CONTIGRA – Towards a Document-based Approach to 3D Components

Raimund Dachselt
Dresden University of Technology
Department of Computer Science
Heinz-Nixdorf Endowed Chair for Multimedia Technology
D-01062 Dresden
dachselt@inf.tu-dresden.de

Abstract

Even though graphics hardware and 3D technologies are rapidly evolving, the development of three-dimensional applications is still complicated and demands expert knowledge. This paper investigates the component-based development of 3D applications. Current 3D component approaches are classified and a set of technical and authoring requirements for three-dimensional component architectures derived. The CONTIGRA architecture is introduced as a declarative high-level 3D component framework entirely based on XML documents.

Keywords: Component-based Development, 3D Components, 3D User Interfaces, Virtual Environments, Extensible Markup Language (XML), CONTIGRA.

1 Introduction

Continuos improvements in 3D graphics hardware, the increasing availability for consumer platforms and widespread internet use have pushed the development of 3D technologies. In addition to activities centered around X3D [12] as the VRML successor one can notice a variety of proprietary web 3D formats currently being developed. It is unlikely that a certain 3D technology will dominate the field in the near future, not even X3D. Beside divergent industrial strategies this can be attributed to the multitude of application areas for which 3D graphics is being used. On the web we can find 3D objects integrated into HTML-pages, for example in product presentations or as virtual agents. Besides there are also complex virtual environments (VEs) to interact with and walk through like in distributed virtual environments or online games. In addition to that 3D applications with user interfaces consisting of three-dimensional application controls (3D widgets) are being developed to manipulate 3D objects as documents of an application.

The CONTIGRA project pursues the development of *Component-oriented Three-dimensional Interactive Graphical Applications* especially for the latter group. Difficulties in developing three-dimensional applications and VE have motivated this research. Even though many new 3D technologies and tools exist, the development of three-dimensional applications is still demanding work due to format dependencies, missing standards and lack of software engineering support. 3D graphics APIs are flexible and powerful, but not well suited for rapid prototyping or even for non-programmers. Consequently the vision is less or even no programming. 3D exchange formats on the other hand are easier to handle due to their declarative nature, but do not provide enough expressiveness, extensibility and concepts of reuse. Thus the vision are flexible high-level components, which can be easily combined to complex applications. This implies powerful authoring environments. Most of the few actually existing authoring tools do not support interdisciplinary design of VEs. The authoring tools for proprietary 3D formats like Pulse, Viewpoint or Cult3D are all limited to the respective format. The produced 3D scenes or applications are mostly monolithic and restricted with regard to reuse, platform independence, or adaptability.

1.1 Component Technologies

Considering modern software architectures in general there is no doubt about the potential of component-based development, even though it still has to be translated into practice [2]. The usage of component concepts for 3D graphics is a solution as obvious as difficult to achieve. None of the competing technologies like CORBA, DCOM or Enterprise Java Beans (EJB) are tailored to 3D applications on the web, although they all support distributed architectures. By now component technologies are only rarely used in VE systems [3].

Current component technologies are inherently oriented towards code construction using *imperative* programming languages. We call it a *code-centered* view. Another approach – especially with creating graphical user interfaces and multimedia applications – is to design software in a *declarative* way, possibly using authoring tools or user interface builders. The JavaBeans component technology is an example for this approach. Since code can be automatically generated from declarative descriptions this approach is *document-centered*. Compound document architectures like Microsoft OLE, OpenDoc or HTML-pages with embedded objects are representatives. Nevertheless they too were not developed for 3D graphics.

Since most of the 3D applications and VEs employ three-dimensional objects generated by modeling tools with more or less functionality / behavior added to them, we focus on a document-centered view. The mere *programming* of 3D graphics like for example with OpenGL will not be feasible for most of the applications mentioned above. It is rather promising to describe VEs in a declarative fashion, also considering, that borders between (passive) 3D documents and (functional) interface elements are gradually disappearing. Our vision are 3D components like for example 3D widgets, 3D agents, which can be easily configured and composed into VEs and other interactive graphical applications.

