

.NET meets CORBA

.NET-Applikationen in CORBA-Landschaften

Justus Schach

Bestehende Serverapplikationen sind oft das Ergebnis einer umfangreichen, langjährigen Entwicklung. In den meisten Fällen ist ein Redesign weder bezahlbar noch aus technischen Gründen notwendig. An der grafischen Benutzungsoberfläche sieht dies aber oftmals anders aus. An einer Fallstudie wird demonstriert, dass der Weg zu neuen modernen GUIs, hier eine moderne, auf dem .NET-Framework basierende Windowsoberfläche, leicht fällt, wenn als Middleware die Common Object Request Broker Architecture (CORBA) gewählt wurde.

Ausgangssituation

Die Firma REALTECH bietet in der Sparte Softwareprodukte eine Reihe von Monitoring-Lösungen an, die eine ganzheitliche Überwachung von Applikationen, Betriebssystemen und Netzwerkkomponenten rund um die Uhr ermöglichen.

Im Bereich Applikations-Monitoring handelt es sich dabei um mehrere Managementserver, die auf einer gemeinsamen Architektur mit CORBA als Middleware basieren. Diese standardisierte Technologie bietet eine stabile und flexible Kommunikation zwischen den einzelnen Komponenten.

Aufgaben der Managementserver stehen im Kontext von Systemmanagement und sind Teil der „theGuard!“-Produktlinie. Im Applikations-Monitoring sind die beiden wichtigsten Server der „theGuard! ApplicationManager“ (im weiteren kurz tgAM genannt) und der „theGuard! IdentityManager“ (tgIDM). Der tgAM-Server ist die zentrale Serverplattform für eine Systemmanagementlösung, dessen Hauptaufgabe die Überwachung von Applikationen wie SAP, Betriebssysteme, Datenbanken usw. ist. Alle Aufgaben im Bereich des Benutzermanagements werden zentral durch den tgIDM übernommen. Dabei werden Benutzer und Rollen sowie deren Privilegien auf Funktionen des tgAM-Servers verwaltet. Beide Server besitzen fest definierte CORBA-Schnittstellen, die es diversen Clientapplikationen ermöglicht, auf sie zuzugreifen.

In beiden Fällen sind mehrere Jahre an Entwicklungsarbeit investiert worden. Ein weiterer wichtiger Aspekt im Bereich Systemmanagement ist die Stabilität der Lösung selbst. So wird von einem Managementserver erwartet, dass dieser im Dauerbetrieb als ausfallsicher gilt. Auch aus diesen Gründen ist ersichtlich, dass Änderungen oder gar ein Redesign der Serverkomponenten als äußerst kritisch einzustufen und demzufolge zu vermeiden sind.

Etwas anders sieht es im Frontendbereich aus. Hier muss schnell auf marktrelevante Ansprüche reagiert werden. Optik und Anwenderfreundlichkeit spielen eine immer größer werdende Rolle, wobei die hierfür nötigen GUI-Bibliotheken auch eine schnellere und kostengünstigere Entwicklung bieten. Es müssen neue Technologien eingeführt werden, die diesen Ansprüchen genügen, zumal auch die kommenden Generationen von spezialisierten GUI-Entwicklern immer mehr von diesen abhängig werden. Umgekehrt ist die Konfrontation eines auf GUI-Programmierung ausgerichteten Entwicklers mit Kommunikationsdetails zwischen Client- und Serveranwendung zu vermeiden.

In unserem Fall fiel die Entscheidung, die Cliententwicklung auf Windows auszurichten, wobei das .NET-Framework



von Microsoft die bisher verwendeten Microsoft Foundation Classes (MFC) als GUI-Framework ablöst. Damit konnten die meisten der folgenden Architekturziele erreicht werden:

- ▼ direkte Verwendbarkeit der bestehenden CORBA-Infrastruktur,
- ▼ moderne Windowsoberfläche,
- ▼ Kapselung clientseitiger Applikationslogik,
- ▼ einfache Implementierung von Webservices,
- ▼ Internationalisierung,
- ▼ Einsatz von Unit-Testverfahren sollen die Qualitätssicherung erleichtern.

