TIDE
Ein integrierbares Tool zur Analyse von Ausgabedaten mikroskopisch basierter Simulationsrechnungen an komplexen Bediensystemen aus dem Verkehrsraum

Dr. Christos Lois, Axel B. Claßen
DLR – Deutsches Zentrum für Luft- und Raumfahrt
Flughafenwesen und Luftverkehr

Linder Höhe, D-51147 Köln
E-Mail: christos.lois@dlr.de

Zusammenfassung

Im Mittelpunkt der Forschungsaktivitäten der DLR (Deutsches Zentrum für Luft- und Raumfahrt) Flughafenwesen und Luftverkehr steht unter anderem die Erforschung der Ereignisse und Prozesse eines Flughafenterminals zur Unterstützung des Terminalmanagements. Eine der Hauptaufgaben des Terminalmanagements besteht darin, für den optimalen Ablauf der Terminalprozesse zu sorgen. Zum optimalen Verlauf dieser Prozesse gehört die Vermeidung von langen Wartezeiten und Warteschlangen sowie Verspätungen und zu hohen Passagierverdichtugen in kritischen Terminalbereichen (Check-In-Schalter, Kontrollstellen, Treppen etc.). Da sich die obigen Prozesse teilweise über mehere Terminalräume verteilen und miteinander interagieren, gilt ein Flughafenterminal als komplexes System, dessen Analyse hoch entwickelter Methoden bedarf. Dazu gehören Simulationstechniken. TOMICS (Traffic Oriented Microscopic Simulator) ist ein von DLR Flughafenwesen und Luftverkehr entwickeltes, softwarebasiertes Tool zur Simulation von Systemen aus dem Verkehrsraum mit Focus auf Flughafenterminals. Die Modellierung eines Flughafenterminals mittels TOMICS erfolgt mit Hilfe von Grundobjekten („Source“, „Exit“, „Room“, „Obstacle“ und „Person“), die als GUI-Elemente über die TOMICS-Benutzeroberfläche komfortabel bedienbar sind. Diese Objekte sind mit editierbaren Charateristika ausgestattet, die die Eingabe von Modellierungsparametern ermöglichen. Dazu gehören beispielsweise mittelere Passagiergeschwindigkeiten, mittlere Bedienzeiten von Bedienungsstellen sowie Gruppenzugehörigkeiten bei Gruppenbildung und „Exit-Source“-Verküpfungen zur Bestimmung von möglichen Zielrichtungen. Ziel einer Terminalsimulation ist, die Situation in kritischen Terminalbereichen zu einer gegebenen Zeit vorherzusagen und dadurch einen möglichen Verbesserungsbedarf zu identifizieren sowie eine Aussage über die Leistungsfähigkeit des modellierten Terminals zu gewinnen und Optimierungspotenziale zu erkennen. Am Ende einer Terminalsimulation stellt sich also die Aufgabe der Auswertung und Visualisierung charakteristischer Leistungsgrößen und Merkmale eines simulierten Terminals. Da sich der Umfang der TOMICS-Funktionalitäten auf Modellierung und Simulation beschränkt, ist zur Interpretation und Analyse der Ausgabedaten ein Analysetool erforderlich. Auf der Basis einer Anforderungsanalyse und einer Tradeoff-Untersuchung alternativer Tools wurde entschieden, dieses Tool durch DLR-eigene Entwickler als eigenständige Anwendung neu zu entwickeln. Das zu entwickelnde Tool wurde TIDE (**T**OMICS **I**ntegrated **D**ata **E**xplorer) genannt und sollte plattformunabhängig und interoperabel sein. Bei der Entwicklung wurde die Programmiersprache Java verwendet. Dabei kam das von Sun Microsystems frei zur Verfügung gestellte Softwarepaket JSDK 1.4 und einige weitere externe Bibliotheken zum Einsatz, die ohne Ausnahme der „Open Source“-Gemeinde zuzurechnen sind. TIDE wurde mit folgenden Funktionalitäten ausgestattet: Projektverwaltung, Online- und Offline-Betrieb, Logging, Datenexport, Datenauswertung und Ergebnisvisualisierung. Die aktuelle TIDE-Version ermöglicht die Ermittlung der Leistungrößen eines Bedienungsystems, die typischerweise zur Bewertung der Leistungsfähigkeit eines simulierten Terminalsmodells herangezogen werden. Diese sind die mittlere Wartezeit in einer Warteschlange, die mittlere Warteschlangenlänge und die Serverauslastung. Parallel zu diesen Berechnungen werden die Ankunftsverteilung der Passagiere innerhalb eines vorgegeben Zeitfensters analysiert und einige repräsentative Verteilungsprofile graphisch dargestellt.

1. Motivation

Die Organisationseinheit (OE) „Flughafenwesen und Luftverkehr“ des Deutschen Zentrums für Luft und Raumfahrt (DLR) beschäftigt sich unter anderem mit der Optimierung von Prozessen in Flughafenterminals. Dies erfordert das Verständnis dieser Prozesse und deren Zusammenhänge. Diese Aufgabe stellt insofern eine Herausforderung dar, als dass diese Prozesse vielfältig, dynamisch und teilweise über mehre Räumlichkeiten hinweg miteinander interagieren. Außerdem sind sie durch das oft unvorhersehbare menschliche Verhalten geprägt und bilden aufgrund externer Einflüsse (land- und luftseitig) ein offenes System ab. Damit stellt ein Flughafenterminal ein komplexes System dar, dessen Analyse sophistischer Methoden wie z.B. Simulationstechniken bedarf. Simulationsuntersuchungen sind an Flughafenterminals bereits mehrfach eingesetzt worden (Babeliowsky, 1997; Snowdon et al., 1998; Gatersleben & Van der Weij, 1997; Joustra & Van Dijk, 2001; Verbraek & Valentin, 2002, Doshi & Moriyama, 2002). Diese Untersuchungen zeigen, dass die eingesetzten Simulationstools nur näherungsweise die reale Situation eines Flughafenterminals wiedergeben können. Der Bedarf an intelligenteren und in ein Flughafenmanagementsystem integrierbaren Simulationssystemen ist noch groß. Die oben genannte DLR Flughafenwesen und Luftverkehr hat ein Projekt in dieser Richtung gestartet, in dessen Rahmen das Simulationsprogramm TOMICS (Traffic Oriented Microsscopic Simulator) entwickelt wurde. TOMICS ist ein leistungsfähiges Tool zur Modellierung und Simulation von Ereignissen und Bedienprozessen aus dem Verkehrsraum, dessen Focus auf dem Flughafenterminal liegt. Charakteristische Ereignisse dieser Art sind beispielsweise die Ankunft von Personen (land- und luftseitig), die Wegfindung, das Warten bzw. der Aufenthalt von Personen (Passagiere, Begleitpersonen, Personal etc.) im Flughafenterminal und das Verlassen des Terminals (land- und luftseitig). Typische Prozesse einer Passagierabfertigung sind z.B. der Check-In, die Gepäckaufgabe, die Gepäckrückgabe, die Bordkarten- und die Sicherheitskontrolle sowie die Grenz- und die Zollkontrolle (s. Bild 1). Da sich der Umfang der TOMICS-Funktionalitäten vorwiegend auf die Modellierung und die Simulation beschränkt, können weitere Aufgaben, die mit der Interpretation und der Analyse der Simualtionsausgabedaten verbunden sind, durch TOMICS nicht behandelt werden. Die Untersuchungen, die diesem Paper zugrunde liegen, haben daher das Ziel, TOMICS mit Hilfe neuer Tools zu unterstützen.