This paper is organized as follows. The following section classifies existing 3D component approaches and integrates them into a classification scheme. From that a set of technical and authoring requirements for 3D component architectures is derived in section three. The main part of the paper is dedicated to the introduction of the document-based CONTIGRA approach with its XML grammars. Finally we conclude giving the prospects for future work.

2 Classification of 3D-Component Approaches

Because 3D components are a relatively new field of research, there exist only few approaches and solutions to combine 3D graphics and component technology. The following classification does not claim to be exhaustive, though the main existing projects have been considered. It is a first attempt to systematize existing approaches as a basis for further development. The division into five groups shows only one possible order principle from code-centered to document-centered solutions. After the introduction of the approaches a two-dimensional classification scheme is presented.

2.1 Early Approaches

Already with the development of the *Open Inventor 3D Toolkit* mechanisms for extending node types and creating abstractions to scene graphs were introduced. The Inventor *Node Kits* were an early attempt to 3D components, realized as DLL/DSO in the respective operating systems. *VRML Prototypes* as part of the VRML97 ISO standard are similar to this concept, but based on a declarative document syntax. Both approaches allow to encapsulate parts of the scene graph (the implementation), to attach semantics, and to facilitate reuse in different applications. They are however limited to the respective format and restricted in terms of configuration and distribution.

2.2 Code-centered Approaches

NPSNET-V primarily supports the development of scalable, distributed VEs. A proprietary component system was designed, *Bamboo* [11], where code modules operate in a cross-platform and cross-language manner [3]. With NPSNET-V Java components can be dynamically loaded across a network at runtime. Although XML is used to describe the syntax of network messages in the so called *Dynamic Behavior Protocol*, the NPSNET approach is clearly code-centered.

The *Scene-Graph-As-Bus* approach aims at independent distributed 3D components, which do not adhere to a certain component interface model [15]. The only requirement for 3D applications to communicate is an underlying scene graph API like Open Inventor, Jot or Java3D, which is mapped to a *neutral scene graph* layer. This layer connects different scene graph oriented 3D applications, which themselves do not even know from each other.

2.3 Usage of Existing Component Technologies

The solutions of this category are based on an existing component technology and modify or extend it to integrate 3D graphics or scene graph functionality. Since platform independence and web-enabled technologies are primary goals, the usage of JavaBeans and Java3D suggests itself. A typical example of this approach are *Three-dimensional Beans* [4], which employ these technologies and extend the bean concept with scene graph semantics. The 3D Beanbox editor is used for authoring and assembling 3D Beans. Although this is done in a declarative, graphical way, the solution is still code-centered and dependent on the mentioned technologies.

2.4 Dedicated 3D Component Solutions

These approaches are based on an existing 3D API or format and extend it or integrate it into a proprietary architecture to achieve component functionality. Component interfaces or scene assemblies are often described in separate, mostly XML-based documents. The *i4D architecture* [5] introduces so called *actors, high-level attributes,* and *actions* to modify them, which all together create an abstraction to scene graphs. Components are realized as DLL/DSO and i4D scenes can be described with XML documents. Both API programming and scripting facilities are provided for rapid prototyping. The layered architecture runs on various platforms and 3D APIs.

Smart Virtual Prototypes [8] are based on VRML and Java classes to define simulation components consisting of user interface objects (realized as VRML prototypes), interactor components (on the client side) and virtual components (on the server side), both implemented as Java classes.

2.5 Document-centered Approaches

While the competing component technologies evolved from a code-centered view, meanwhile they all provide description languages for component interfaces. These are XML-based markup languages such as the *Bean Markup Language* (BML), *CORBA Component*

Descriptor, or *EJB Deployment Descriptor*. Based on that development, markup languages seem to be promising for 3D component description and assembly, too.

An interesting approach for 3D graphics is the *Jamal declarative component framework* [6] based on a flexible and expandable *Component Interface Model*. BML is being used for declarative description of component connections. The JamalPlayer parses BML-Scenes and displays them using Java3D. The isomorphisms between VRML-Prototypes, X3D-documents, Java Beans and IDL described in [7] show the possibility to define component interfaces and connections in an abstract way, which can be translated to a concrete implementation later. The CONTIGRA approach described in this paper also belongs to this category.

