CONTIGRA

Towards a Document-based Approach to 3D Components

Raimund Dachselt
Dresden University of Technology
Heinz-Nixdorf Endowed Chair for Multimedia Technology

Workshop on Structured Design of Virtual Environments and 3D-Components at the Web3D 2001 Conference in Paderborn / Germany, 19th February 2001

Outline

- Introduction
- Classification of 3D-Component Approaches
- Requirements for a 3D Component Architecture
- The CONTIGRA Approach
- Conclusion & Future Work

Introduction

- Various applications areas & types of 3D VE's:
  - 3D objects integrated into HTML-pages
  - complex virtual environments to interact/walk through
  - 3D applications, 3D-GUI/widgets, 3D objects as documents
- Variety of proprietary web 3D formats, not only X3D
- Many new 3D technologies & tools exist, but development very difficult, need for expert knowledge
  - due to format dependencies, missing standards and lack of SE support
  - 3D graphics APIs are flexible and powerful, but not suited for rapid prototyping, difficult for non-programmers (Vision: less or no coding)
  - 3D exchange formats easier to handle, not enough expressiveness, extensibility and concepts of reuse
  - few authoring tools, often proprietary, no support of interdisciplinary design (Vision: high-level, graphical approach)
  - produced 3D scenes or applications monolithic, reuse difficult, rarely platform independence or adaptability (Vision: reuse, SE support)

→ Potential: component-based development for 3D app.

Introduction

- Component technologies rarely used in 3D systems:
  - CORBA, DCOM or EJB not tailored to 3D applications on the web

  Code-centered view
  - most current component technologies oriented towards code construction using imperative programming languages
- Focus of this work:

  Document-centered view
  - developing GUI's and multimedia applications (with authoring tools, UIB)
  - compound document architectures like Microsoft OLE, OpenDoc or HTML-pages with embedded objects (not made for 3D graphics)
  - 3D objects usually generated by modeling tools and not coded (mere programming of 3D graphics no longer feasible)
  - promising to describe VE's in a declarative fashion, borders between (passive) 3D documents and (functional) interface elements blurred
  - JavaBeans component technology example for this declarative approach

  → Vision: 3D components (3D widgets, agents...) can be easily configured and composed into VE's and interactive 3D graphical applications

Classification of 3D-Component Approaches

Early Approaches

- mechanisms to extend node types and create abstractions to scene graphs
- **Open Inventor Node Kits** (realized as DLL/DSO)
- **VRML Prototypes**, similar concept, based on declarative document syntax

Code-centered Approaches

- **NPSNET-V** supports scalable, distributed VE's (Java) + component system **Bamboo** (cross-platform/language operation of code modules)
- **Scene-Graph-As-Bus**: independent distributed 3D components, no component interface model, scene graph API → neutral scene graph layer

Approaches using existing component technologies

- *based on existing component technologies + 3D graphics / scene graphs*
- *typically JavaBeans and Java3D*
- **Three-dimensional Beans**, employ these technologies and allow authoring of 3D Beans in the 3D Beanbox