Bisheriger Systemaufbau

Der Aufbau des tgAM/tgIDM-Servers entspricht einer klassischen Mehrschichten-Architektur, bestehend aus drei Schichten, die sich physisch auf mehreren Rechnersystemen verteilen können:

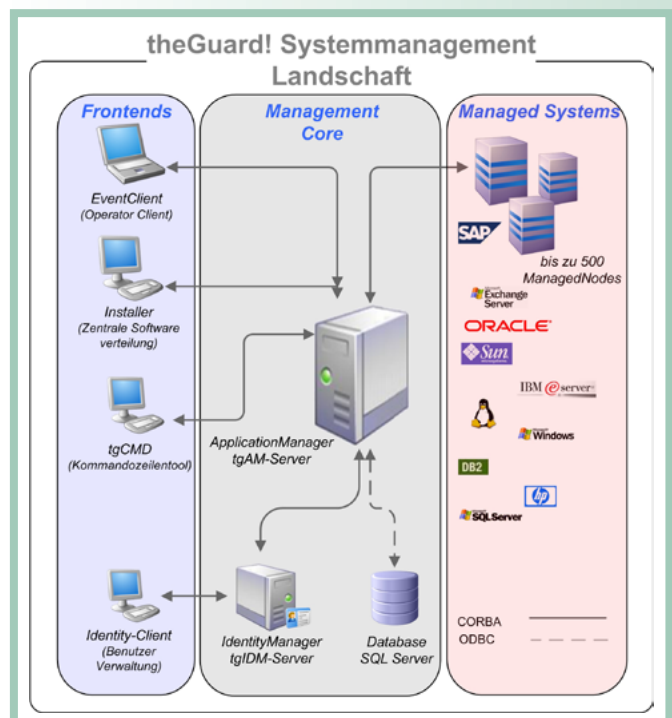


Abb. 1: theGuard! Systemmanagement-Landschaft



- ▼ Frontendapplikationen
(GUI/Präsentationsschicht; C++; CORBA; MFC)
- ▼ tgAM/tgIDM Core Server (Zwischenschicht; C++; CORBA)
- ▼ tgDB (Datenbankschicht, SQL-Server Instance)

Eine der Hauptaufgaben des tgAM als Systemmanagement-Werkzeug ist die Überwachung anderer Computersysteme. Hierbei steht der Server mit bis zu 500 Rechnern in Verbindung. Aufgrund der Heterogenität der Rechnerlandschaften im Serverbereich wurde CORBA als Middleware ausgewählt. Zum Einsatz kam die ORB-Implementierung ORBACUS der Firma IONA [Orbacus]. Auf der Visualisierungsebene wird bis heute im Wesentlichen auf Microsoft Windows gesetzt (s. Abb. 1).

Das neue Ziel

Um die beschriebenen Architekturziele zu erreichen, wurde ein eigenes Framework entworfen, welches neben dem Großteil der Applikationslogik auch sämtliche Kommunikationsdetails zum Managementserver kapselt. Das als *AppCon Access Objects* (ACAO) bezeichnete Framework soll sich clientseitig als Schicht zwischen Server und Präsentationsebene einfügen [PoSa00].

Das aktuell größte Problem ist, dass Entwicklungen und Änderungen am Frontend immer schneller durchgeführt werden müssen. Auch muss eine grafische Oberfläche den aktuellen ergonomischen Richtlinien entsprechen. Zu guter Letzt geht es hier auch um optische Beweggründe, ein Frontend kontinuierlich zu optimieren. Diese Flexibilität kann mit der bisher verwendeten Kombination aus C++ und MFC nicht mehr erreicht werden. Der Einsatz des .NET-Frameworks und Windows Forms bzw. zukünftig auch *Windows Presentation Foundation* (WPF) bietet Lösungen und neue Chancen, den Entwicklungszyklus im Bereich der grafischen Oberflächenentwicklung zu verbessern. Diese neue Generation einer GUI-Entwicklungsumgebung setzt allerdings eine der bei .NET üblichen Programmiersprachen voraus. Es muss daher nach Wegen gesucht werden, aus der .NET-Welt heraus mit einem CORBA-Server zu kommunizieren.