Though the mentioned solutions have various strengths, they mostly depend on particular platforms, 3D APIs, or component technologies. A few of them have in common to utilize XML documents, for example to describe component interfaces or 3D-scenes. More often a mix of description formats is used, like for example IDL description + data sheet + C header files + short textual description. This impedes a fast development and reuse of 3D components.

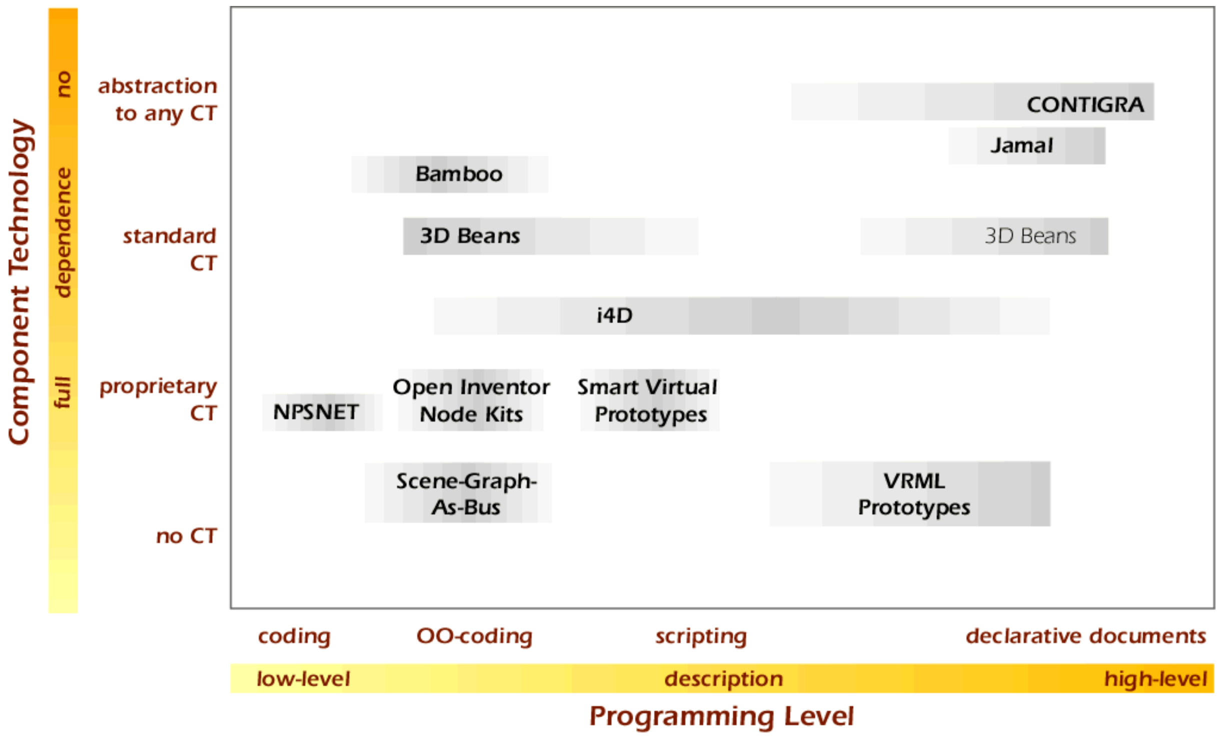

Figure 1: Classification scheme for 3D component approaches

2.6 Classification Scheme

The diagram in figure 1 shows the existing approaches with respect to common established component technologies on the Y-axis and the level of programming on the X-axis. The position of approaches within the scheme is not fixed, possible directions of further development are indicated with the shading of the boxes. Especially the X-axis does not represent a continuos development with regard to the programming level, but rather coding strategies often existing in parallel. Take for example 3D Beans, which were recently extended with a declarative XML syntax describing 3D components. The i4D approach already provides various possibilities to code 3D scenes from object-oriented programming to scripting facilities and XML documents. The Jamal and CONTIGRA approaches provide abstractions to component technologies and specific toolkits and heavily rely on declarative documents. Still, they too need scripting or programming facilities to implement complex functionality. Thus the position of technologies in the diagram has to be considered an approximation only and refers to the core idea of the respective approach.

