Distributed Com Application Development Using Visual C 60 With Cdrom Prentice Hall Series On Microsoft Technologies

Distributed COM Application Development Using Visual C++ 6.0: A Deep Dive into the Prentice Hall Series

Developing distributed applications has always been a challenge, requiring careful consideration of network communication, data synchronization, and component interaction. This article delves into the world of distributed component object model (DCOM) application development using Visual C++ 6.0, referencing the invaluable resource: the Prentice Hall series on Microsoft technologies, which often included accompanying CD-ROMs packed with code samples and helpful utilities. We'll explore the intricacies of this now-legacy technology, highlighting its strengths and weaknesses, and providing insights into its practical application. Key areas we will cover include **DCOM architecture**, **Visual C++ 6.0 implementation**, **remote procedure calls (RPC)**, **distributed transaction processing (DTP)**, and **COM+ interoperability**.

Understanding the Fundamentals of DCOM with Visual C++ 6.0

The Prentice Hall series on Microsoft technologies provided a comprehensive guide to building distributed applications using DCOM within the Visual C++ 6.0 environment. This approach leveraged the power of COM (Component Object Model) by extending its capabilities to handle inter-process communication across networks. The CD-ROMs accompanying these texts usually contained sample projects, libraries, and documentation that greatly simplified the learning curve. These resources were essential for understanding the complexities of marshaling, object referencing, and security considerations inherent in distributed applications.

The core concept revolves around creating components – self-contained, reusable software units – that can reside on different machines and communicate seamlessly. Visual C++ 6.0, coupled with the Microsoft COM libraries, provided the tools to create these components, define their interfaces, and manage their interaction. The Prentice Hall books explained the process step-by-step, guiding developers through the intricacies of

interface definition language (IDL), creating proxy and stub objects, and configuring the DCOM infrastructure.

The Role of Remote Procedure Calls (RPC)

A cornerstone of DCOM is the use of Remote Procedure Calls (RPC). The Prentice Hall books would explain how RPC allows a component on one machine to invoke a method on a component located on another machine transparently. This process involves marshaling – converting data into a network-transportable format – and unmarshaling – reconstructing the data on the receiving end. Visual C++ 6.0, with its support for COM and RPC, greatly simplified the implementation of these mechanisms. The CD-ROMs likely contained examples showing how to use the RPC runtime libraries provided by Microsoft.

Distributed Transaction Processing (DTP) in DCOM Applications

For applications requiring data integrity across multiple resources, the Prentice Hall series invariably covered Distributed Transaction Processing (DTP). DTP guarantees that operations affecting multiple distributed components either all succeed or all fail atomically. Visual C++ 6.0, in conjunction with the Microsoft Transaction Server (MTS), enabled developers to build robust and reliable DCOM applications. The books likely included detailed discussions on transaction coordinators, resource managers, and the complexities of distributed transactions within the DCOM framework.

Benefits and Drawbacks of Using Visual C++ 6.0 and DCOM

The primary benefit of using Visual C++ 6.0 with DCOM, as detailed in the Prentice Hall books, was the ability to create robust, scalable, and modular distributed applications. The component-based architecture fostered code reusability and simplified development and maintenance. However, the technology had its limitations. Setting up and configuring the DCOM infrastructure could be complex, requiring a thorough understanding of network security, firewalls, and distributed systems in general.

Furthermore, debugging DCOM applications could prove challenging, demanding specialized tools and a good grasp of the underlying mechanisms. The CD-ROMs associated with the Prentice Hall series potentially offered debugging aids and utilities to help overcome these hurdles. Finally, DCOM’s reliance on RPC and marshaling could introduce performance overhead, especially in high-traffic scenarios.

Practical Implementation Strategies from the Prentice Hall Series

The Prentice Hall series likely followed a structured approach to teaching DCOM development with Visual C++ 6.0. The books probably began with a conceptual overview of distributed systems and the COM architecture, progressing to hands-on exercises using Visual C++ 6.0. The accompanying CD-ROMs undoubtedly played a significant role, providing ready-to-compile code examples that illustrated key concepts such as:

- **IDL Compilation and Interface Definition:** Learning how to define interfaces using IDL and generating proxy and stub code.
- **Component Registration and Deployment:** Understanding the mechanics of registering components in the COM registry and deploying them to different machines.
- **Security Considerations:** Implementing appropriate security measures to protect distributed applications from unauthorized access.
- **Error Handling and Exception Management:** Developing strategies to handle network errors and exceptions gracefully.