Die Qual der Wahl

Der Evaluierungsprozess für eine solche Integrationstechnik schloss neben kommerziellen Produkten auch Open-Source-Projekte ein (siehe auch [AmRo05]).

Aufgrund dieser Untersuchung fiel die Wahl auf das IIOP.NET-Projekt. Dieses Projekt entstand aus einer Diplomarbeit von Dominic Ullmann an der ETH Zürich und wird von der Firma ELCA, Schweiz weiterentwickelt. Als Open-Source-Projekt ist es bei SourceForge [IiopSf] erhältlich und unterliegt der LGPL.

Dieses Projekt hat gegenüber anderen im SourceForge geführten IIOP/.NET-Interoperabilitäts-Projekten den Vorteil, dass es nahezu vollständig implementiert ist und vor allem „lebendig“ wirkt. Gerade die sehr wichtigen technischen Voraussetzungen wie die Unterstützung von problematischen Datentypen wie Wide-Strings und ANYs wurden umgesetzt. Ein weiteres wichtiges Leistungsmerkmal ist die Möglichkeit, Clientcallbacks zu implementieren. In diesem Fall ist der Client ebenfalls ein CORBA-Server und stellt umgekehrt dem Managementserver Interfaces zur Verfügung. Tests in Bezug auf Datendurchsatz und Speicherverbrauch fielen ebenfalls äußerst positiv aus.

Die Entscheidung fiel nicht leicht, immerhin soll die ausgewählte Bibliothek für die nächsten Jahre Grundlage für alle clientseitigen Anwendungen bilden. Es gilt zum einen, technische Dinge wie die Kompatibilität gegenüber dem bereits verwendeten CORBA-ORB zu gewährleisten, aber auch die Frage nach der

Zukunft einer solchen Bibliothek muss gesichert sein. Neben den bekannten Vorteilen einer Open-Source-Bibliothek muss nicht auf professionelle Hilfe verzichtet werden. Ein Blick in die Projektdokumentation bietet dazu entsprechende Informationen.

Neue Aufgabenverteilung

Ein großer Teil der benutzerbezogenen Logik befindet sich auf der Clientseite. Diese Funktionalität fließt in das ACAO-Framework ein und steht somit allen potentiellen Clientanwendungen zur Verfügung.

Zusammenfassend soll die Einführung des ACAO-Frameworks die bisherigen Probleme entschärfen und neue Möglichkeiten schaffen:

- ▼ Cliententwickler arbeiten direkt mit einem .NET-basierenden Framework, welches im Assembly *RTAppConAm.dll* bzw. *RTAppConIdm.dll* implementiert ist.
- ▼ Das GUI-Team benötigt kein CORBA-Programmier-Know-how, da die Kommunikationsebene vollkommen durch das ACAO-Framework abstrahiert wird.
- ▼ Bisherige Clientlogik wird im ACAO-Framework implementiert und kann bei Bedarf in den Code zukünftiger Serverversionen verlagert werden, ohne dass die davon abhängige GUI-Applikation davon betroffen ist.
- ▼ Sehr hohe Produktivität in Bezug auf GUI-Design gegenüber der bisherigen MFC.
- ▼ Hohe Motivation im Team für den Einsatz dieser neuen Technologie.
- ▼ „Look and Feel“ entspricht automatisch dem aktuellen Windows-Standard.
- ▼ Einsatz einer zukunftsweisenden Technologie.
- ▼ Es steht eine breite Palette von .NET-GUI-Bibliotheken zur Verfügung.
- ▼ Die einfache Implementierbarkeit eines Webservices stellt die Möglichkeit in Aussicht, Web-Frontends zu realisieren. Natürlich sind mit dem Einsatz von .NET auch Nachteile und Risiken verbunden:

- ▼ Die so erstellten Windows Forms GUIs sind aktuell nur für Windows-Rechner mit Microsofts .NET-Framework einsetzbar.
- ▼ Mit dem Einsatz von C# wird eine eher proprietäre Programmiersprache eingesetzt (C# ist bei Ecma International [Ecma] standardisiert, die Verbreitung auf verschiedene Rechnerplattformen ist allerdings mit Sprachen wie z. B. C++ oder Java nicht zu vergleichen).
- ▼ Wird der .NET-Code als MSIL-Code ausgeliefert, besteht die Gefahr des Code-Reengineering.
- ▼ Erste Versuche haben gezeigt, dass die Verwendung des „Global Assembly Cache“ (GAC) als zentrale Assembly-Abfrage näher untersucht werden muss. Speziell auf Entwicklermaschinen kommt es zu Kollisionen zwischen Assemblys, die per *Microsoft Installer* (MSI) im GAC installiert wurden, und den Assemblys, an denen ein Entwickler arbeitet.

