AI Magazine Volume 21 Number 4 (2000) (© AAAI)

Stand-Allocation System (SAS)

A Constraint-Based System Developed with Software Components

Andy Hon Wai Chun, Steve Ho Chuen Chan, Francis Ming Fai Tsang, and Dennis Wai Ming Yeung

■ The stand-allocation system (SAS) is an AI application developed for the Hong Kong International Airport (HKIA) at Chek Lap Kok. SAS uses constraint-programming techniques to assign parking stands to aircraft and schedules tow movements based on a set of business and operational constraints. The system provides planning, real-time operation, and problem-solving capabilities. SAS generates a stand-allocation plan that finely balances the objectives of the airline-handling agents, the convenience of passengers, and the operational constraints of the airport. The system ensures a high standard of quality in customer service, airport safety, and use of stand resources. This article describes our experience in developing an AI system using standard off-the-shelf software components. SAS is an example of how development methodologies used to construct modern AI applications have become fully inline with mainstream practices.

Costing over US$20 billion to construct, the Hong Kong International Airport (HKIA) at Chek Lap Kok replaced the old airport at Kai Tak, which was already one of the world's busiest international airports in terms of its passenger and cargo throughput. Although there were some initial hitches when the new airport opened on 6 July 1998, operations quickly returned to normal within a week's time. Within a month, operational statistics surpassed those of the old airport—80 percent of all flights were on time or within 15 minutes of schedule, all passengers cleared immigration within 15 minutes, and average baggage waiting time was only 10 minutes. During the 1998 Christmas holiday, HKIA serviced about 100,000 passengers daily and maintained equally high service standards. In January 1999, *Travel and Leisure Magazine* awarded HKIA the Critics' Choice Award in recognition of its levels of satisfaction and praises received from travelers to Hong Kong.

The main responsibility of ensuring that the airport operates smoothly and that travelers are satisfied rests on the shoulders of the Hong Kong Airport Authority. The airport authority manages and controls all activities related to airport operations. It also has the responsibility of scheduling and managing all aircraft parking and ground movements at HKIA. On a daily basis, the airport authority assigns parking stands to aircraft based on the daily flight schedule, rotation information, and a set of operational constraints. It also schedules aircraft tows to optimize the use of inner stands. In addition, to cope with conflicts caused by changes in actual operations, the airport authority also needs to make real-time problem-solving decisions on stand reassignments.

The stand-allocation system (SAS) was designed and developed to support the airport authority's ramp-management function. Figure 1 is a snapshot of the SAS Gantt chart displaying two days of stand assignments. The system is installed and used in the Airport Control Center (ACC), which is located in the control tower. SAS provides planning, real-time management, and reactive scheduling capabilities for stand management. The system supports concurrent use by multiple operators in nonstop 24-hour-a-day operations because HKIA is a 24-hour airport.

Because all stands at HKIA (passenger and cargo) are centrally allocated and managed by

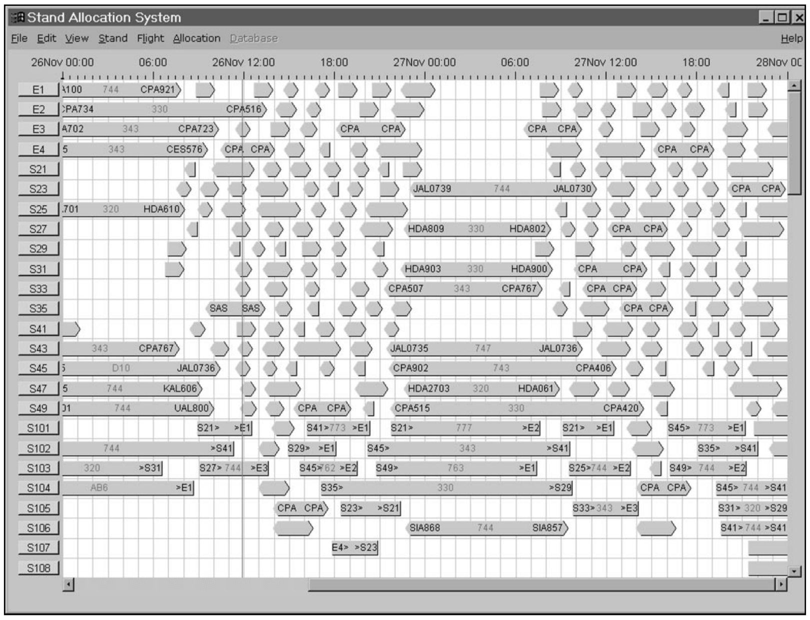


Figure 1. SAS Gantt Chart with Two Days of Assignments.

the airport authority, the main objective of SAS is to be as fair as possible to all airlines and handling agents while it makes efficient and safe use of airport resources. An efficient stand-allocation plan can maximize the use of stands and, thus, permit additional flights during peak traffic hours and holiday seasons. Although HKIA was designed to have more stands than the old airport, not all stands are operational yet; part of the airport terminal building is still under construction. Furthermore, air traffic will most likely increase once the second runway becomes operational. Optimization of stand assignment is still an important factor at the new airport.