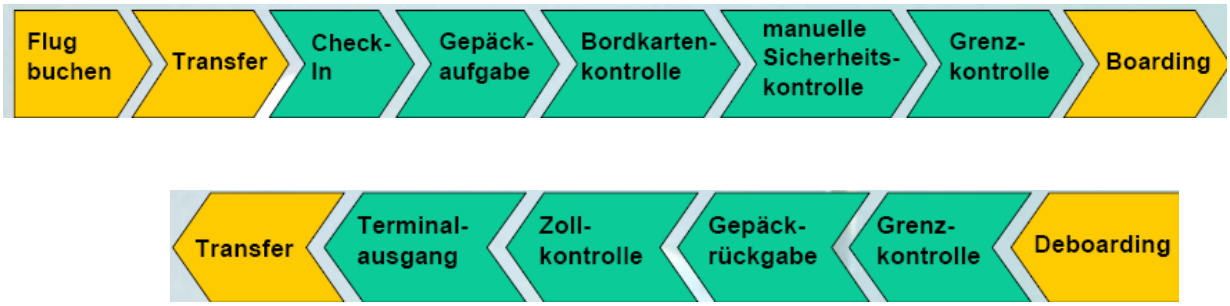

Bild 1: Typische Passagierabfertigungsprozesskette im Terminal

2. TOMICS-Simulationsgrundlagen

Zur Simulation eines Anwendungsszenarios mittels TOMICS ist die Erstellung eines entsprechenden Modells erforderlich. TOMICS ist mit einer graphischen Benutzeroberfläche (GUI) ausgestattet, die zur Modellierung und zur Visualisierung der Ereignisse und Prozesse während der Simulation dient. Dazu werden graphische Elemente (Objekte) benutzt, die Grundelemente aus der Verkehrs- und Warteschlangentheorie darstellen und erforderlich für die Modellierung von Bediensystemen sind. TOMICS unterscheidet die folgenden für den Verkehrsraum (Terminal) relevanten Elemente:

- Sources (Quellen) und Exits (Senken), die zur Modellierung von Ein-, Ausgängen und Bedienstellen dienen,
- Rooms und Obstacles, die die Modellierung von Räumlichkeiten und Hindernissen ermöglich,
- Persons und Aircrafts, als dynamische Objekte, die für die Modellierung von Menschen und Flugzeugen vorgesehen sind.

Mit Hilfe dieser Elemente werden typische Anwendungsszenarien eines Flughafenterminals abgebildet. Um eine realitätsnahe Funktion jedes Objekts innerhalb des Anwendungsszenarios zu erreichen, ermöglicht TOMICS eine interaktive Eingabe von Objekteigenschaften (Charakteristika).

Zu den Charakteristika eines Source- bzw. eines Exits-Objekts gehören graphische Merkmale (der Name, das Aussehen, die Breite und die Höhe), prozessorientierte Merkmale (Kapazität, Prozesszeit etc.) und die Art des Ein- oder Ausganges (Start- oder Zwischeneingang bzw. End- oder Zwischenausgang). Ein Starteingang kann entweder als eine permanente oder als eine planmäßige Personen-Quelle mit oder ohne Personengrenze modelliert werden. Die Person-Generierung erfolgt mit Hilfe eines Zufallsgenerators, dessen mittlere Zwischengenerationszeit einer Verteilungsfunktion unterliegt. Hierbei stehen sowohl mehrere Modellverteilungsfunktionen als auch empirisch ermittelte Funktionen zur Verfügung. Bei der Modellierung eines Zwischeneingangs hinter einem Ausgang (Exit) ist die Verbindung zu diesem Ausgang erforderlich. Es ist darüber hinaus möglich, Menschenströme zu modellieren, die sowohl einzelne Personen als auch Gruppen unterschiedlicher Geschwindigkeit und Gruppenbindungsstärke enthalten. Dies erfolgt durch Adaption von Person-Typen, die mit Hilfe des Person-Elements erstellt werden können (s. unten). Ein Endausgang kann entweder als eine permanente oder als eine planmäßige Personen-Senke mit oder ohne Personengrenze modelliert werden. Zu den Charakteristika eines Objekts vom Typ „Room“ oder „Obstacle“ gehören z.B. der Name und ihre geometrische Struktur. „Room“-Objete sind darüber hinaus durch ihre Kapazitätsgrenze, ihre Entropie (Maß der „Übersichtlichkeit“ in einem Simulationsraum) und ihre „Path Cuboid size“ charakterisiert, die für den Wegfindungsalgorithmus der Personen maßgeblich ist. Hindernisse können mit unterschiedlichen Durchlässigkeiten modelliert werden.

Das Person-Element dient dazu, Klassen (Typen) von beweglichen Objekten (in der Regel Menschen) unterschiedlicher Laufgeschwindigkeiten zu definieren, die dann bei der Modellierung von Sources (Personen-Quellen) verwendet werden können. Die mittlere Laufgeschwindigkeit wird auf der Basis einer Verteilungsfunktion (Normal, Expo, Gamma etc.) beschrieben.

Die einzelnen Person-Objekte werden als Individuen behandelt. D.h. unter anderem, dass jede in der Simulation erzeugte „Person“ den optimalen Weg zu ihrem Ziel separat und unabhängig von den anderen „Personen“ sucht, es sei denn, es gehört zu einer Gruppe, die dann den Regeln des so genannten „Swarmings“ unterworfen ist. Zur Modellierung der Flexibilität einer Gruppe dient der Modellparameter der Bindungsstärke. Es ist darauf hinzuweisen, dass die Person-Objekte insofern (noch) keine Künstliche Intelligenz besitzen, die ihnen evtl. ermöglichen würde, Gegenstände und Hinweise automatisch zu erkennen, um so Ziele selbst zu finden. Der Modellierer legt die „Sources“ (Ausgangspositionen), die „Exits“ (Ziele) und deren Verknüpfung fest. Die Aufgabe der Person-Objekte besteht darin, unter Zuhilfenahme der bereits festgelegten Zielvorgaben, den Weg dorthin, unter Beachtung der Raumgeometrie, zu finden. Bei der Ermittlung des nächsten optimalen Schrittes werden die folgenden Bedingungen berücksichtigt:

- Minimierung der Abweichung von der Zielrichtung
- Einhaltung von gewissen Abständen zu anderen „Person“-Objekten
- Vermeidung von Stößen an Gegenstände (Wände, Hindernisse etc.)

