of a collection of components and levels of each component, whereby a minimum support criterion for all nodes of the component is defined.

The extension mechanism of the intended International Standard X3D facilitates the definition of new nodes, components, and profiles.

1.1. Creating new X3D Nodes with Prototypes

The *ProtoDeclare* statement allows the definition of a new node type in terms of sub graphs of already defined built-in or prototyped nodes, whereby the declaration of the interface and the implementation of the new node is not separated. The current X3D specification declares that once defined, prototype nodes can be instantiated in the scene graph like built-in nodes. This is not completely true, especially not for the X3D XML encoding. If the prototype is defined in an external X3D document, an author has to write an *ExternProtoDeclare* definition before actually using the new node with the *ProtoInstance* element. Instead of writing a *ProtoInstance* statement, the direct usage of the new node by its name would be desirable. Beside this the interface definition of the new node has to be declared in the external X3D document as well as in X3D scene using the prototype. This redundancy is a typical source of error during the creation of a new application. Further on object-oriented features, such as inheritance, polymorphism, a safe type concept etc. would be desirable. Therefore prototypes are in contrast to the built-in nodes second-class nodes.

1.2. Components and Profiles

X3D may be extended by the creation of a new part of the International Standard or by the registration of new components, new levels within components, and new profiles using the procedures of the ISO International Registration Authority for Graphical Items [2]. These extensions should not be used to modify existing parts of the standard. The current X3D specification describes conceptual, but no syntactical aspects to define new components and new profiles. The process of extending X3D with new components and profiles is still under discussion and development, e.g. there exist no differences between the sections *4.5.2 Extensibility and component registration* and section *4.6.2 Extensibility and profile registration* of the current X3D specification [5].

2. Proposal for an New Extension Mechanism

An author, a programmer, or a company producing rich media and highly interactive X3D applications very often face the problem that the existing language set isn't sufficient. The only possibility to create new reusable and configurable modules is the usage of the prototype mechanism with all its mentioned drawbacks. Referring to this the biggest problem is the fault of creation new built-in nodes, which can really be seen as first-class citizens.

Defining new components is also difficult. The current X3D specification allows this extension only through a new part of the International Standard or through a restrictive registration process. A Java-like extension mechanism is desirable, which builds upon a standard language set. Besides the standard Java distributions, e.g. the Java 2 Platform Standard Edition (J2SE) [3], there exist a huge set of open-source projects, which distribute results mostly as Java Archives (JAR). If these programs reach a mature state and if they are useful for a lot of new programs, they may be integrated into the standard Java distribution.

The following sections sketch an X3D extension mechanism, which allows the definition, implementation, and integration of new first-class nodes, new components, and new profiles. That way a huge set of new practical proven nodes, components, and profiles could be developed, which may be a basis for new standardized X3D elements to support the efficient creation of complex interactive 3D applications.

2.1. A Three Level Architecture to Extend X3D

The following figure illustrates a possible XML architecture for an enhanced X3D extension mechanism, which has an XML grammar, an XML instance, and an implementation for each level.
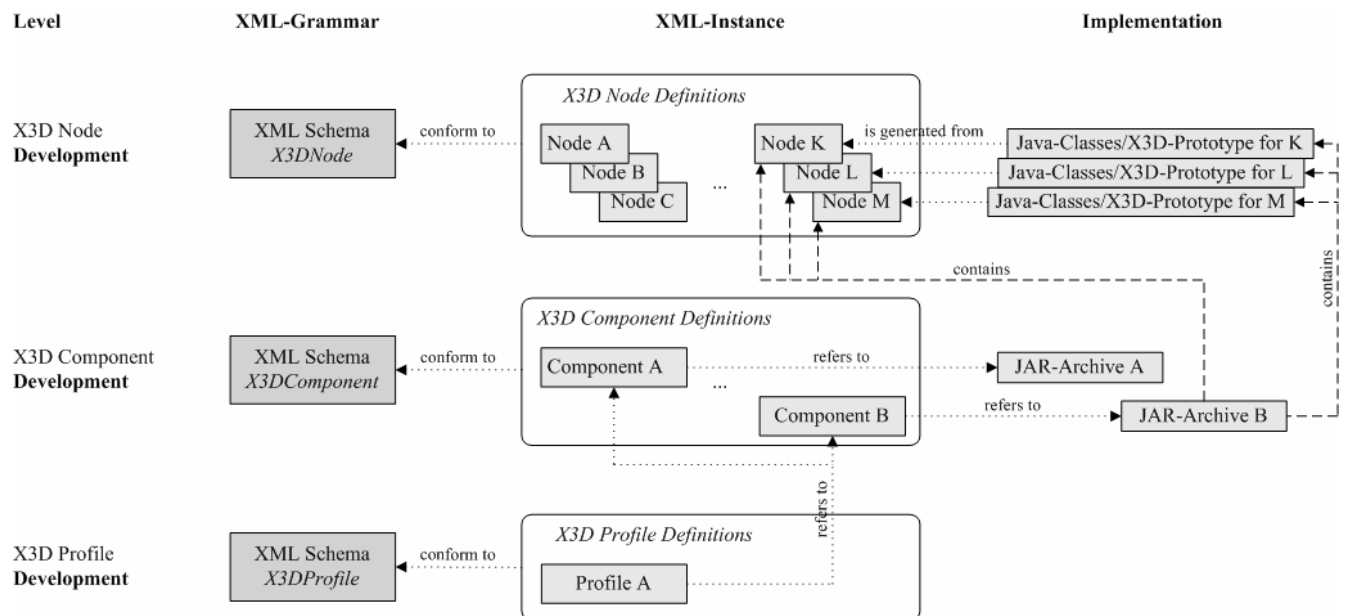