3 Requirements for a 3D Component Architecture

The analysis of the mentioned 3D component solutions and prior experiences with the development of 3D applications suggest a number of requirements to be realized by an ideal 3D component technology. First of all the same well known characteristics of all component technologies apply to 3D-components, too. Among them are (partly derived from [10]):

- providing abstractions
- hiding implementations
- separating production and deployment
- 3rd party development
- composability

In our opinion components do not necessarily need to exist in binary format. This is an important issue with regard to platform, language, or even component technology independence. Some specific requirements for 3D components shall be outlined in the following sections. They are grouped into rather technical requirements and those demands arising from an authoring point of view.

3.1 Technical Requirements

These requirements are concerned with the interoperability of components, the whole component architecture, the underlying framework, and runtime system.

- *Portability:* independence from specific 3D toolkits, programming languages, component technologies, target platforms, special browsers or plug-in solutions
- *Distribution:* web-enabled & distributed applications
- *Interoperability:* support of a distributed event model and dynamic component loading/binding at runtime
- *Performance:* small size and efficiency, compression, streaming support
- *Adaptation:* to network bandwidth and client platforms (e.g. available I/O devices); to user preferences, languages, cultural differences etc.

3.2 Authoring Requirements

These are requirements for component description, composition, and authoring tools.

- *Abstraction:* high-level abstractions from scene graph semantics (beyond nodes and fields); containment functionality to encapsulate other 3D components
- *Rich component interfaces* for their representation, storage, retrieval / acquisition and deployment:
    - Structured data like offered/required services, explicit dependencies, contract semantics, configurable geometry parts, alternative representations, range of parameters, required performance etc.
    - Meta data for search, distribution, and sales, like for example: version, author, company, license model/payment options, conformance to standards etc.

- Meta data for semantically important information like *may-contain*, *suited for*, or *recommended number of items;* hints on available editors for this particular component
  - Documentation and description of the component
- *Authorability:*
  - support of authoring tools and rapid prototyping
  - support of a declarative syntax, scripting facilities and programming access
  - declarative description of 3D scenes and applications to facilitate interdisciplinary development
  - configuration of typed parameters and design parts / component geometry

Rethinking these requirements mainly from an authoring point of view we conceived the CONTIGRA architecture described in the next section.

4 The CONTIGRA Approach

The aim was to develop a 3D component concept that is largely independent of implementation issues and allows easy, declarative and interdisciplinary authoring of 3D applications. A first step was done with the introduction of an abstract component framework for 3D widgets based on UML/XML [1]. At present the CONTIGRA architecture provides a component framework for 3D graphics, which is entirely based on structured documents describing the component implementation, their interfaces, and assembly/configuration. The heart of the architecture are markup languages based on the Extensible Markup Language (XML) [13], which allow a consistent, declarative description starting from the scene graph level up to complex 3D scenes.

4.1 Advantages of Using XML for 3D Components

There are many reasons for choosing XML as the meta markup of this document-based approach. XML is more than just a data format for structured document interchange on the Web. It also allows the declarative description of program logic, which is important for component wiring and behavior description. Other issues are:

- Platform independence of the format itself
- Standardization and interoperability with other media and internet standards
- Availability of XML-tools, databases, and search engines
- Component descriptions both suitable for automated tools and human readable
- Structured description of meta data for selection, evaluation, and integration of components
- Component documentation can be provided together with the component interface in a homogenous way
- Suitability for document hierarchies, which match the scene graph concept (see X3D)
- Support of name spaces (especially for distributed systems)
- Usage of the Document Object Model (DOM) or XSL Transformations [14] to transform/ evaluate documents

It is especially the last mentioned advantage, that makes the CONTIGRA approach feasible and flexible. The XML-documents describing a 3D application in an independent way are being translated to particular 3D technologies at the latest possible point, deferring the concrete realization to runtime. For performance reasons (binary) applications can also be generated at configuration time, e.g. for stand alone applications.