The books probably also provided case studies and real-world examples, demonstrating how DCOM can be applied to various application domains.

COM+ Interoperability and Legacy Considerations

While Visual C++ 6.0 and DCOM represent a legacy technology, understanding them offers valuable insight into the evolution of distributed systems. The concepts explored in the Prentice Hall series laid the groundwork for later technologies like COM+, which addressed some of DCOM's limitations. COM+ simplified deployment and management, and introduced features such as transactions and queuing. Therefore, studying DCOM can enhance one's understanding of modern distributed application frameworks.

Conclusion

The Prentice Hall series on Microsoft technologies, especially those focusing on distributed COM application development using Visual C++ 6.0, provided a vital learning resource for developers in the late 1990s and early 2000s. While the technology is now considered legacy, mastering the principles of DCOM remains valuable. The skills acquired in understanding component-based architecture, RPC, and distributed transactions remain highly relevant in today's world of microservices and cloud computing. The focus on structured code, robust error handling, and secure design, emphasized within the series, are timeless principles of software engineering.

FAQ

Q1: Is DCOM still relevant in 2024?

A1: While DCOM is largely superseded by more modern technologies like .NET Remoting and WCF (Windows Communication Foundation), understanding its underlying principles remains valuable. It provides a foundational understanding of distributed application development concepts that remain relevant in current architectures.

Q2: What are the main differences between DCOM and .NET Remoting?

A2: .NET Remoting offers a more streamlined and managed approach to distributed application development compared to DCOM. .NET Remoting leverages the .NET framework's capabilities, simplifying deployment and offering better interoperability with other .NET technologies. DCOM is more tightly integrated with the Windows operating system.

Q3: How does DCOM handle security?

A3: DCOM employs various security mechanisms, including authentication, authorization, and encryption. It leverages Windows security features to control access to distributed components. The Prentice Hall books likely detailed the configuration of DCOM security settings and the importance of properly securing distributed applications.

Q4: What were the common challenges faced when developing DCOM applications?

A4: Common challenges included configuring the DCOM infrastructure, debugging distributed applications, managing network latency, and ensuring data consistency across multiple machines. The CD-ROMs from the Prentice Hall series might have included tools or guidance to mitigate some of these issues.

Q5: Can I still find the Prentice Hall books and CD-ROMs on DCOM and Visual C++ 6.0?

A5: Finding physical copies of the Prentice Hall books and their accompanying CD-ROMs might be difficult. However, you might find some digital copies online through used booksellers or online archives.

Q6: What are some good alternatives to DCOM for building distributed applications today?

A6: Modern alternatives include .NET Remoting, WCF, gRPC, RESTful APIs, and message queues like RabbitMQ or Kafka. The choice depends on specific requirements and the technologies used within the application ecosystem.

Q7: What is the role of IDL in DCOM development?

A7: IDL (Interface Definition Language) is crucial for defining the interfaces of DCOM components. The IDL compiler generates proxy and stub code, which enables communication between the client and the server components. The Prentice Hall books likely detailed the syntax and usage of IDL within the Visual C++ 6.0 development environment.

Q8: How does marshaling work in DCOM?

A8: Marshaling is the process of converting data into a network-transportable format so it can be sent between components located on different machines. Unmarshaling is the reverse process, reconstructing the data on the receiving end. Visual C++ 6.0 and the DCOM runtime libraries handled the complexities of marshaling transparently, as described in the Prentice Hall series.

Diving Deep into Distributed COM Application Development with Visual C++ 6.0

Developing robust distributed applications was a substantial challenge in the late 1990s. The arrival of Component Object Model (COM) and its extension for distributed environments, Distributed COM (DCOM), offered a hopeful solution. This article examines the development of DCOM applications using Microsoft Visual C++ 6.0, drawing heavily on the knowledge provided by the Prentice Hall series on Microsoft technologies, likely accompanied by a helpful CD-ROM featuring illustrations and additional materials. We'll examine the core concepts, real-world implementation strategies, and potential pitfalls to assist you in building your own efficient distributed systems.