Figure 1. A three level architecture to extend X3D

The X3D node development level allows the declaration of new node types, which conform to the XML Schema *X3DNode*. The implementation, using a preferred programming language, could be generated by the help of XSLT stylesheets. These generated templates have to be implemented and compiled. Figure 1 assumes a possible implementation in Java, which is based on the Java platform scripting reference and the prototype mechanism of VRML97. This approach has been explained in [1] in the context of the creation of new behavior nodes.

```
<X3DNode>
    <Header name="SequentialStateMachine"/>
    <Interface nodeType="public" extends="BaseStateMachine">
        <Fields>
            <Field name="nextState" dataType="Time">
                <ChangeMode configurable="false" receivesEvents="true" generatesEvents="false"/>
            </Field>
            <Field name="previousState" dataType="Time">
                <ChangeMode configurable="false" receivesEvents="true" generatesEvents="false"/>
            </Field>
        </Fields>
    </Interface>
</X3DNode>
```

Figure 2. Declaration of a new X3D node

The example in Figure 2 shows the declaration of the new X3D node type *SequentialStateMachine* that inherits from node type *BaseStateMachine*. Furthermore two fields with a specific name, a data type, and a change mode are defined. A detailed explanation of this new node concept with object-oriented features can be found in [1], wherein it was only be used to define new behavior nodes. This kind of declaration and implementation of new nodes can coexist with the current prototype mechanism.

```
<X3DComponent name="StateMachine">
    <Meta description="The nodes of this component allow the easy definition of state machines."/>
    <Level number="1" url="http://www. SomeCorporation.com/StateMachine1.jar">
        <X3DNode name="BaseStateMachine"/>
        <X3DNode name="SequentialStateMachine"/>
        <X3DNode name="StateMachine"/>
    </Level>
</X3DComponent>
```

Figure 3. Declaration of a new X3D component

Figure 3 shows a declaration of the new X3D component *StateMachine*, which has one level with three nodes. The element *Level* contains an attribute *url*, which refers to the implementation, e.g. a JAR file. This archive contains the XML declaration and the implementation of every node type of the component. This concept is visualized in Figure 1 as the X3D component development level.

The declaration of a new X3D profile can be done by the creation of a document conforming to the XML Schema grammar *X3DProfile*, which refers to existing component levels. This should only be done by an official organization, like the ISO International Registration Authority for Graphical Items. The X3D profile development level in Figure 1 illustrates this mentioned opportunity.

2.2. The Declaration of Extensions in the X3D Scene Description

```
<X3D>
    <head>
        <profile name="Interactive"/>
        <component name="Sound" level="1"/>
        <component name="StateMachine" level="1" url=" http://www. SomeCorporation.com/StateMachine1.jar"/>
        <node name="AnimateTranslation" url="http://www. SomeCorporation.com/AnimateTranslation"/>
    </head>
    <Scene>
        ...
        <AnimateTranslation key="0 1" to="0 0 0, 0 0.05 0" cycleInterval="2.0"/>
        ...
    </Scene>
</X3D>
```

Figure 4. An Example of a possible X3D head statement

Figure 4 shows how profiles and components of the X3D standard and proprietary component levels and node types could be addressed in an X3D file. The *url* attribute of the element *component* refers to the implementation, e.g. JAR archive. This contains the node declarations and implementations. The *url* attribute of the element *node* refers to a directory, which contains the declaration and implementation of this special node type.