Wenig technisch, aber umso problematischer ist auch der Umstand, dass die Neuentwicklung der bestehenden Frontendapplikationen nicht von heute auf morgen zu erledigen ist. Es muss daher mit einer Übergangszeit gerechnet werden, wonach Clientapplikationen der „alten“ Art parallel zu den neueren .NET-basierenden Anwendungen funktionieren müssen.

Aus diesen Anforderungen und den Möglichkeiten des .NET-Frameworks in Verbindung mit der IIOP.NET-Bibliothek lässt sich relativ einfach eine Architektur bilden, die die „alte“ Welt mit der neuen Welt verbindet, ohne dabei die Kompatibilität zu den bisherigen Anwendungen zu verlieren.

Darüber hinaus stehen weitergehende Möglichkeiten zur Verfügung, um zukunftsweisende Technologien wie z. B. Webservices zu implementieren. Mit Hilfe dieser Webservice-An-

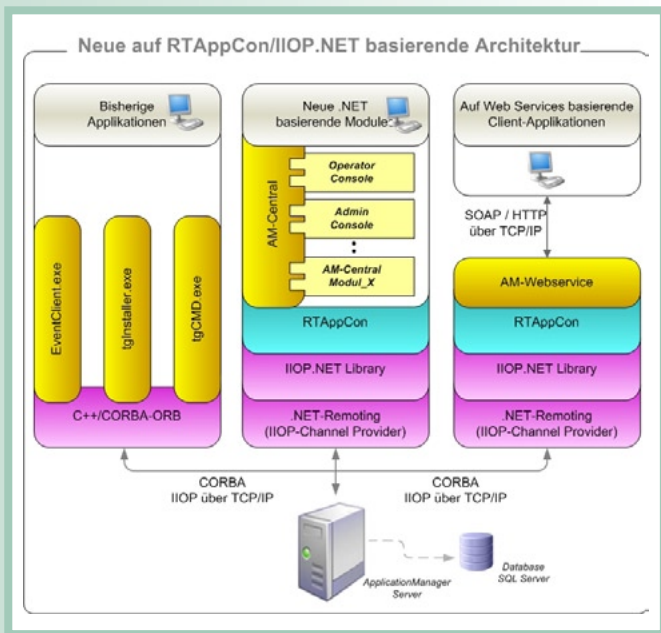


Abb. 2: neue Architektur

bindung öffnet sich der Weg der Integration in eine SOA-geprägte Infrastruktur.

Abbildung 2 gibt einen Überblick über die angestrebte neue Architektur. Dabei stellt die linke Säule die bisherigen Applikationen dar, die direkt auf dem C++/ORB (Object Request Broker) arbeiten. In der mittleren Säule ist der neue mehrschichtige API-Aufbau zu erkennen. Dabei bildet die IOP.NET-Bibliothek zusammen mit dem Microsoft .NET Remoting Framework die unterste Kommunikationsebene. IOP.NET erweitert das .NET Remoting um einen *Internet Inter-ORB Protocol* (IIOP)-Channelprovider, wodurch es möglich wird, direkt mit einer .NET-Programmiersprache einen CORBA-Server zu anzusprechen.

Oberhalb dieser Kommunikationsschicht befindet sich dann in den RTAppCon-Assemblys das .NET-basierende ACAO-Framework. Mit Hilfe dieses Frameworks wird neben infrastrukturellen Aufgaben wie z. B. dem Sessionhandling auch clientseitige Logik elegant gekapselt. Die RTAppCon kann zum einen zur Implementierung von Clientapplikationen verwendet werden. Zum anderen kann die Funktionalität dieses .NET-basierenden Frameworks durch weitere GUI-Frameworks, wie z. B. das „theGuard! ApplicationManager Central“ (AM-Central), erweitert werden.