Understanding the Fundamentals of DCOM

Before launching into the specifics of Visual C++ 6.0 implementation, let's ground a solid understanding of DCOM. At its heart, DCOM allows objects residing on distinct machines to communicate seamlessly. This is achieved through network procedure calls (RPCs), which hide the intricacies of between-process communication.

The Prentice Hall series likely gave a detailed account of COM's underlying structure, including interfaces, GUIDs (Globally Unique Identifiers), and the role of the processing environment. Visual C++ 6.0 streamlined the creation of COM objects through its advanced ATL (Active Template Library) and MFC (Microsoft Foundation Classes). Understanding these parts is essential for effective DCOM development.

Building DCOM Applications with Visual C++ 6.0

The procedure of creating a DCOM application in Visual C++ 6.0 typically included the following stages:

1. **Defining Interfaces:** This is the basis of any COM object. Interfaces define the methods and properties that the object offers to clients. The Prentice Hall materials would have guided you through the process of defining interfaces using the IDL (Interface Definition Language).

2. **Implementing the Object:** This stage involved writing the actual C++ code that realizes the methods defined in the interface. Visual C++ 6.0's ATL or MFC libraries offered useful classes to streamline this process.

3. **Registering the Object:** Before the DCOM object can be used, it needs to be registered with the operating system. This method made the object accessible to distributed clients. The Prentice Hall guides likely included information on this essential step.

4. **Creating the Client Application:** The client application would then employ the DCOM object through its defined interface. This frequently involved using the `CoCreateInstanceEx` function to create an instance of the networked object.

5. **Handling Security and Transactions:** DCOM applications need to handle security problems, ensuring that only approved clients can use the resources. The Prentice Hall manual likely addressed methods for managing access and ensuring data validity.

Practical Benefits and Challenges

DCOM offered several key advantages. It enabled the building of extensible and maintainable distributed applications. Independent design allowed for easier implementation and rollout. However, building DCOM applications posed some obstacles. Safety was a primary concern, and dealing with network problems demanded careful planning and robust error handling.

Conclusion

Developing distributed COM applications using Visual C++ 6.0, guided by the Prentice Hall series on Microsoft technologies, offered developers a

effective tool for building sophisticated distributed systems. While the technology is now obsolete, understanding the concepts behind DCOM remains important for anyone engaged with current distributed architectures. The expertise gained from this technique forms a strong basis for grasping more current distributed programming paradigms.

Frequently Asked Questions (FAQs)

Q1: What are the key differences between COM and DCOM?

A1: COM is for in-process and local inter-process communication, while DCOM extends COM to enable communication between objects on different machines across a network.

Q2: What are some common security concerns when developing DCOM applications?

A2: Security concerns include unauthorized access, data tampering, and denial-of-service attacks. Proper authentication, authorization, and encryption are crucial.

Q3: Is Visual C++ 6.0 still relevant for learning about DCOM?

A3: While outdated, Visual C++ 6.0 provides a good understanding of the fundamental concepts of COM and DCOM. The core principles remain relevant even with modern technologies.

Q4: Are there modern alternatives to DCOM?

A4: Yes, modern alternatives include technologies like .NET Remoting, WCF (Windows Communication Foundation), and gRPC, which offer improved security, performance, and features.

__from-industry__leading_writers.pdf
https://ns1.fairyland.org/203722/262K49C/PPT/organic_chemistry__david__klein__solutions-manual.pdf
https://ns1.fairyland.org/110926/54X71865L2/PAGE/handbook_of_communication-and-emotion__research-theory_applications__and_contexts.pdf
https://ns1.fairyland.org/wj112609/J45849596W203722/PPT/braun-thermoscan_manual-hm3.pdf
https://ns1.fairyland.org/75U77S5/203722110926/TXT/proform-crosswalk_395__treadmill_manual.pdf