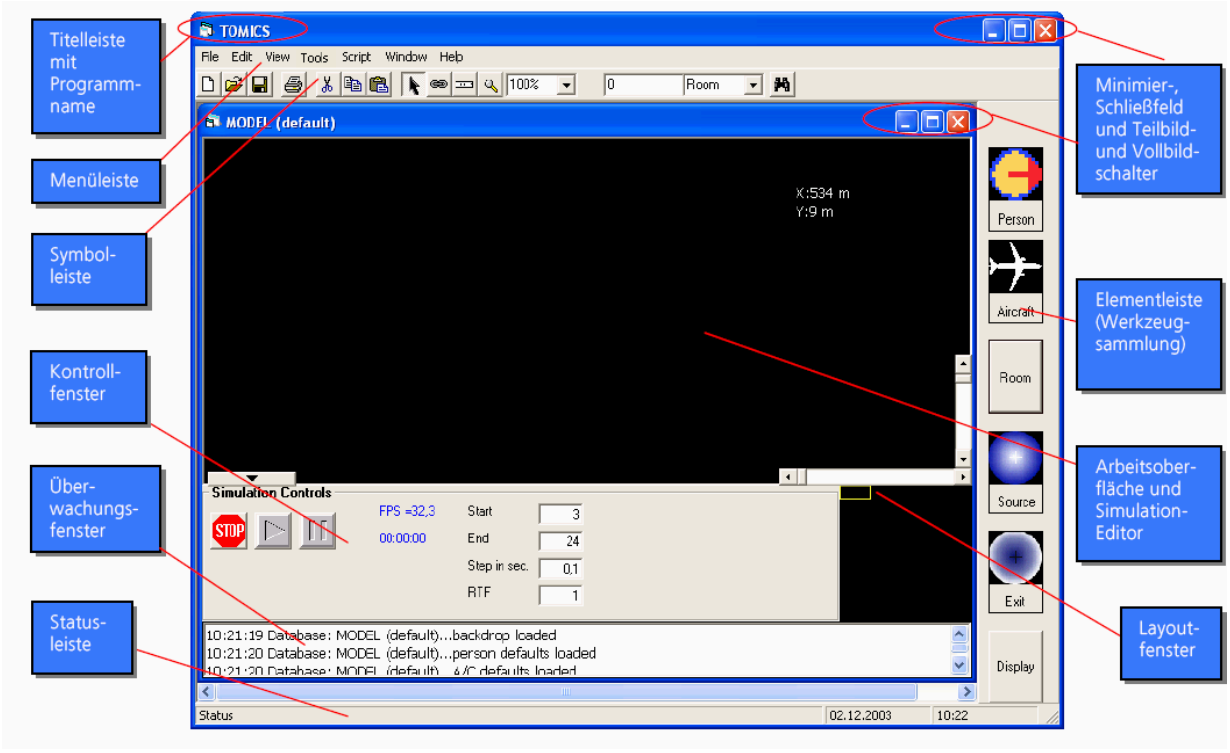

Bild 2: Startseite der TOMICS-Benutzeroberfläche

Bei der Modellierung eines Anwendungsszenarios für ein Flughafenterminal mittels TOMICS ist zunächst die Erstellung eines Flugplans erforderlich. TOMICS bietet einen entsprechenden Editor dazu an. Alle so erstellten Daten werden in einer Datenbank abgelegt. Die Simulation lässt sich über das Kontrollfenster der Benutzeroberfläche starten, anhalten, neu starten oder stoppen. Bild 2 zeigt die Startseite der TOMICS-Benutzeroberfläche.

Zu einer späteren Auswertung des simulierten Modells (Offline-Datenanalyse) werden die Simulationsausgabedaten in einer Datenbank abgelegt. Auch für den Fall einer Online-Datenanalyse bietet TOMICS eine geeignete dafür Schnittstelle. Eine netzwerkfähige Socket-basierte Server-Anwendung wird parallel zu TOMICS gestartet, die an einem Port auf mögliche Client-Verbindungen wartet. Sie ist im Netzwerk über die IP-Nummer des Rechners, auf dem sie läuft, und die Port-Nummer eindeutig identifizierbar. Die an diesem Server angemeldeten Clients erhalten eine dauerhafte (synchrone) Verbindung zu ihm, die auf diese Art und Weise eine Datenübertragung in Echtzeit ermöglicht. Die Server-Anwendung wurde mit einem Datenpuffer implementiert, der von TOMICS mit Daten gefüllt und anschließend als binärer Datenstrom über die Client-Server-Sockets an die Clients weitergeleitet wird. So ist es neben der Online-Datenanalyse zudem möglich, ein verteiltes Simulationsnetzwerk aufzubauen oder ggf. Daten aus dem Realbetrieb eines Terminalmanagementsystems einfließen zu lassen.

3. Nicht-funktionale Anforderungen an das Datenanalysetool

Um den Bedarf eines Tools zur Analyse von flughafenterminalbezogenen Simulationsausgabedaten zu erfüllen, wurde eine erste Analyse von nicht-funktionalen Anforderungen an dieses Tool durchgeführt. Dabei stellten sich unter anderem die folgenden zwei grundlegenden Fragen:

- Sollte dieses Tool einen internen Bestandteil des Simulationstools TOMICS oder eine separate externe Anwendung darstellen?
- Sollte ein solches Tool neu entwickelt oder aus einer bereits existierenden Anwendung heraus adaptiert werden?

Zur Beantwortung dieser Fragen waren die folgenden nicht-funktionalen Anforderungen relevant:

- **Interoperabilität und Wiederverwendbarkeit:**
  Das Tool sollte neben TOMICS noch in anderen Simulationsprogrammen leicht integrierbar (interoperabel, kompatibel) sein. Eine erste Voraussetzung dafür wäre erfüllt, wenn dieses Tool über interoperable Schnittstellen verfügen würde. Die aus einer solchen Interoperabilität resultierenden Vorteile liegen auf der Hand:
  Die Endanwender hätten eine größere Freiheit bei der Auswahl ihres Simulationsprogramms. Eine leichte Integrierbarkeit in bereits existierende oder künftige Simulationsprogramme würde niedrige Entwicklungskosten ermöglichen. Ein wichtiger Schritt in Richtung Wiederverwendbarkeit wäre damit getan. Das Tool würde damit an Mehrwert gewinnen.

- **Plattformunabhängigkeit:**
  Das geplante Tool sollte möglichst plattformunabhängig oder zumindest ohne großen Aufwand auf die verschiedenen Plattformen (Windows, Unix, Linux, Mac) portierbar sein. Das erhöht die Einsatzbreite und folglich die Anzahl der potenziellen Endanwender. Portierungskosten können damit reduziert werden.

- **Verteilbarkeit/Skalierbarkeit:**
  Das angestrebte Tool sollte so konzipiert sein, dass im Fall einer rechenaufwändigen Datenanalyse, bei der evtl. die Rechenkapazität eines Rechners nicht ausreicht (z.B. bei einer Online-Datenvisualisierung), die Prozesslast auf mehrere Prozessoren verteilt werden kann.

- **Multiuser- und Webfähigkeit:**
  Eine weitere Anforderung an das Datenanalysetool war die Möglichkeit, mehrere Benutzer mit der gleichen Anwendung (Installation) bedienen zu können, wenn möglich von jedem Arbeitsplatz in der Welt. Das spart Lizenz-, Installations- und Wartungskosten. Dies ist im Hinblick auf eine Zusammenarbeit mehrerer Anwender aus verschiedenen in der Welt verteilten Forschungseinrichtungen in vielen Fällen ein unverzichtbarer Wunsch. Der erfolgreichste Ansatz zur Realisierung einer solchen Anwendung basiert auf dem Konzept einer Web-gestützten Client /Server-Architektur

- **Offenheit und Herstellerunabhängigkeit:**
  Das geplante Tool sollte möglichst frei von kommerziellen Drittanbieter-Produkten (3rd Party) sein, die mit zusätzlichen Lizenzkosten verbunden sind. Stattdessen sollte auf alternative Lösungen aus der „Open Source Community“ zurückgegriffen werden, deren Benutzung kostenfrei ist und deren Quellcode für mögliche Änderungen offen gelegt ist. Damit werden Entwicklungskosten gespart, was letztlich auch für die Endnutzer Preisvorteile mit sich bringt.