Another objective is to provide better service and comfort to passengers. For example, assigning aircraft closer to the proximity of the immigration counters will allow passengers to quickly exit the terminal. However, assigning flights close to airline service counters will convenience transit passengers.

Maintaining some degree of consistency or patterns in allocation from week to week or day to day is also very important, especially for regular flights. Thus, airlines and handling agents will be able to perform longer-term macroplanning with more accuracy and a certain degree of independence.

Because the airport is a 24-hour airport, stand-allocation planning will be performed at the same time as normal operations. The ability of the scheduling system to coordinate multiple sets of data within a multiuser environment is also very important.

The quality of a stand-allocation plan is determined by many factors—how efficient resources are being used; how convenient it is to the passengers; and, most importantly, whether all safety-related criteria are met. Typically, a human operator must have several years of experience to acquire enough knowledge about airport operations before he/she can produce a "good" quality stand-assignment plan. Generating an allocation plan manually not only requires a highly experienced individual but is also very time consuming because it requires balancing many objectives against many possible alternatives.

Another key problem that is particularly difficult to perform manually is coping with conflicts caused by operational changes. For example, flights might be delayed, aircraft might be swapped, or there might be sudden ground equipment failures. These events can invalidate previous stand assignments and affect assignments of other aircraft. In other cases, mechanical failures or other problems might occur after chocking off, and the aircraft might need to return to the airport as an air return or ground return. The ability to handle changes and events such as these quickly and intelligently and minimize impact on the rest of the committed schedule is vital to ensuring that the airport operates smoothly.

An AI solution allows a highly optimized allocation plan to be produced in a reasonable time, ensures that all operational constraints are considered all the time, and performs problem solving in real time or close to real time. On average, SAS produces a daily stand-allocation plan in about three minutes and performs reactive problem solving in five seconds at HKIA using a PENTIUM II server machine.

Application Description

SAS is designed for planning, real-time operations, and data analysis. Stand-assignment planning is usually performed after midnight when air traffic is light and when all the airlines have finalized their rotation changes. SAS takes the daily flight schedule for the day being planned and generates an optimized stand allocation using an algorithm for constraint-satisfaction problems (CSPs). Our scheduling algorithm considers constraints related to arrival-departure times, type of flight, aircraft type-size, airline preferences, stand configurations, and so on, in producing the stand-allocation plan. SAS also provides a set of menus and commands to allow the operator to enter any last-minute changes before confirming and distributing the plan. Figure 2 is an example of an SAS to modify flight information. The operator performing stand planning uses a separate SAS workstation dedicated for system administrative work. Other live SAS workstations load the new plan after it is finalized.

During real-time operations, SAS operators monitor and enter up-to-date flight statuses such as estimated times of arrival-departure (ETA-ETD), actual chock on-off times, and aircraft registration. Different types of information and status are displayed using various icons, symbols, and color-coded shadings in SAS. SAS automatically reevaluates all constraints whenever any new event occurs. The main screens used by operators are in the form of a Gantt chart and spreadsheets. In real-time operations, several operators and SAS workstations are involved to handle different types of information. SAS was designed for multiuser operations and ensures a consistent display on all SAS workstations. All stand-related data and information are archived for later analysis, auditing, or report generation.

