Computer Standards & Interfaces 34 (2012) 93–100

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

A component-oriented framework for experimental computer graphics

Dietrich Kammer [a,*], Jan Wojdziak [a,**], Thomas Ebner [b], Ingmar S. Franke [a], Rainer Groh [a]

[a] Technische Universität Dresden, Faculty of Computer Science Nöthnitzer Str. 46, Dresden, Germany
[b] Fraunhofer Institute for Telecommunications, Heinrich Hertz Institute, Einsteinufer 37, Berlin, Germany

ARTICLE INFO

Article history:
Received 24 November 2009
Accepted 27 May 2011
Available online 26 June 2011

Keywords:
C++
OSGi
Computer graphics
Software components

ABSTRACT

This paper provides a report about a framework that uses a variety of standards. Readers interested in 3D computer graphics or component-oriented technology in C++ will find a report about the integration of various standards by relying on yet another standard for component-oriented software engineering. The highly successful Java standard called Open Services Gateway initiative (OSGi) is employed in a C++ implementation called Open Service Platform.
The application of this standard, which is primarily focused on network-centric software and embedded systems, in the field of real-time 3D computer graphics, provides novel insights into the usability of the OSGi standard.

1. Introduction

Our research group, working in the field of applied visualistics[1] at the Technische Universität Dresden, was challenged with the task of integrating isolated theories and implementations in an extensible software solution.

Much of our work is related to the adaption of techniques previously employed by painters and artists. Traditional paintings from the Renaissance establish a dialog between painter and observer. This is achieved by the composition of a painting, driven by the intentions and point of view of the painter. Computer graphics rely on mathematical models which produce uniform pictures. It is our goal to introduce the human factor into computer graphics by formalizing some of the rules found in the history of paintings and pictures.

It can be proven that unlike the camera model in computer graphics, Renaissance pictures often employ more than one perspective [1]. A wide angle of aperture, frequently found in Renaissance paintings, normally produces distortions of objects located far away from the focal point. In computer graphics or photography, the result for a simple spherical object would be an elliptical shape. Painters treat spherical objects differently than others and preserve their appearance although located at the border of a painting. As a result, their true shape can be identified more easily. Examples apart from spheres are columns of buildings, people, and cupolas [2].

For the desired framework there were numerous requirements and no existing solution could meet all of them. Especially the real-time capabilities are not addressed by standard software in the domain of 3D computer graphics, which is often centered around modelling aspects. Examples are Autodesk's 3ds Max and Maya [3], both offering some means of extending the existing functionality through plugins or scripts. On the other hand, available engines for 3D computer games suffer from restrictions due to their specialized nature.

The intended evaluations based on eye-tracking devices of the theories proposed above, were impossible while using such tools. Eye-trackers are being used in the field of applied cognition in psychology to construct maps of attention [4]. They are used to assess the way visual stimuli are being perceived. Eye-tracking solutions require highly responsive systems, especially when used in connection with real-time exploration of a three-dimensional scene [5].

This paper provides insights into the development process of the presented framework. The focus lies on the identification and integration of standard solutions for the 3D visualization component, the graphical user interface and the extensibility mechanism.

There are already components available based on the framework. Some of them deal with modifications of the rendering pipeline, others offer tools for data visualization. To discuss these components and their specific features is beyond the scope of this paper.

This report discusses related work and describes our framework in more detail. The focal point is the identification and integration of standard solutions for specific areas by using a standard for component based software development.

2. Related work

In this section a brief survey of related work in the field of component based software engineering is given. For the area of 3D

* Corresponding author. Tel.: +49 351/463 37928; fax: +49 351/463 39261.
**Corresponding author.
E-mail addresses: dietrich.kammer@mail.inf.tu-dresden.de (D. Kammer), jan.wojdziak@inf.tu-dresden.de (J. Wojdziak).

[1] The research field of applied visualistics investigates means of producing visualizations of knowledge and data. The goal is to compete with conventional representations that are based on numbers and text.

0920-5489/$ – see front matter © 2011 Published by Elsevier B.V.
doi:10.1016/j.csi.2011.05.010

computer graphics an introduction to the scene graph concept is provided.

2.1. Component based software engineering

Much work has been done in the field of component based software engineering. With component-oriented approaches, the use of readily available software is no longer mandatory. In most cases not all requirements can be met by the programs mentioned above. Another option is programming new software from scratch, which requires much time and effort. Software components should work as independent units of deployment, subject to composition by third parties [6]. The selection of suitable components for a desired software system is thus possible.

For a component standard it is necessary to define three main properties: a component model, a composition technique, and a composition language [7]. The component model defines the nature of a component. It explains the contents and structure as well as the available interfaces. The composition technique explains the way components are connected with each other. The composition language is used to drive and parameterize this process.