Such an X3D file does not conform to the existing X3D grammars [4], such as *x3d-compact.dtd*, *x3d-compromise.dtd*, or *X3DSchema.xsd*. A dynamic grammar is needed representing the language set defined by the statements inside the *head* element. This grammar can be automatically generated during the creation of the scene in the authoring program or by the X3D player, which parses the X3D file.

Figure 5 illustrates the creation of the extended X3D grammar for the X3D file described in figure 4.
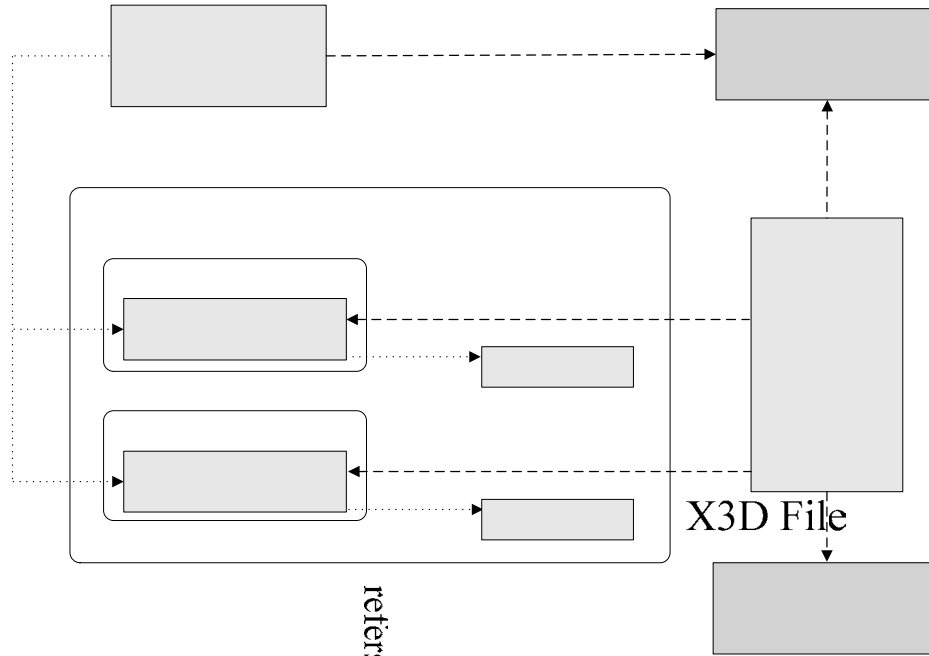

refers to

Figure 5. Dynamic creation of the extended X3D Grammar

Local Library

X3D Node Definitions

AnimateTranslation

X3D Component Definitions

StateMachine

X3D File

has an

has an

refers to

An XSLT stylesheet uses all extension information, which is provided by the proprietary X3D node (*AnimateTranslation*) and X3D component (*StateMachine*) definitions as well as the official standardized X3D grammar to generate an extended X3D grammar, e.g. an XML schema. The X3D file conforms to this extended grammar. Through this mechanism no *ExternProtoDeclare*-statement and no declaration of the node interface are needed. The new nodes can be used as built-in nodes.

The extended X3D grammar can be used by an authoring program and the X3D player to validate the generated or received X3D file.

An X3D player loads the mentioned X3D file, analyzes the elements in the *head* element and loads the implementation of the stated nodes and components from a location provided by the *url* attribute.

3. Conclusion

The sketched approach allows the definition of new first-class nodes, components, and profiles with the help of a three level architecture. Authors of a complex X3D scene are allowed to define their own first-class nodes and components, which can be easily distributed with the application itself. A huge set of proprietary X3D nodes and components can be developed to fulfill the industrial and scientific requirements by means of this decentralized and liberal procedure without any registration process.

4. References

[1] Dachselt, R.; Rukzio, E. 2003. BEHAVIOR3D: An XML-Based Framework for 3D Graphics Behavior. In Proceeding of the 8th International Conference on 3D Web Technology, France.

[2] ISO/IEC 9973 Graphical Items Register, http://jitc.fhu.disa.mil/nitf/graph_reg/welcome.htm

[3] J2SE™ - Java™ 2 Platform, Standard Edition, http://java.sun.com/j2se/

[4] X3D grammars http://www.web3d.org/TaskGroups/x3d/x3d_tags.html

[5] X3D specification, ISO/IEC FCD 19775:200x,
http://www.web3d.org/technicalinfo/specifications/ISO_IEC_19775/index.html