- **Unproprietär und zukunftssicher:**
  Auch bei der Auswahl verschiedener Techniken und IT-Technologien (Programmiersprachen, Datenbanken, etc.) sollte darauf geachtet werden, dass sie bereits als IT-Standards etabliert sind und ihre langfristige Weiterentwicklung und Existenz gesichert ist.

Drei dieser sechs Anforderungen konnten bei der Beantwortung der ersten Frage herangezogen werden:

- ***Interoperabilität und Wiederverwendbarkeit*:** Es ist in der Regel schwierig bzw. mit großem Aufwand verbunden, ein Tool zu entwickeln, das sowohl als interner Bestandteil einer anderen Anwendung als auch als eigene selbständige Anwendung eingesetzt werden kann. Bekannte Beispiele dafür sind z.B. Plugins. Sowohl Plugins als auch die Anwendungen, in die sie integriert werden, müssen bei ihrer Konzeption gewisse Regeln einhalten. Da TOMICS keine Plugin- oder Plugin-ähnlichen Konzepte unterstützt, ist eine zweigleisige Beschaffung bzw. Entwicklung des geplanten Tools mit mehr Aufwand verbunden und daher unvorteilhaft.
- ***Plattformunabhängigkeit:*** TOMICS wird in der Programmiersprache Visual Basic implementiert und wird daher nur von der WINDOWS-Plattform unterstützt. Ein in TOMICS integriertes Tool würde also den gleichen Einschränkungen unterliegen. Damit wäre die Anforderung einer Plattformunabhängigkeit nicht erfüllbar.
- ***Offenheit:*** Auch die „Open Source"-Philosophie wird von Microsoft und von vielen auf der Basis von Microsoft-Technologien entwickelten Produkten nicht unterstützt.

Auf der Basis der obigen Überlegungen wurde entschieden, das geplante Datenanalysetool nicht als internen Bestandteil von TOMICS zu entwickeln bzw. zu beschaffen, sondern als separate externe Anwendung.

Zur Beantwortung der 2. Frage, ob das geplante Datenanalysetool neu entwickelt oder durch eine bereits existierende Anwendung adaptiert werden soll, wurde zunächst eine Funktionsanforderungsanalyse für dieses Tool durchgeführt, die als Basis für eine weitere Tradeoff-Analyse alternativer Tools herangezogen wurde. Es stellte sich dabei heraus, dass keines der in Frage kommenden Tools (kostenfrei oder kommerziell) ohne entsprechende Änderungen bzw. Erweiterungen in der Lage war, die Mehrzahl der oben aufgeführten Anforderungen zu erfüllen und ein passendes Funktionalitätsprofil aufzuweisen. Da einerseits bei den frei verfügbaren Tools der Aufwand für solche Anpassungen oft sehr schwer abzuschätzen ist, und andererseits bei den kommerziellen Tools die Kosten für Wartung, Pflege und Support in der Regel recht hoch sind, wurde entschieden, das gewünschte Datenanalysetool durch DLR-interne Entwickler voranzutreiben.

4. Architekturdesign, Schnittstellenkonzeption und Entwicklungsumgebung

Bei der Konzeption der Software- und Hardware-Architektur des Datenanalysetools wurde auf die Erfüllung der oben aufgeführten Anforderungen geachtet. Darüber hinaus wurde eine Analyse zur Klassifizierung aller in Frage kommenden Anwendungsszenarien durchgeführt. Dieser Klassifizierung wurden die folgenden drei anwendungsbezogenen Betriebsmodi zugrunde gelegt:

I. Lokal oder verteilt (im Internet/Intranet)
II. Offline oder Online
III. Ein-Anwender- und Mehr-Anwender-Unterstützung

Die Kombination dieser Betriebsmodi liefert insgesamt 9 mögliche Anwendungsszenarien, aus denen sich unterschiedliche Anforderungen bei der Implementierung ergeben. Im Folgenden werden zwei repräsentative Szenarien näher erläutert. Diese definieren sich durch folgende Betriebsmodi:

1. Lokal, Offline und Ein-Anwender-Unterstützung
2. Verteilt, Online und Mehr-Anwender-Unterstützung

1. Szenario:
Zur Realisierung eines solchen Szenarios bieten sich mehrere bereits etablierte Architekturmodelle an. Strebt man dabei eine gewisse Wiederverwendbarkeit und Interoperabilität an, so gibt es nur wenige Realisierungsmöglichkeiten. Die Wiederverwendbarkeit und Interoperabilität einer Softwareanwendung hängt davon ab, wie die Schnittstellen dieser Anwendung zur Kommunikation und zum Datenaustausch mit anderen Anwendungen implementiert sind. Zu diesem Zweck sollten sie möglichst sprachunabhängig sein und auf standardisierten Datentransportprotokollen basieren. Eine wichtige Rolle dabei spielt die Form und das Format der auszutauschenden Daten. In der Regel liegen sie als Dateien (Files), als serialisierte Objekte oder als Rekords auf einer Datenbank vor. Bild 3 zeigt das Architekturdiagramm eines datenbankbasierten Datenaustausches zwischen zwei Anwendungen.

Ein solches Kommunikationsmodell ist mehrfach eingesetzt worden und daher gut etabliert. Ein Grund für dessen Erfolg besteht darin, dass die Datenbank eine Datenschnittstelle offeriert, die sprachunabhängig zugänglich ist. Dies erfolgt dadurch, dass sich die Implementierung dieser Schnittstelle auf den TCP/IP-Level beschränkt und daher sprachunabhängig ist. Die Clients können ihren Zugang (Datenbanktreiber) in einer beliebigen Programmiersprache implementieren, die das TCP/IP-Netzwerkprotokoll unterstützt. Der DB-Treiber ist in der Regel nicht vom Anwendungsentwickler selbst zu entwickeln, sondern ist Bestandteil der Lieferbibliotheken der von ihm benutzten Programmiersprache und

kann vom ihm benutzt werden. Er ist mit Funktionen ausgestattet, die die komplexen Schritte des (netzwerk-basierten) Datenaustausches mit der Datenbank kapseln und damit die Arbeit des Entwicklers vereinfachen. Das obige datenbankbasierte Datenaustauschmodell hat jedoch den Nachteil, dass es für eine Realtime-Kommunikation und damit für eine Online-Datenanalyse wenig geeignet ist. Dies liegt daran, dass eine Datenbank das parallele Abspeichern und Exportieren von Daten nicht zulässt. Diese zwei Aktivitäten können nur sequenziell durchgeführt werden, wobei jede Aktivität mit dem Öffnen der Datenbank beginnt und mit dem Schließen der Datenbank endet. Das ständige Öffnen und Schließen einer Datenbank bei einer Realtime-Übertragung wäre für die Datenbank eine Überforderung.