4.2 CONTIGRA XML Suite

This sections introduces the CONTIGRA markup languages as the core of the approach. These multi-layered XML grammars hierarchically include each other. Figure 2 shows the different levels and associated grammars, documents, and additional files.

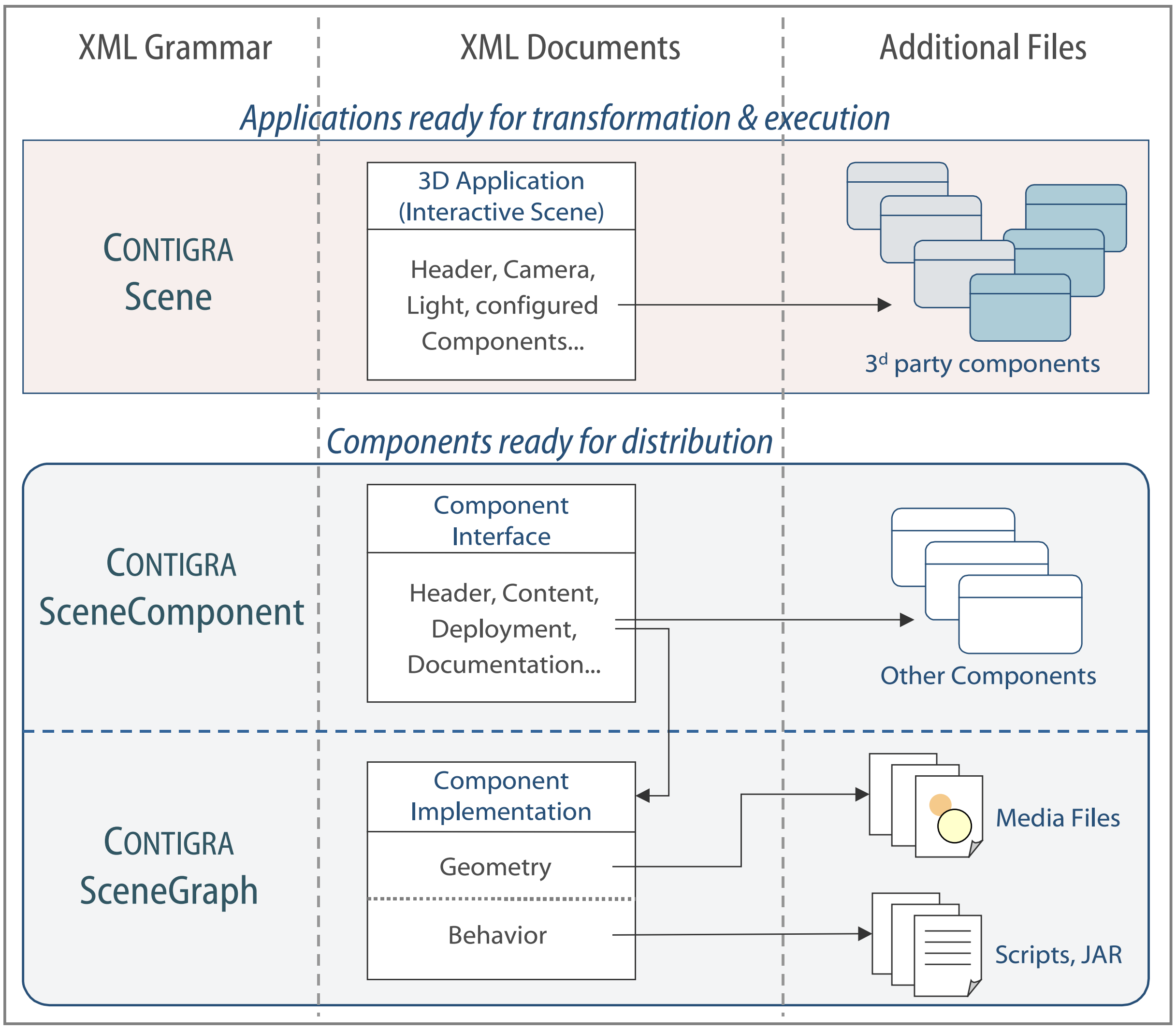

Figure 2: Overview of the CONTIGRA XML-Suite

CONTIGRA SceneGraph

This grammar describes the "implementation" of a 3D component in terms of geometry and behavior. XML is used to code scene graph semantics similar to X3D. Since scene graphs are well established data structures, it is desirable to develop a universal or neutral scene graph format. This can be mapped to actual scene graph based formats like Java3D or VRML. However, it is not very meaningful and also difficult to invent a neutral scene graph format, which would be yet another scene graph format difficult to translate to proprietary 3D formats. That is why we use and extend X3D for the CONTIGRA SceneGraph language. With its profile mechanism X3D allows extensions, which the CONTIGRA SceneGraph grammar takes advantage of. As opposed to X3D we clearly separate between a geometry and behavior graph on this level. In addition to predefined *behavior nodes* (e.g. keyframe animations or interpolators) scripts or other code can be integrated in a homogenous manner using the same XML syntax. Non-standard functionality is coded in separate scripts or code archives (e.g.

JAR) and referenced in the SceneGraph document. In addition to that other (media) files like sounds or textures are also referenced within the documents. The set of geometry and behavior nodes can be extended, domain specific subsets of nodes are possible. The CONTIGRA SceneGraph level provides an abstraction to proprietary 3D formats. Documents coded with it can be easily translated to formats like VRML and displayed in standard viewers.

CONTIGRA SceneComponent

With this component description language the interface of a component is defined. This language can be seen as an analogy to e.g. CORBA Component Descriptors. It provides an encapsulation of the implementation (the SceneGraph part of a component) and thus an abstraction to scene graph semantics. That means, fields of the scene graph can be hidden or combined to form higher level parameters. CONTIGRA SceneComponent documents are separated from the implementation. This way it is easy to store, distribute, or search them and to check them for their suitability in certain application contexts.

The interface documents are subdivided into different sections:

- *header* contains
  - unique id, name
  - short description
  - component type
  - author, company, version and other meta information (see requirements section)
- *interface* contains
  - generalized sensor interface