Die neue Welt

Das ACAO-Framework stellt den verlängerten Arm des Managementsservers dar. Die reine Visualisierung kann nun in Form von vielen GUI-Applikationen umgesetzt werden. Ist das gemeinsame ACAO-Framework detailliert dokumentiert und getestet, ist der Grad der Codewiederverwendung extrem hoch. Noch besser ist es, anstatt einer Menge von vielen Einzelapplikationen eine gemeinsame Applikationsplattform zu finden und die einzelnen Anwendungen als Plug-Ins zu realisieren. Werden diese Plug-Ins zentral von einem der vorhandenen Managementserver zum Download angeboten, ist ganz nebenbei auch das Problem der Softwareverteilung gelöst.

Alle zukünftigen REALTECH-Frontendapplikationen basieren daher auf der *Composite Application Block*-Technologie

[CAB] der Microsofts *Patterns & Practices* [PandP]. In Verbindung mit der *Smart Client Software Factory*-Technologie [SCSF] ergibt sich eine optimale Grundlage für ein Plug-In-basierendes GUI-Framework. Diese als AM-Central-Module bezeichneten Komponenten stellen dann die eigentlichen Frontendapplikationen dar (s. Abb. 2).

Das AM-Central bietet damit clientseitige Services, die in ihrer Art von typischen Systemmanagement-Anwendungen immer wieder benötigt werden. So gibt es Services für die Anmeldung an Managementservern (z. B. Single Sign-On (SSO)), einen Repository-Service für die Abbildung der Managementbäume, Kryptomodule zum Verschlüsseln und Verwalten von Passwörtern und vieles mehr. Neue GUI-Applikationen werden in Zukunft in Form eines AM-Central-Moduls realisiert. Alle weiteren serverspezifischen Dienste erreichen die GUI-Module über das jeweils benötigte ACAO-Framework.

Wie die mittlere Säule von Abbildung 2 zeigt, bildet die Verwendung von CAB die oberste Ebene der clientseitigen Services und Frameworks, die ein GUI-Entwickler braucht, um Anwendungen für einen Managementserver zu schreiben.

Als dritte und letzte Säule ist die Umsetzung eines Webservices zu sehen. Auch hier bildet das ACAO-Framework die Grundlage für einen Webservice.

IOP.NET im Einsatz

Nach dem Entpacken der von SourceForge herunter geladenen ZIP-Datei zeigt sich eine übersichtliche Ordnerstruktur. Nachdem zwei IDL-Dateien des verwendeten ORBs (ir.idl und orb.idl) in das lokale IOP.NET-IDL-Verzeichnis kopiert wurden, kann die Bibliothek mit den zugehörigen Tools durch Aufruf eines Make-Skripts leicht gebaut werden. Das Resultat sind das Kernmodul IOPChannel.dll und die Werkzeuge CLSToIDLGenerator.exe bzw. IDLToCLSGenerator.exe. Im vorliegenden Fall ist der IDLToCLSGenerator das Werkzeug, welches aus den bestehenden IDL-Dateien der Serveranwendung die zugehörigen Proxyklassen in Form von .NET-Assemblys erzeugt. Das IOPChannel.dll-Assembly hat im Wesentlichen die Aufgabe, das Microsoft .NET Remoting Framework um das von CORBA benutzte IOP-Protokoll zu erweitern.