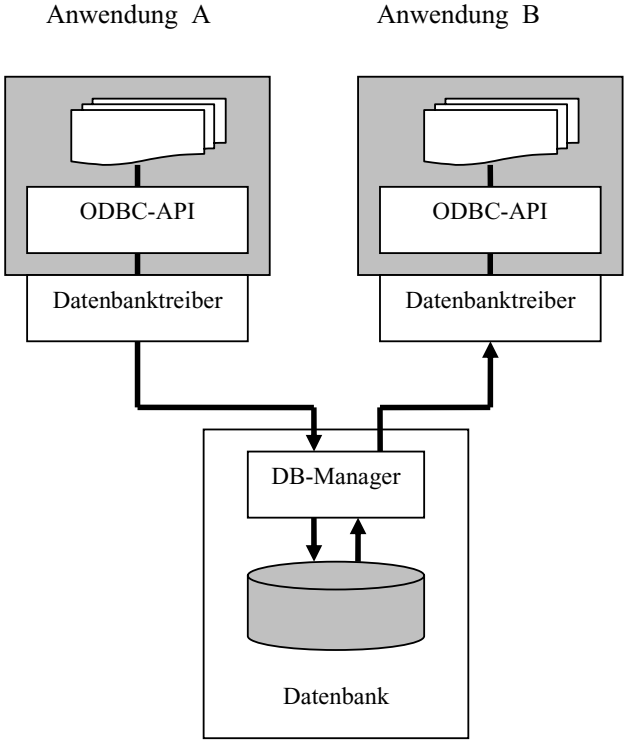

Bild 3: Architekturmodell eines Datenbank-basierten Datenaustausches zwischen zwei Anwendungen

2. Szenario:
Das datenbankbasierte Datenaustauschmodell ist für eine Realtime-Datenübertragung wenig geeignet. Zur Lösung dieses Problems werden Netzwerkkommunikationstechniken nach dem Client/Server-Modell benutzt. Dabei kommunizieren die Anwendungen direkt miteinander, wobei eine Anwendung als Server (Datenlieferant) agiert und die andere(n) Anwendung(en) als Client(s) (Datenempfänger). Die am meisten verbreiteten Netzwerke sind das Internet und das Intranet. Bei diesen Netzweren basiert der Datentransport auf dem TCP/IP- und dem UDP/IP-Transportprotokoll. Je nach dem ob die Daten bei ihrem Versand mit zusätzlichen (Anwendungs-) Protokollen verpackt werden oder nicht, lassen sich die folgenden drei am häufigsten realisierten Transportarten unterscheiden:

- Transport über Socket-basierte Verbindungen
- Transport über FTP- oder http-basierte Verbindungen und
- Transport über API-basierte Verbindungen

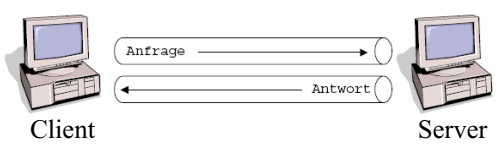

Bild 4: Synchrone Client/Server-Kommunikation

Socket-basierte Verbindungen ermöglichen eine synchrone, schnelle und sichere (frei von Informationsverlusten) Datenübertragung von Daten im Netzwerk (s. Bild 4). Die Daten werden dabei nur mit dem TCP/IP-Protokoll verpackt und in Form eines binäreren Datenstroms transportiert. Da es sich dabei um eine synchrone, verbindungsorientierte Datenübertragung handelt, ist sie für eine Realtime-Datenübertragung gut geeignet.

Bei einer FTP-basierten Kommunikation werden die Daten, die bereits als Dateien (files) vorliegen sollen, vor ihrem Versenden vom Server mit einem zusätzlichen (Anwendungs-)Protokoll, nämlich dem FTP-Protokoll verpackt. Dies dient zur Dekodierung der Daten durch die Client-Anwendung. Gleiches gilt für das http-Protokoll. Web-Browser stellen die bekanntesten http-basierten Client-Anwendungen dar. Da bei den FTP- und http-Anwendungen meistens das UDP/IP-Transportprotokoll eingesetzt wird, sind FTP- bzw. http-Verbindungen asynchrone Verbindungen und daher wenig geeignet für eine Realtime-Datenübertragung.

Die Kommunikation zwischen zwei Anwendungen über ein API („Application Programming Interface“) erfolgt über eine Programmierschnittstelle. Dabei wird eine Schnittstelle definiert, die eine oder mehrere leere Funktionen (Methoden) definiert. Ob der Server oder Client diese Schnittstelle zu implementieren hat, hängt vom Anwendungsszenario ab. Soll z.B. eine API-basierte Realtime-Datenübertragung zwischen einem Server und einem Client realisiert werden, so wird meistens die so genannte „callback“-Technik eingesetzt, bei der der Client die API-Schnittstelle implementiert. Eine API-basierte Client/Server-Kommunikation setzt jedoch voraus, dass Client und Server in derselben Programmiersprache implementiert sind. Das Problem der Integration eines Clients und eines Servers in unterschiedlichen Sprachen wurde bereits früher erkannt. Das bekannteste Lösungskonzept dazu lautet „Middleware“. Die am meist verwendete Middleware-Technologie unter den objektorientierten Sprachen, deren Konzept sprachunabhängig

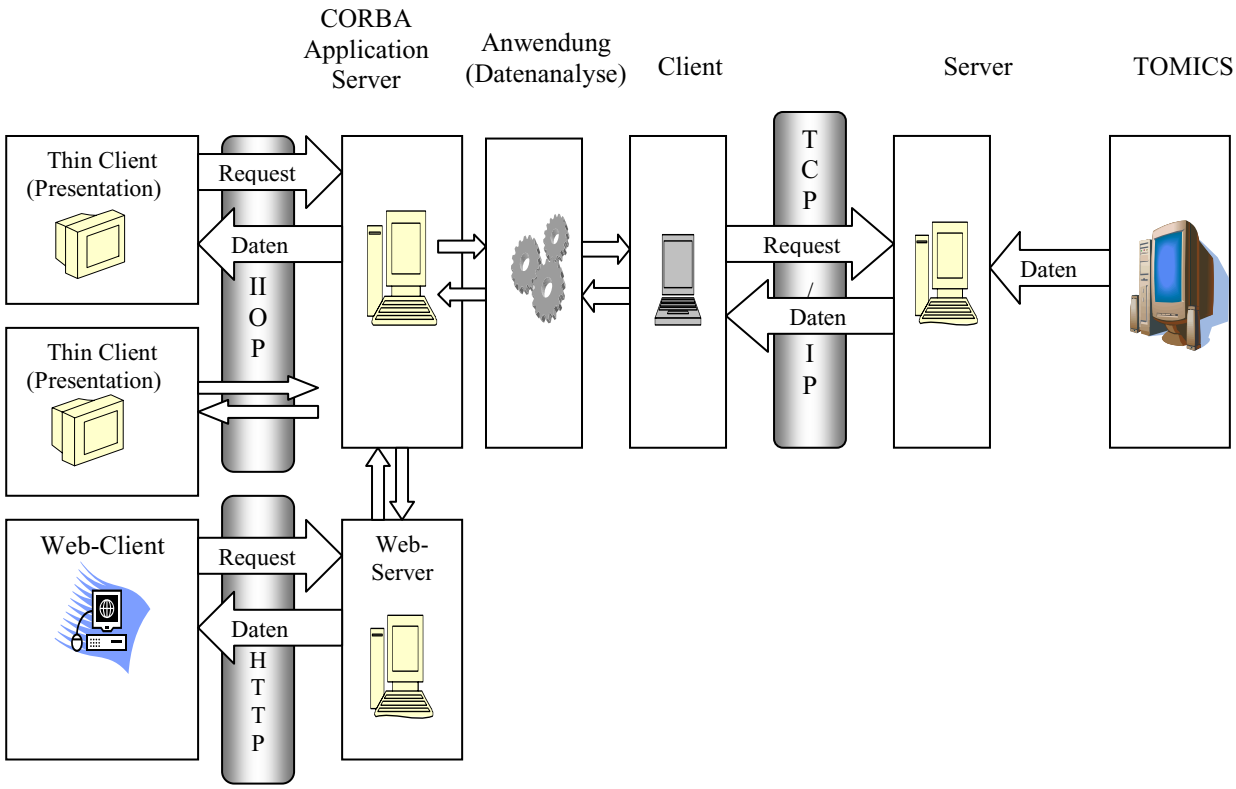

Bild 5: Architekturmodell einer Realtime-Kommunikation zwischen TOMICS und einem Datenanalysetool mit Unterstützung mehrerer Anwender.