Classification of 3D-Component Approaches

Dedicated 3D Component Solutions

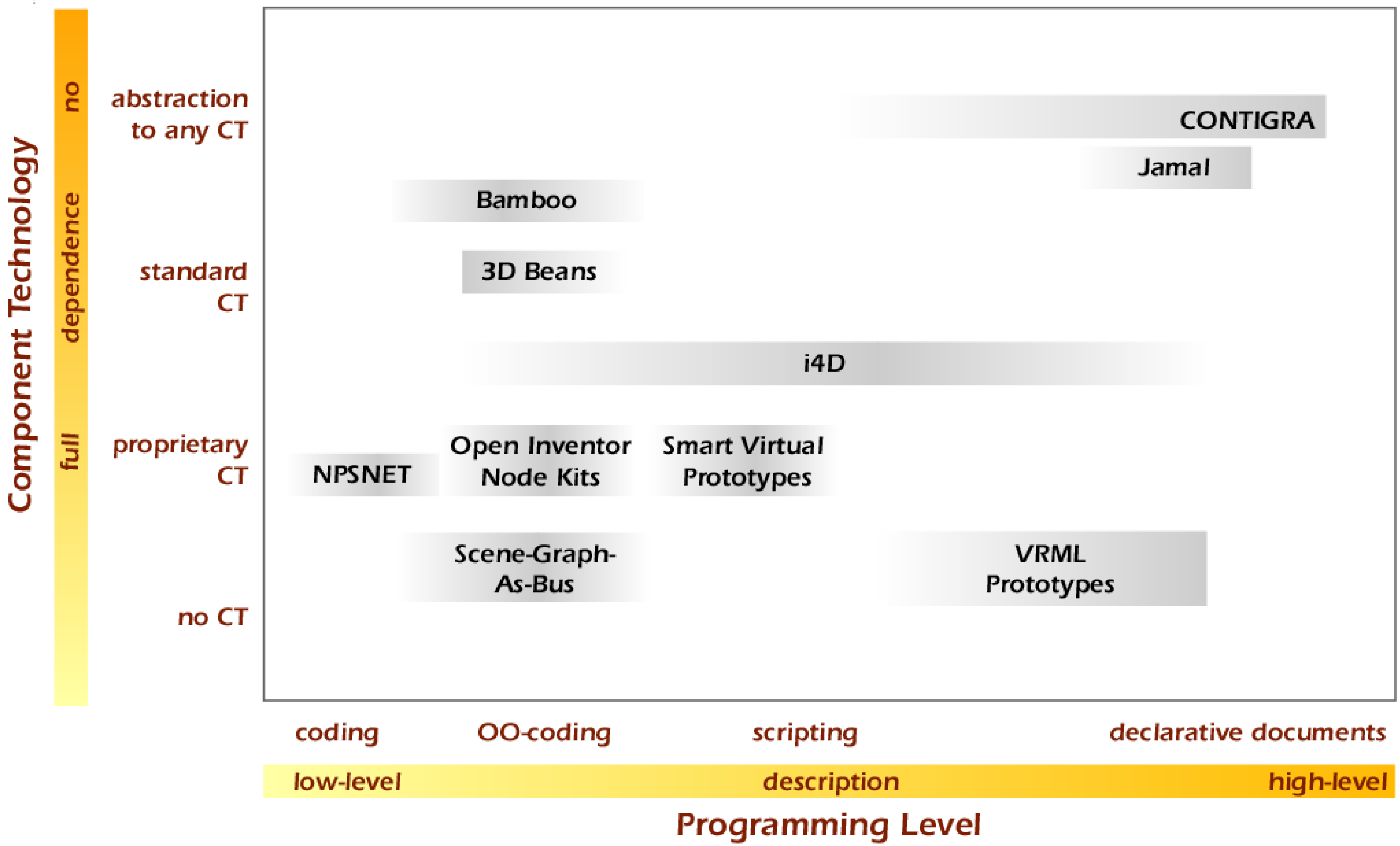
- *based on existing 3D API / format, proprietary extension/integration*
- *Component interfaces / scene assemblies described in XML documents*
- **i4D architecture**: framework for structured design of VR/AR content, high-level descriptions (XML), components (DLL/DSO), layered architecture
- **Smart Virtual Prototypes:** simulation components consisting of UI objects (VRML Prototypes), interactor components (Client side) and virtual components (Server side) as Java classes

Document-centered Approaches

- *XML description languages for component interfaces (BML, CORBA CD, EJB DD)*
- **Jamal declarative component framework** based on a flexible and expandable Component Interface Model (XML), Bean Markup Language (BML) used for declarative description of component connections, Java3D
- isomorphisms between *VRML-Protos, X3D-documents, Java Beans* and *IDL* → abstract definition of component interfaces and connections
- **CONTIGRA** approach described later

Various strengths, dependence on platforms, 3D APIs or CT
Mix of description formats (IDL + data sheet + C header + text)

Classification of 3D-Component Approaches



Other dimensions: Language-dependence and 3D Toolkit/Format - Dependence

Requirements for 3D Component Architectures

- providing abstractions, hiding implementations
- separating production and deployment (reuse), 3rd party development
- composability

Technical Requirements

for component interoperability, architecture, framework, runtime

Portability:

- independence from specific 3D toolkits, programming languages, component technologies, target platforms, special browsers/plug-ins
- late binding through using 1) Java, 2) scripting languages, 3) generalized, abstract document formats

Distribution: web-enabled & distributed applications

Interoperability: distributed event model, dynamic component loading

Performance:

- small size and efficiency, compression, streaming support, (binary format)

Adaptation:

- network bandwidth, client platforms, user preferences, languages, cultures

Requirements for 3D Component Architectures

- Authoring Requirements

  for component description, composition, authoring tools

  Abstraction:
  - high-level, beyond scene graph semantics; component encapsulation