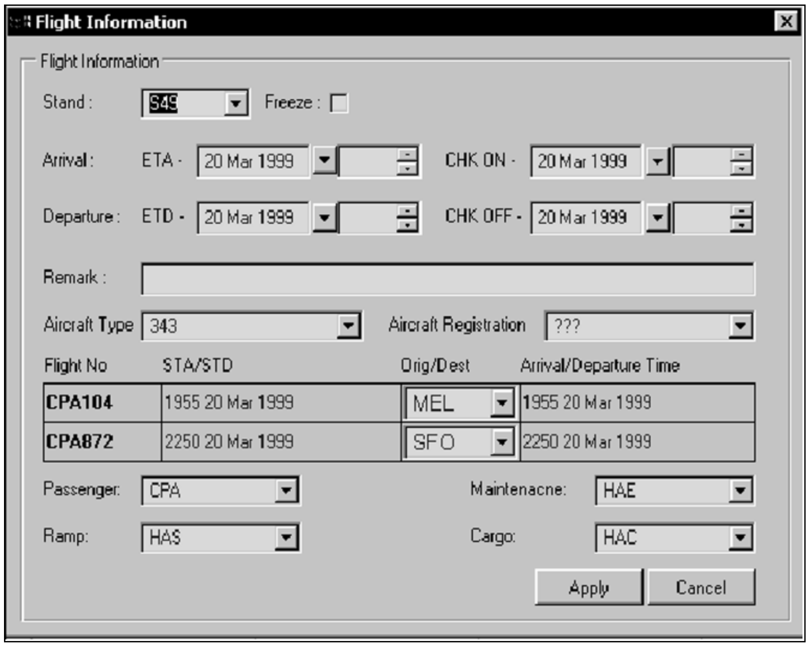
Whenever an event causes a "conflict," SAS alerts the operator and highlights the conflicting assignments in red. For example, a delay in departure might prevent the next aircraft from parking, or a delay in arrival might prevent adequate time to deplane passengers before a scheduled tow. SAS has explanation-generation capabilities and can give justification on assignment penalties and reasons for conflict. Once a conflict occurs, the operator can request SAS to automatically resolve the conflict using a reactive scheduling algorithm that solves conflicts while it minimizes the impact on the current plan.

SAS was designed to supplement the centralized airport system, known as the terminal management system (TMS). TMS has basic stand-assignment support but lacks the constraint-based intelligence and the scope of stand-related functions provided by SAS. Mission-critical AI applications, such as SAS, usually have two orthogonal sets of design criteria to satisfy: (1) design criteria related to the AI implementation and (2) criteria related to the information technology implementation.

From the AI point of view, the technologies selected and used for SAS must satisfy the following criteria:

Robust: The knowledge representation used must be robust enough to capture all types of knowledge related to stand allocation. The representation must be able to capture knowledge on physical dimensions, geometry and layout, proximity, and operational requirements.

Transparent: The logic or reasoning mechanism used must be easy to understand and, hence, should be reasonably similar to the logic used by human operators. The system should be able to give explanation on its actions to improve transparency.



Flight Information

Stand : S49 Freeze : ☐

Arrival : ETA - 20 Mar 1999 CHK ON - 20 Mar 1999

Departure : ETD - 20 Mar 1999 CHK OFF - 20 Mar 1999

Remark :

Aircraft Type 343 Aircraft Registration ???

Flight No	STA/STD	Orig/Dest	Arrival/Departure Time
CPA104	1955 20 Mar 1999	MEL	1955 20 Mar 1999
CPA872	2250 20 Mar 1999	SFO	2250 20 Mar 1999

Passenger: CPA Maintenacne: HAE

Ramp: HAS Cargo: HAC

Figure 2. SAS *Menu to Display-Modify Flight Information.*

Optimizing: The reasoning mechanism must be able to optimize on a given set of potentially contradicting constraints and criteria because the airport authority needs to satisfy the needs of many different parties—airlines, handling agents, passengers, and the airport itself.

Fast: The reasoning mechanism must be highly efficient to be able to consider several hundred constraints for each assignment. There can be several hundred assignments a day.

Problem solving: Because the airport is a highly dynamic environment, the AI component must be able to perform problem solving to resolve conflicts during real operations.

From the information technology point of view, SAS was designed to operate as a mission-critical application, thus adding requirements on top of those related to AI and scheduling. The following highlights some of the key information technology considerations:

Real-time performance: The term *mission critical* implies that SAS response time must be close to real time and must be able to cope with real-time changes quickly. The implementation technology selected must be highly efficient. Interpreted programming languages might perform poorly in this respect. Multithread support is a must.

Multiuser support: Most mission-critical operations will require several human operators