ist, heißt CORBA (Corba; 2006). Für nicht objektorientierte Sprachen gibt es nur proprietäre Insel-Lösungen, die jedoch nur für bestimmte Sprachen gelten. Aus den obigen Überlegungen geht also hervor, dass eine sprachunabhängige Realtime-Kommunikation zwischen dem Simulationsprogramm TOMICS und einem Datenanalysetool standardmäßig nur auf der Basis von TCP/IP-Sockets erfolgen kann. Zur Unterstützung von mehreren Anwendern in einer Netzwerkumgebung müsste das Datenanalysetool nach dem Client/Server-Architekturmodell mit Hilfe eines CORBA- bzw. Web-basierten Applikationsservers konzipiert werden. Bild 5 zeigt das Datenflussdiagramm einer solchen Client/Server-Architektur.

Die Realisierung einer Technologie, die eine Plattformunabhängigkeit von Softwareprogrammen ermöglicht, gelang der Firma Sun Microsystems, Inc. (Suna, 2006; Sunb, 2006) durch die Erfindung der Programmiersprache Java im Jahr 1995. Die Innovation bestand darin, dass ein kompilierter Java-Quellcode nicht von der Hardware (Prozessor) des Rechners auszuführen ist, sondern von einem softwarebasierten (virtuellen) Prozessor. Der wurde als Java-VM („Java Virtual Machine“) bezeichnet. Die Ausführbarkeit der Java-Programme setzt also das Vorhandensein der Java-VM auf dem jeweiligen Rechner voraus. Da Sun mittlerweile Java-VM für fast alle Plattformen zur Verfügung stellt, ist ein lauffähiges Java-Programm auf fast allen Plattformen ausführbar, unabhängig davon, auf welcher Plattform es kompiliert wurde. Es gilt also hier der bekannte Satz „Write Once, Run Anywhere“. Java wird von Sun ständig erweitert und die dadurch entstehenden Bibliotheken kostenfrei zur Verfügung gestellt. Der Umfang dieser Bibliotheken umfasst mittlerweile so viele APIs und Frameworks, dass Java nicht nur eine Programmiersprache im klassischen Sinne des Wortes darstellt, sondern eine mächtige Entwicklungsplattform, die die Programmierung von komplexen GUIs bis hin zur Realisierung von Anwendungen für Netzwerke (Internet/Intranet), mobile Geräte, Chipkarten und interaktive 2D/3D-Visualisierungen unterstützt. Dieses breite Spektrum an frei verfügbaren Bibliotheken lässt kaum noch Anwendungsgebiete zu wünschen übrig. Darüber hinaus stellen sowohl Sun Microsystems als auch andere Hersteller und „Communities“ eine Reihe von hilfreichen Tools frei zur Verfügung, die die Softwareentwicklung mit Java unterstützen, so dass eine Java-basierte Softwareentwicklung bei Nullinvestition möglich ist. Java ist im Vergleich zu anderen objektorientierten Programmiersprachen relativ einfach zu erlernen und ist darüber hinaus webfähig – d.h. sie wird von allen gängigen Web-Browsern unterstützt. Damit kann ein Java-Programm, das sich auf einem bestimmten Rechner (Server) im Internet befindet, von jedem anderen Rechner (Client) im Internet über einen Web-Browser ausgeführt werden. Damit eignet sich Java für die Entwicklung von Web-basierten Multiuser-Anwendungen sehr gut.

All diese Aspekte und Merkmale machen Java zu einer der beliebtesten Programmiersprachen zwischen den Entwicklern aus allen Bereichen und vor allem aus dem Bereich der Forschung und Entwicklung. Aus diesen Gründen wurde Java als Entwicklungssprache für das Datenanalysetool gewählt.

5. Die Grundfunktionalitäten von TIDE

Das zur Analyse von Simulationsausgabedaten entwickelte Tool heißt TIDE (**T**OMICS **I**ntegrated **D**ata **E**xplorer). Die aktuelle Version von TIDE (Ver. 1.08) enthält eine Reihe von Funktionalitäten, die den Anwender bei der Datenaufbereitung, der Datenauswertung und der Visualisierung von Simulationsergebnissen unterstützen. Um den Anforderungen an ein modernes, bedienerfreundliches Tool zu entsprechen, wurde TIDE mit einer modernen graphischen Benutzeroberfläche (GUI) ausgestattet (s. Bild 6). Layout und Bedienungsweise wurden so gestaltet, dass sie dem Anwender möglichst bekannt vorkommen, wobei bekannte Entwicklungs- und Office-Programme als Orientierung dienten. TIDE ist mit den folgenden Tools und Funktionen ausgestattet:

- Projektverwaltung über das Pulldown-Menü „Project“ (s. Bild 7a). Sie umfasst das Öffnen, die Erstellung, das Löschen und den Import von Projekten sowie die Entfernung („close“) von Projekten aus dem Verzeichnis der aktiven Projekte. Jedes neu erstellte Projekt wird automatisch in das Verzeichnis der so genannten aktiven Projekte abgelegt. Alle aktiven Projekte sind über das Projektfenster auflistbar und abrufbar. Ein aktives Projekt kann aus dem „aktiven“ Verzeichnis über die „close“-Funktion entfernt werden. Solche Projekte werden in das „inaktive“ Verzeichnis abgelegt. Sie sind dann über das Projektfenster nicht mehr sichtbar. Der Abruf solcher Projekte kann aber jederzeit über die „open“-Funktion erfolgen.
- Die Möglichkeit einer Umschaltung zwischen Offline- und Online-Betriebsmodus mit Hilfe des Pulldown-Menüs „*Mode*“ (s. Bild 7b).
  - Beim Offline-Betrieb wird ein GUI-gestütztes Tool („DB Client“; s. Bild 7c) bereitgestellt, das die Anbindung an die TOMICS-Datenbank, die Abfrage von Daten und den Export dieser Daten als ASCII-Dateien ermöglicht. Die Exportfunktion unterstützt die folgenden Formate: XLS, CSV, SSV und XML (s. Bild 8).
  - Im Online-Betrieb wird ebenfalls ein GUI-gestütztes Tool („RTConnection Client“) zur Verfügung gestellt, das dazu dient, eine dauerhafte synchrone Verbindung mit dem TOMICS-Server herzustellen und die Daten zwischen dem TOMICS-Server und dem Datenanalysetool zu übermitteln. Dieses Tool befindet sich derzeit noch in der Entwicklung.