Wie bereits erwähnt, gründen alle Managementserver auf einer gemeinsamen Codebasis. Es handelt sich dabei um eine Serverapplikation, die alle üblichen Funktionalitäten und Services eines Managementsservers bereitstellt. Das tgServerTemplate genannte Projekt ist vollständig in sich geschlossen und lauffähig. Es bietet ausschließlich Basisfunktionalitäten, die eine Serveranwendung grundsätzlich benötigt (Speicher-, Session-, Objektlaufzeit-Verwaltung, Datenbankanbindung, Logfile-Services, usw.) [VöKiZd05]. Dieses Template ist der Ausgangspunkt für die tgAM- und tgIDM-Serveranwendung. Aufgrund dieser Gemeinsamkeit teilen sich diese Server auch einen erheblichen Teil von CORBA-Interfaces. Diese „Shared-IDLs“ sollen auf ein eigenständiges Proxy-Assembly abgebildet werden. Die IDLs der Server (tgAM und tgIDM) erhalten ihre eigenen Proxy-Assemblys. Gerade hier lauern aber Gefahren von Mehrdeutigkeiten, welche erst bei der Verwendung der Assemblys in einem .NET-Projekt zu Tage kommen. Naturgemäß referenzieren die IDLs einer Serveranwendung auch die „Shared-IDLs“. Deren Struktur, Datentyp und Interfacedefinitionen sind nun sowohl in den Shared-Proxy-Assemblys als auch in den Server-Assemblys enthalten. Werden beide Assemblys in einem .NET-Projekt referenziert, kommt es zu den angesprochenen Mehrdeutigkeiten. Abhilfe schafft hier die Option `-r assembly` des IDLtoCLSCompilers, welche dafür sorgt,

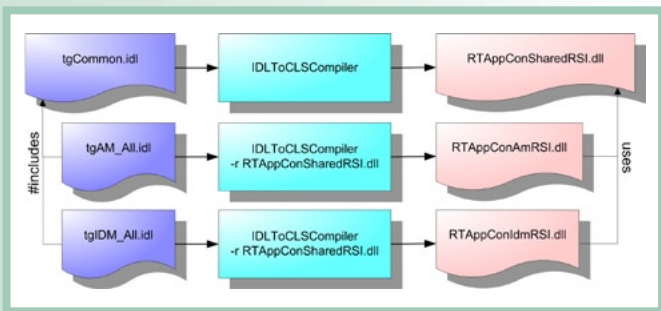


Abb. 3: Von der Interface Definition Language (IDL) zum Remote Server Interface (RSI)

dass in dem angegebenen Assembly befindliche Datentypen, Strukturen und Interfaces nicht erneut generiert werden.

Für unseren Systemmanagementserver tgAM und den Identitymanagementserver tgIDM ergeben sich demnach insgesamt drei Assemblys. Der Namenszusatz „RSI“ steht für Remote-Server-Interface und kennzeichnet somit alle Assemblys, die eine oder mehrere Server-IDLs kapseln und durch das IIOP.NET-Tool IDLtoCLSCompiler erzeugt wurden (s. Abb. 3).

Diese Assemblys stellen die reine Abbildung der serverseitigen CORBA-Interfaces dar und sollten von einem Cliententwickler nicht direkt benutzt werden. Vielmehr wird mit Hilfe des neuen ACAO-Frameworks eine .NET-übliche Programmierschnittstelle geschaffen, die einem GUI-Entwickler ermöglicht, den Server so anzuprogrammieren, als wäre er lokal vorhanden. Immer wieder benötigte Clientlogik wie z. B. das Sessionhandling, Remote Errorhandling usw. kann bereits so aufbereitet werden, dass sie im ACAO-Framework transparent vorhanden ist. Das jeweilige ACAO-Framework manifestiert sich in seiner jeweiligen Assembly. So befindet sich das tgAM-Framework in der RTAppConAM.dll, bzw. das tgIDM-Framework in der RTAppConIDM.dll. Abbildung 4 verdeutlicht noch mal den strukturellen Aufbau aller beteiligten Komponenten.

Fazit

Der Erfahrungszeitraum erstreckt sich nunmehr auf fast zwei Jahre. In der Zwischenzeit wurden bereits die ersten Produkte ausgeliefert, die der beschriebenen Architektur zu Grunde liegen. Der eingeschlagene Weg, die dargestellten Technologien einzusetzen, hat technisch wie personell Anklang gefunden. Die Entscheidung, im Clientbereich eine Trennung zwischen GUI- und Logikprogrammierung vorzunehmen, wurde vom Team motivierend aufgenommen, da der Einzelne seine Stärke im jeweiligen Bereich zeigen kann. Im Übrigen fördert diese Trennung eine saubere Gesamtarchitektur in der Clientanwendung.