  Rich component interfaces
  - *for representation, storage, retrieval / acquisition and deployment*
  - offered/required services, explicit dependencies, contract semantics, configurable geometry parts, alternative representations etc.
  - meta data for searching, distribution and sales like version, author, company, license model/payment options, conformance to standards etc.
  - meta data for semantically important information like *may-contain, suited for, in context with* or *recommended number of items*;
  - 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 VE's (for interdisciplinary development)
  - configuration of parameters + design parts / component geometry

The CONTIGRA Approach
Overview

Component-oriented Three-dimensional Interactive Graphical Applications

- 3D component concept
  - that is largely independent of implementation issues (Toolkits, CT, ...)
  - allows easy, declarative and interdisciplinary authoring of 3D applications
- first step: introduction of an abstract component framework for 3D widgets based on UML/XML
- CONTIGRA architecture

  provides a component framework for 3D graphics
  - based on structured documents describing,
  - the component implementation,
  - their interfaces and assembly/configuration

  heart of the architecture: markup languages
  - for consistent, declarative description from scene graph level up to complex 3D scenes
  - XML-documents describing a 3D VE are being translated to particular 3D technologies at the latest possible point

The CONTIGRA Approach
Advantages of using XML

- XML

  data format for structured document interchange +
  declarative description of program logic (e.g. behavior)
  Other Advantages:
  - Platform independence of the format itself
  - Standardization and interoperability with other media and internet standards (XHTML, SMIL,…)
  - Availability of XML-tools, databases, search engines
  - Component description suitable for automated tools & human readable
  - Structured description of meta data for selection, evaluation & integration
  - Homogenous component documentation (with interface)
  - Suitability for document hierarchies, match scene graph concept
  - Usage of the Document Object Model (DOM) or XSL T to transform documents

- CONTIGRA markup languages:

  multi-layered XML grammars, hierarchical inclusion

The CONTIGRA Approach
XML Suite

- CONTIGRA *SceneGraph*
  - "implementation" of a 3D component (geometry and behavior)
  - XML coding of scene graph semantics similar to X3D
  - from scene graphs to a universal / neutral scene graph format
  - mapping to actual scene graph based formats (Java3D, VRML…)
  - clear separation between geometry and behavior graph
  - predefined behavior nodes + integration of scripts & other code
  - extensible set of geometry and behavior nodes + subsets of nodes

  → abstraction to proprietary 3D formats
- CONTIGRA *SceneComponent*
  - component description language for component interfaces
  - implementation encapsulation (of the *SceneGraph* part), abstraction to SG's
  - CONTIGRA *SceneComponent* documents separated from implementation → easy storage, distribution, search or suitability checks
  - **Different sections:**
  - *header:* data like id, description or type name + meta information
  - *interface:* generalized sensor interface, configurable parts, attributes and services of the component

The CONTIGRA Approach
XML Suite

- *deployment:* requirements, component dependencies, component semantics, license information
- *content:* references to *SceneGraph* documents and children components
- *authoring:* alternative representations, links to component editors
- *documentation*
- not all sections are required

- CONTIGRA *Scene*
  - high level configuration language for component integration
  - hierarchical assembly of configured scene component instances
  - component cooperation with declarative elements of connection oriented programming
  - also abstraction to scene graph functionality (except transformations)
  - 3D scene/application parameters coded with elements:
    cameras, runtime performance hints, integration with other media or web pages, desired window sizes etc.
  - CONTIGRA *Scene* document represents a declarative description of a 3D application based on assembled component descriptions
  - exchange format for 3D authoring tools

The CONTIGRA Approach

- no deliverable program, but a complete description of a *potential executable*
- CONTIGRA *Scene* description:
  transformed into stand alone application during configuration time
  translated into executable code during runtime (DOM, XSL-T)
- Java classes or IDL interfaces can be used as linking elements (XML / code)
- markup languages currently encoded as Document Type Definitions (DTD), XML Schema definition language (XSD) possible successor

Advantages

- separation of component design and deployment
- support of declarative authoring
- 3D applications and VE's independent of specific 3D toolkits

Difficulties

- development of a neutral/general scene graph format (CONTIGRA *SceneGraph*, at present X3D)
- high flexibility and abstractions demand powerful translators (plenty of work to do!)
- expression of behavior / functionality more complicated without coding

Conclusion and Future Work

- Necessity of structured, reusable design of 3D worlds
- Introduction & classification of current 3D component approaches
- Definition of requirements for 3D component architectures
- Document-based CONTIGRA-approach

- Further improvements of grammars/ XML schemes
- Development of the runtime-framework, prove of concept
- 3D User Interface Builder

- *Looking forward to moderate the working group*