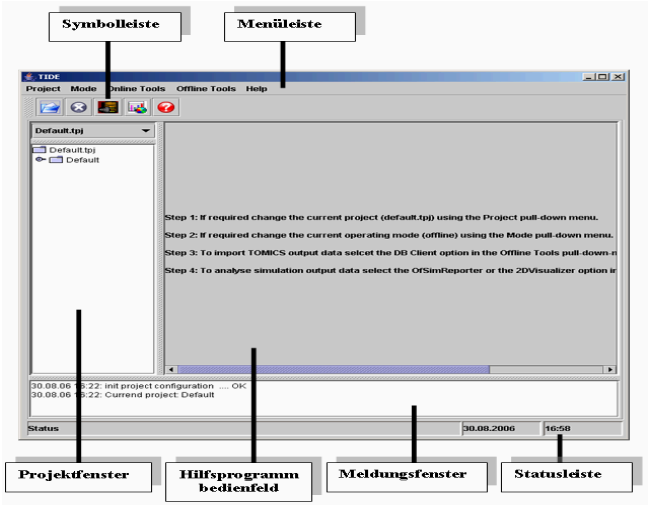

Bild 6: Startseite der TIDE-Benutzeroberfläche

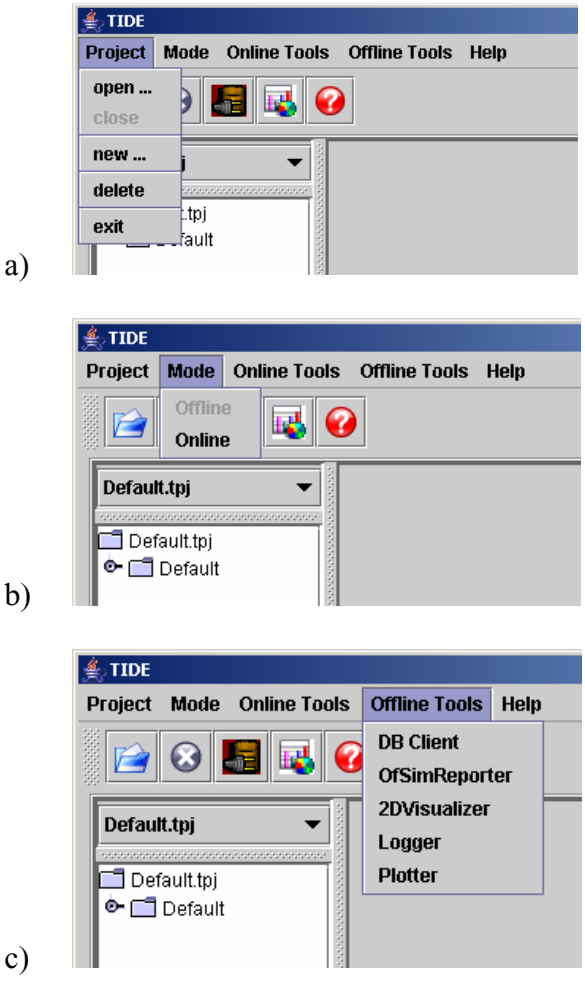

Bild 7: Die Pulldown-Menüs a) „*Project*“, b) „*Mode*“ c) „*Offline Tools*”

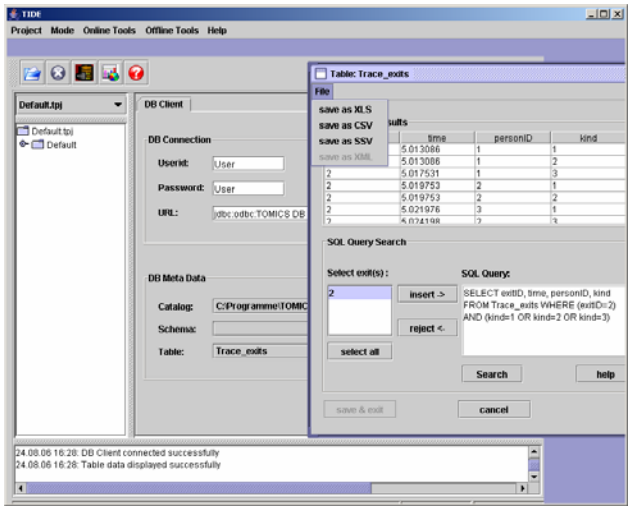

Bild 8: Export von TOMICS-Simulationsausgabedaten aus der TOMICS-Datenbank mit Hilfe des „*DB Client*“-Tools

- Ein Auswertungstool („OfSimReporter“) ermöglicht das Handling, die Aufbereitung und die Auswertung der bereits exportierten Daten (s. Bild 9).
- Die Visualisierung der 2D-bezogenen Ergebnisse soll mit dem „2DVisualizer“ erfolgen, der sich aber noch in der der Entwicklung befindet
- Ein weiteres Modul übernimmt die Protokollierung („Logging“) von Nachrichten während der Datenprozessierung und der Ergebnisvisualisierung.

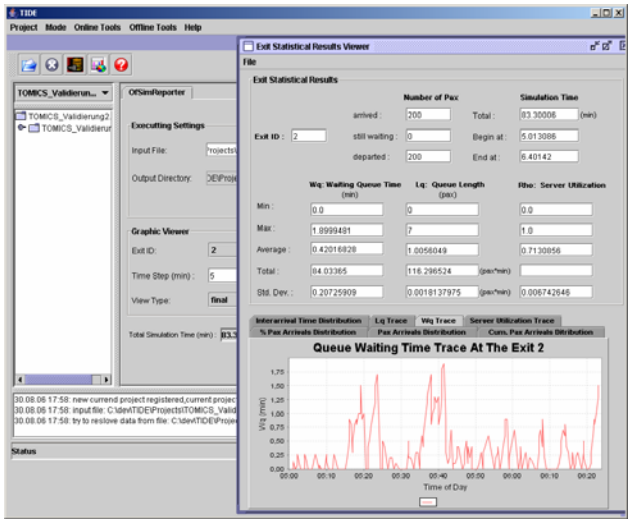

Bild 9: Auswertung (Offline) und Visualisierung von (exportierten) TOMICS-Simulationsausgabedaten mit Hilfe des „OfSimReporter“-Tools

6. Grundlagen zur Analyse der Simulationsausgabedaten

Ziel eines Terminalmanagementsystems ist der optimale Ablauf der Terminalprozesse. Dabei sollen die Betriebskosten durch effiziente Nutzung der verfügbaren Ressourcen möglichst niedrig gehalten werden. Dieses Ziel kann durch eine optimale Ressourcenplanung im Voraus maßgeblich unterstützt werden. Die Erstellung einer solchen Planung setzt die Vorhersage der Flughafenterminallast und des damit verbundenen Kapazitätsbedarfs zu einem bestimmten Zeitpunkt voraus. Kapazitätsengpässe können oft nur durch eine optimale räumliche und zeitliche Lastverteilung bewältigt werden. Eine solche Optimierung kann z.B. durch ein iteratives Testen verschiedener Flughafenterminalmodelle (Anwendungsszenarien) mittels analytischer oder simulationsbasierter Verfahren erfolgen. Diese Tests bestehen darin, charakteristische Leistungsmerkmale und Leistungsgrößen des modellierten Flughafenterminals zu ermitteln und daraus eine Aussage über die Terminalleistungsfähigkeit zu gewinnen. Neben den für ein Bediensystem (Server) typischen Leistungsgrößen der Wartezeit, der Warteschlangenlänge und der Serverauslastung können die LOS-Standards („Level Of Service“; Ashford, 1988; IATA, 1995) als eine erste Basis zur Bewertung der Leistungsfähigkeit eines Flughafenterminals verwendet werden. Die LOS-Standards definieren eine Bewertungsskala für die charakteristischen Größen der Durchflussstärke (pax/m•min) und der Platzverfügbarkeit (m^2/pax) für Bereiche, in denen sich Passagiere oder andere Personen befinden. Im Mittelpunkt der LOS-Untersuchungen stehen kritische Bereiche wie etwa Treppen, Check-In- und Kontrollschalter etc.