Die hohe Kompatibilität zu dem von uns verwendeten ORB und die sehr gute Laufzeitstabilität der IIOP.NET-Bibliothek gibt genug Freiraum, sich auf die Arbeiten in der Entwicklung zu konzentrieren, wovon das Produkt profitiert.

Der Betreuer der IIOP.NET-Bibliothek, Dominic Ullmann, hat bereits signalisiert, dass es mit dem Projekt weitergehen wird. Es ist verständlich, dass Anfragen im Supportbereich nicht immer sofort beantwortet werden, bedenkt man, dass ein solch komplexes Projekt sehr viel Zeit in Anspruch nimmt. Aus eigener Erfahrung kann aber gesagt werden, dass der Autor für externe Beteiligung an Problemlösungen durchaus offen ist und diese Hilfe auch gerne annimmt.

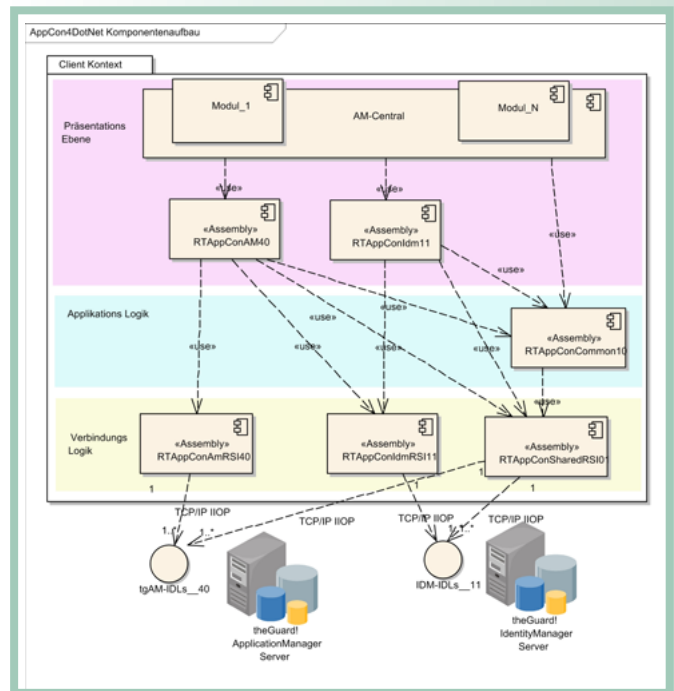


Abb. 4: Komponentenaufbau

Literatur und Links

- [AmRo05] R. Ammon, K. Rohe, Integrationsmöglichkeiten von .NET-Anwendungen mit J2EE und CORBA, in: JavaSPEKTRUM, 6/2005
- [CAB] Smart Client – Composite UI Application Block, Microsoft, <http://msdn.microsoft.com/en-us/library/aa480450.aspx>
- [Ecma] Ecma International, private Standardisierungsorganisation zur Standardisierung von Informations- und Kommunikationssystemen, <http://www.ecma-international.org>
- [IiopSf] IIOP.NET, sourceforge.net, <http://iiop-net.sourceforge.net> und <http://sourceforge.net/projects/iiop-net>
- [Orbacus] Orbacus, IONA, www.orbacus.com
- [PandP] patterns and practices Developer Center, Microsoft, [http://msdn.microsoft.com/de-de/practices/default\(en-us\).aspx](http://msdn.microsoft.com/de-de/practices/default(en-us).aspx)
- [PoSa00] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, Pattern-orientierte Softwarearchitektur, Addison-Wesley, 2000
- [RaSz05] I. Rammer, M. Szpuszta, Advanced .NET Remoting, Second Edition, Apress, 2005
- [SCSF] Smart Client Software Factory, Microsoft, <http://msdn.microsoft.com/en-us/library/aa480482.aspx>
- [VöKiZd05] M. Völter, M. Kircher, U. Zdun, Remoting Patterns, Wiley, 2005



Justus Schach arbeitet als Teamleiter und Systemarchitekt bei der Firma REALTECH system consulting GmbH. Schwerpunkte seines Teams sind die Pflege und Weiterentwicklung der ApplicationManager-Serverplattform und deren Frontendapplikationen. E-Mail: justus.schach@realtech.com.