Several standards for component systems have emerged that offer different levels of flexibility and ease of use. Most flexible systems like Toolbus [9] employ process algebra based techniques. Components and their composition are defined in a descriptive way, challenging the programmer more than traditional systems. For instance, the Common Object Request Broker (CORBA) started out providing an abstraction for distributed objects. In its latest release a component model is being defined as well as a more powerful composition language [10].

One of the most mature and usable component standards today is the specification of the Open Services Gateway initiative [8]. It is defined using the Java programming language and implements an extension of the module system of Java. Components are called bundles and are usually deployed in a compressed archive file containing all binary files and resources, as well as a bundle specification. This file contains information about the component and declares dependencies on other bundles or resources.

You should be aware of the following standard classification underlying this report. The OSGi specification uses standardized interfaces to compose components. The composition language used in the bundle specification uses a regular grammar according to the Chomsky hierarchy of formal languages. Therefore, it is very easy to be interpreted by the component runtime environment and allows for a simple declaration of properties and dependencies.

The OSGi specification defines a number of layers (Fig. 1), namely the execution environment, the module layer, the life-cycle layer, the service layer, the bundle layer and the security layer. A number of standard services are specified as well. The execution environment is responsible for loading and executing bundles and considering dependencies between them. The module layer defines how bundles are being deployed and packages are shared between bundles. The life-cycle layer enables the installing, starting, stopping, updating, and uninstalling of bundles. For reference, the state diagram in Fig. 2 from the OSGi specification is provided.

The service layer provides a programming model for sharing functionality between bundles. This is essential for achieving interoperability of components. The security layer adds some constraints to the existing Java security model. Actual bundles provided by third party developers are contained in the bundle layer which builds on the previously mentioned layers.

Apart from the official specification, Wütherich et al. [11] recently published the first book about the OSGi standard. It offers a more practical overview, based on the successful application of the OSGi framework within the Eclipse Community [12]. Based on their Equinox framework, the integrated development environment that featured a proprietary extensibilty mechanism for plugins, has been refactored to a rich client platform. The highly dynamic features of the OSGi standard allow the development of flexible systems, built on top of the eclipse framework. Similarities between the framework described here and the Eclipse approach are emphasized throughout the following sections.

2.2. Scene graphs

Graphs are used in many application domains to cope with the complexity of data structures. Many algorithms and theories deal with calculations of sub-groups, reachability of nodes and graph drawing. They can be used both in technical fields like network analysis, as well as social studies like the study of interactions within social groups. For the field of real-time 3D computer graphics the organization of data in form of a scene graph has proved most effective [13].

The earliest implementation is SGI's performer [14]. The scene graph approach is traditionally used in the entertainment industry to build 3D games with a specialized environment. More recently, scene graphs are applied in general purpose environments for 3D visualization [15].

The scene graph is responsible for a number of tasks that otherwise the programmer would have to care about. It is an abstraction of the actual underlying graphics hardware, simplifying the use of available

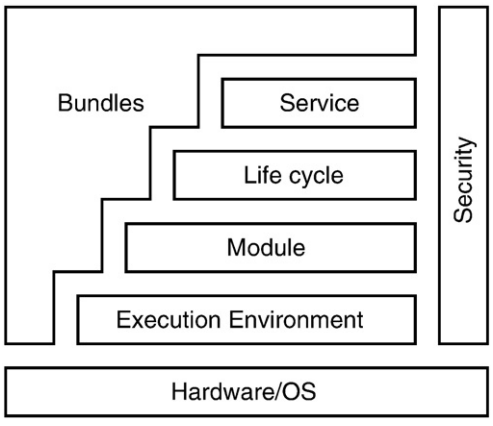

Fig. 1. Layers of the OSGi framework [8].

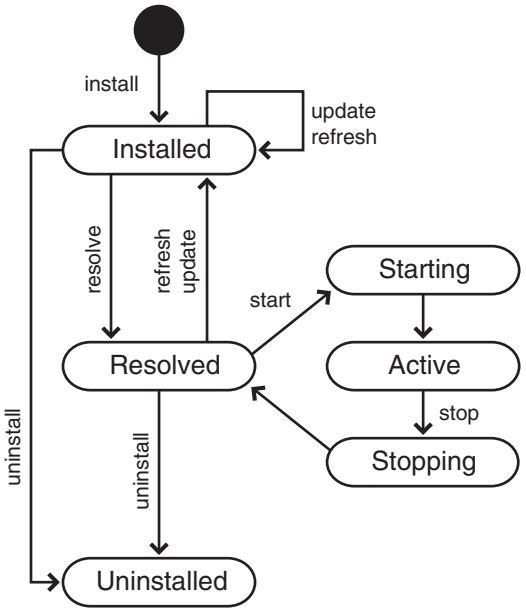

Fig. 2. Life cycle of bundles from OSGi as a state diagram [8].