Wie in Abschnitt 5 beschrieben, ist in TIDE das Reporting-Tool „OfSimReporter“ integriert, das die Auswertung der TOMICS-Simulationsausgabedaten ermöglicht. Der „OfSimReporter“ unterstützt die Ermittlung des mittleren Wertes und des zeitlichen Verlaufs der folgenden Leistungsgrößen einer Bedienstelle (Server):

- die mittlere Wartezeit der Passagiere in der Warteschlange (Wq),
- die mittlere Warteschlangelänge (Lq) und
- die Auslastung (ρ) einer Bedienstelle (Serverauslastung)

Parallel dazu wird die Ankunftsverteilung der Passagiere innerhalb eines vorgegeben Zeitfensters analysiert und repräsentative Verteilungsprofile können graphisch dargestellt werden. Die Berechnung der mittleren Werte der obigen Leistungsgrößen basiert auf den folgenden Formeln (Law & Kelton, 2000):

- Die mittlere Wartezeit in der Warteschlange („Average Queue Delay“):

$$W_q = \overline{D}(n) = \frac{\sum_{i=1}^{n} D_i}{n} \quad (1)$$

wobei D_i die Wartezeit („Delay“) des i-ten Passagiers und n die Anzahl der bedienten Passagiere sind.

- Die mittlere Warteschlangenlänge („Average Queue Length“):

$$L_q = \overline{q}(n) = \frac{\sum_{i=0}^{Q_{\max}} iT_i}{T(n)} \quad (2)$$

wobei i eine beobachtete Warteschlangenlänge von i Passagieren, $T(n)$ die gesamte Simulationszeit und T_i die Summe aller Zeitintervalle ist, für die eine Warteschlangenlänge von i Passagieren beobachtet wurde. $Q_{\max}$ ist die maximale Warteschlangenlänge, die bei der Simulation beobachtet wurde.

- Die Auslastung einer Bedienstelle („Server Utilization“):

$$\rho = \overline{u}(n) = \frac{\int_0^{T(n)} B(t)dt}{T(n)} \quad (3)$$

wobei $T(t)$ die gesamte Simulationszeit ist und $B(t)$ die „busy function“ der Bedienstelle beschreibt. Sie hat den Wert 1 für die Zeitintervalle, in denen die Bedienstelle genutzt wird, und den Wert 0 für die Zeitintervalle, in denen die Bedienstelle ungenutzt ist.

Zur Ermittlung der obigen Leistungsgrößen eines Bedienungssystems benötigt man die folgenden drei Zeiten jedes bedienten Objekts:

- Zeitpunkt der Ankunft (T_i),
- Beginn der Bedienung (B_i) und
- Ende der Bedienung (E_i)

Um diese Informationen als Ausgabedaten einer TOMICS-Simulationsrechnung zu erhalten, wurde zunächst TOMICS entsprechend modifiziert. Damit lassen sich am Ende einer TOMICS-Simulation „Trace“-Daten generieren, die die obigen Informationen enthalten. Dabei werden drei Ereignistypen unterschieden, die den obigen Zeiten zuzuordnen sind: Ankunft, Beginn und Ende einer Bedienung. Tritt eines dieser Ereignisse auf, so werden die folgenden Daten registriert und am Ende der Simulation in der Datenbank in Form von Records abgelegt:

application:	Name des simulierten Anwendungsszenarios
exitID:	Identifikationsnummer der Bedienstelle
time:	Zeit, in der das Ereignis aufgetreten ist (T_i , B_i oder E_i)
personID:	Identifikationsnummer des Objekts (Index j), das an diesem Ereignis beteiligt ist
kind:	Art des registrierten Ereignisses. Dieser Parameter erhält den Wert 1 bei einer Ankunft und den Wert 2 oder 3 beim Beginn bzw. Ende einer Bedienung

Auf der Basis der obigen Informationen lassen sich die registrierten Zeiten als T_i , B_i oder E_i einordnen und dadurch z.B. die Wartezeit D_i des i-ten Passagiers in der Warteschlange durch $D_i = B_i - T_i$ ermitteln. Diese Werte können anschließend zur Berechnung der mittleren Wartezeit in der Warteschlange aller Passagiere anhand Gleichung (1) verwendet werden. Eine Erläuterung der Implementierungslogik zur Berechnung der Ausdrücke (2) und (3) würde den Rahmen dieses Papers sprengen. Dazu wird auf Law, A.M. & Kelton, W.D. (2000) verwiesen.

TIDE befindet sich in einigen Bereichen noch in der Entwicklung. Sobald diese Entwicklung abgeschlossen ist, steht ein wichtiges, leistungsstarkes und wirtschaftliches Tool zur Auswertung von Flugahensimulationsdaten zur Verfügung, das eine simulationsbasierte Optimierung des Terminalmanagements effizienter und komfortabler zu gestalten hilft.

Literatur

Ashford, N. (1988)
Level of service design concept for airport passenger terminals – a European view, Transportation Planning and Technology 12, 5-21, 1988

Babeliowsky, M. (1997)
Designing Interorganizational Logistic Networks. Ph.D. Thesis, Delft University of Technology, 1997

CORBA (2006)
http://www.corba.org/

Doshi, N. & Moriyama, R. (2002)
Application of Simulation Models in Airport Facility Design, In: E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes (eds.), Proceedings of the 2002 Winter Simulation Conference, Piscataway, NJ: IEE, pp. 1725-1730, 2002

Gatersleben, M.R. und van der Weij, S. (1999)
Analysis and Simulation of Passenger Flows in an Airport Terminal, In: P.A. Farrington, H.B. Nembhard, D.T. Sturrock and G.W. Evans, eds., Proceedings of the 1999 Winter Simulation Conferenc, Piscataway, NJ: IEE, pp. 1226-1231, 1999

IATA (1995)
Airport Development Reference Manual: 8th Edition. International Airline Transport Association, Montreal, 1995.

Joustra, P.E. und Van Dijk, N.M. (2001)
Simulation of Check-In at Airport. In: B.A. Peters, J.S. Smith, D.J. Medeiros and M.W. Rohrer (eds.), Proceedings of the 2001 Winter Simulation Conference, Piscataway, NJ: IEE, pp. 1023-1026, 2001

Law, A.M. & Kelton, W.D. (2000)
Simulation modeling and analysis, 3rd ed. New York: McGraw-Hill, 2000.

Snowdon, J.; El-Taji, S.; Montevecchi, M.; MacNair, E.; Callery, C.A. und Miller, S. (1998)
Avoiding the Blues for Airline Travelers, D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan (eds.), Proceedings of the 1998 Winter Simulation Conference, Piscataway, NJ: IEE, pp. 1105-1112, 1998

Suna (2006)
Sun Microsystems, Inc.: JavaTM 2 Platform, Standard Edition, v1.4.2 API Specification. http://java.sun.com/j2se/1.4.2/docs/api/, gesehen August 2006

Sunb (2006)
Sun Microsystems, Inc.: Online-Dokumentation zum JDK 1.4.2. http://java.sun.com/j2se/1.4.2/docs/, gesehen August 2006

Valentin, E. und Verbraeck, A. (2002)
Simulation Using Building Blocks for Airport Terminal Modeling, In: E. Yücesan, C.-H. Chen, J.L. Snowdon, and J.M. Charnes (eds.), Proceedings of the 2002 Winter Simulation Conference, Piscataway, NJ: IEE, pp. 1199-1206, 2002