  - services of the component
  - configurable parts (e.g. geometry) and high-level parameters (as opposed to scene graph fields)
- *deployment* contains
  - requirements (resources) and performance costs for this component
  - component dependencies and component contracts
  - component semantics (e.g. *may contain*, *suited for, …*)
  - license information, adherence to standards, certificates
  - security level
- *content* contains
  - references to CONTIGRA SceneGraph documents (the implementation)
  - references to child components
- *authoring* contains
  - alternative representations (e.g. picture instead of geometry)
  - links to component editors
- *documentation*
  - detailed component description
  - help, instructions for use

Most of the sections are required, others like the *authoring* section are optional.

CONTIGRA Scene

This is the high-level configuration language for component integration. As such it is comparable to the Bean Markup Language (BML). The CONTIGRA Scene grammar also contains different sections:

- *componentHierarchy*

Instances of scene components are configured and arranged in a hierarchical fashion. Geometrical transformations are used to describe the position of 3D components. Apart from this exception the CONTIGRA Scene level provides the highest abstraction to scene graph functionality.

- *componentConnection*

Component cooperation is described using declarative elements of connection oriented programming.

- *sceneDescription*

Typical parameters for 3D scenes or applications are coded with elements for cameras, runtime performance hints, integration with other media or web pages, desired window sizes etc.

The final CONTIGRA Scene document represents a declarative description of a 3D application based on assembled component descriptions. It also serves as an exchange format for 3D authoring tools.

With this approach component design and deployment are separated, declarative authoring is especially well supported, and the resulting 3D applications or VEs are independent of specific 3D toolkits. The result of the authoring process is not a deliverable program, but a complete description of a *potential* executable. The CONTIGRA Scene description can either be transformed into a stand alone application during configuration time (as a result of the authoring process) or being translated into executable code during runtime using the DOM interface or Extensible Stylesheet Language (XSL-T). Java classes or IDL interfaces can be used as the linking element between XML-syntax and source code. At the moment the markup languages are encoded as Document Type Definitions (DTD). However, the XML Schema definition language (XSD) is being considered as a successor due to richer expressions, better data typing, and integration with namespaces.

5 Summary and Future Work

The paper has outlined the importance of components for the structured and reusable design of three-dimensional VEs and applications. Current 3D component approaches were summarized and classified. From that a set of technical and authoring requirements for 3D component architectures were derived. We introduced the consequently document-based CONTIGRA approach along with the XML grammars being at the core of it. Though this declarative approach is promising in terms of authorability, interdisciplinary development, and format independence, it causes problems as well. The high flexibility and abstractions demand powerful translators to proprietary 3D formats. From that follows a lot of work to do, especially considering conceptual differences of various 3D formats or the expressiveness of the respective formats. Another problem to solve will be performance issues. Though not yet proven, it is very likely that performance is not sufficient with parsing XML files at runtime and translating them to other 3D formats. In addition to that another question has to be answered. It is how to elegantly express basic behavior and functionality in a declarative way. The CONTIGRA grammars are still work in progress, likewise the implementation of the runtime architecture and a 3D user interface builder on top of it.

References

1. Braig, A.; Dachselt, R.: „An Abstract Component Framework for Interactive Three-dimensional Graphical Applications“ (German, English abstract), In *Proceedings of the Workshop "Grafiktag 2000" (GI-Annual Conference)*, Berlin, Germany, Sep 2000.
2. Brereton, P., Budgen, D.: „Component-Based Systems: A Classification of Issues“. In *Computer*, Nov. 2000, pp. 54-62.
3. Capps, M., McGregor, D., Brutzman, D., Zyda, M.: „NPSNET-V“. In *IEEE Computer Graphics and Applications*, Vol. 20, No. 5, 2000, pp. 12-15.
4. Dörner, R., Grimm, P.: „Three-dimensional Beans – Creating Web Content Using 3D Components in a 3D Authoring Environment“. In *Proceedings of the Web3D-VRML 2000 Symposium,* Monterey, USA, 2000, pp. 69-74.
5. Geiger,C., Reiman, C., Rosenbach, W.: „Design of Reusable Components for Interactive 3D Environments“. In: *Proceedings of the Workshop on Guiding Users through Interactive Experiences*, Paderborn, Germany, April 2000.
6. Rudolph, M.: „Jamal: Components Frameworks and Extensibility“. http://www.web3d.org/TaskGroups/x3d/lucidActual/jamal/Jamal.html, 1999.
7. Rudolph, M.: „X3D Components“. http://www.web3d.org/TaskGroups/x3d/lucidActual/X3DComponents/X3DComponents.html, 1999.
8. Salmela, M., Kyllönen, H.: „Smart Virtual Prototypes: Distributed 3D Product Simulations for Web based Environments“. In: *Proceedings of the Web3D-VRML 2000 symposium*, 2000, pp. 87-93.
9. Schönhage, B., van Ballegooij, A., Eliens, A.: „3D Gadgets for Business Process Visualization - A Case Study“. In: *Proceedings of the Web3D-VRML 2000 symposium,* 2000, pp. 131-138.
10. Szyperski, C.: „Component Software - Beyond Object oriented Programming“. Addison-Wesley, 1997.
11. Watsen, K.; Zyda, M.: „Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments“. Proceedings of the *IEEE Virtual Reality Annual International Symposium (VRAIS'98)*, Atlanta, Georgia, 1998, pp. 252-259.
12. X3D: The Virtual Reality Modeling Language - International Standard ISO/IEC 14772:200x. http:// www.web3D.org/TaskGroups/x3d/specification/
13. XML http://www.w3.org/XML/ XML Schema, http://www.w3.org/XML/Schema
14. XSL Transformations (XSLT) Version 1.0. W3C Recommendation, http://www.w3.org/TR/xslt, 1999.
15. Zeleznik, B.; Holden., L.; Capps, M.; Abrams, H.; Miller, T.: „Scene-Graph-As-Bus: Collaboration between Heterogeneous Stand-alone 3-D Graphical Applications“, In *Proceedings of Eurographics 2000*, Vol. 19 / Nr. 3.