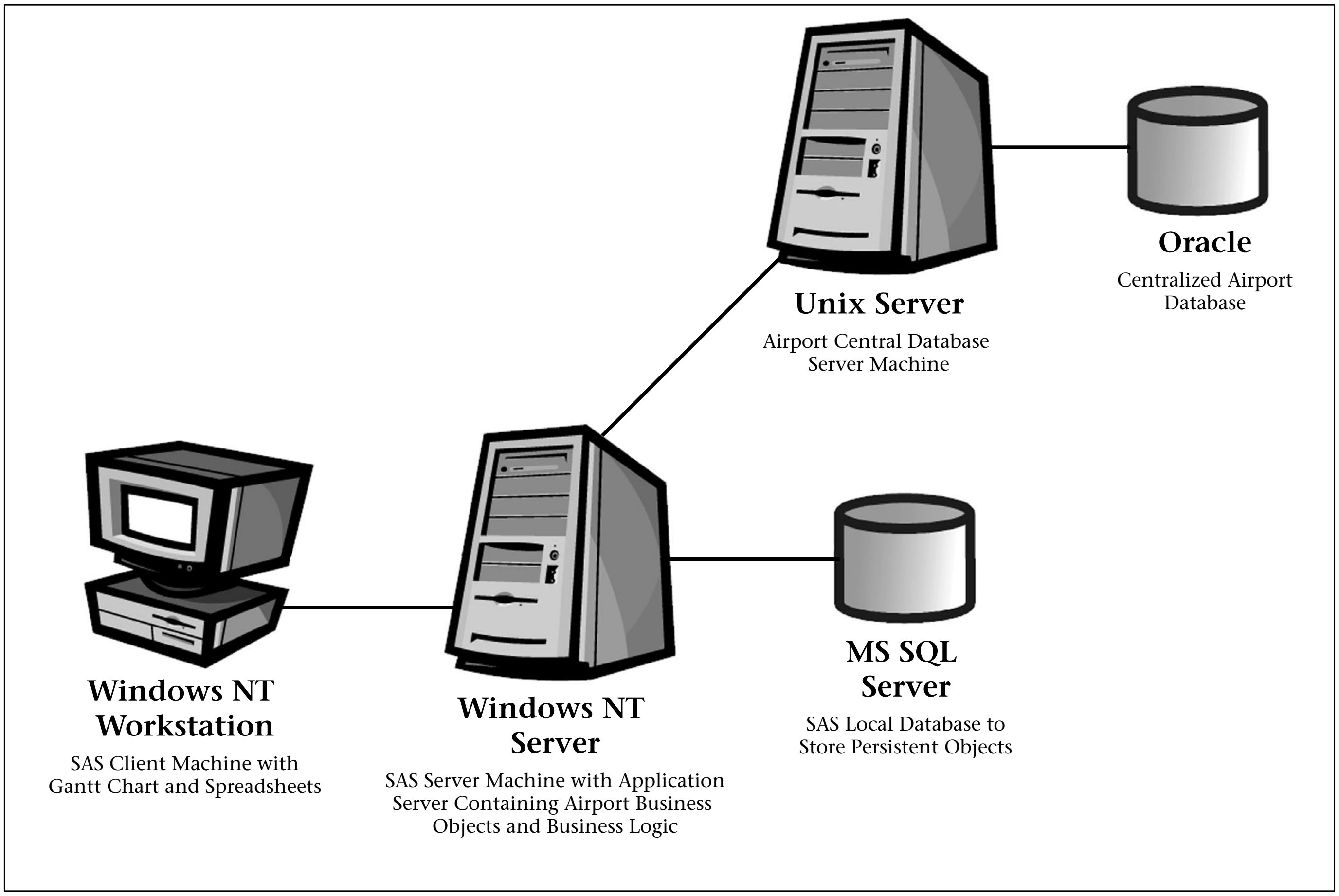

Unix Server
Airport Central Database Server Machine
Oracle
Centralized Airport Database
Windows NT Workstation
SAS Client Machine with Gantt Chart and Spreadsheets
Windows NT Server
SAS Server Machine with Application Server Containing Airport Business Objects and Business Logic
MS SQL Server
SAS Local Database to Store Persistent Objects

Figure 3. The SAS *Hardware Architecture.*

cooperating together. The system architecture must support the simultaneous use by several operators and be able to synchronize all clients' machines in real time. The implementation platform must support some form of asynchronous messaging or callback.

Scalability: Although the actual number of SAS operators who will be making decisions will be limited, the number of users that need to access information from the system might eventually be large. The technology must be able to scale up to support a large number of potential users.

Fault tolerance: Mission critical of course implies the system must be available close to 100 percent of the time. The technology selected must be able to support an architecture that is fault tolerant with hardware and software redundancy and switchover capabilities.

Load balancing: Although not really necessary for the stand-allocation problem, many mission-critical systems will require some form of load balancing to improve performance.

Interoperability: Any mission-critical system will need to interact with a set of other systems to exchange data or request services, including the ability to easily access potentially different types of relational or object database. The degree of interoperability depends on the technology selected.

A Solution Based on Object Technology

Based on the AI and information technology requirements, we decided to select a technology-methodology that can address all these issues at a broad level. We needed a technology that can be applied to all components within our software architecture. To meet all the rigid requirements of an intelligent AI system that is also mission critical, we finalized on a three-tiered distributed object architecture based on CORBA.

There is, of course, a lot of flexibility in selecting technologies if we are designing a stand-alone single-tiered system. There is less flexibility but still quite a lot if we are designing a two-tiered system. However, when it comes to developing a multiuser *n*-tiered architecture, the choices must be carefully made. The fol-

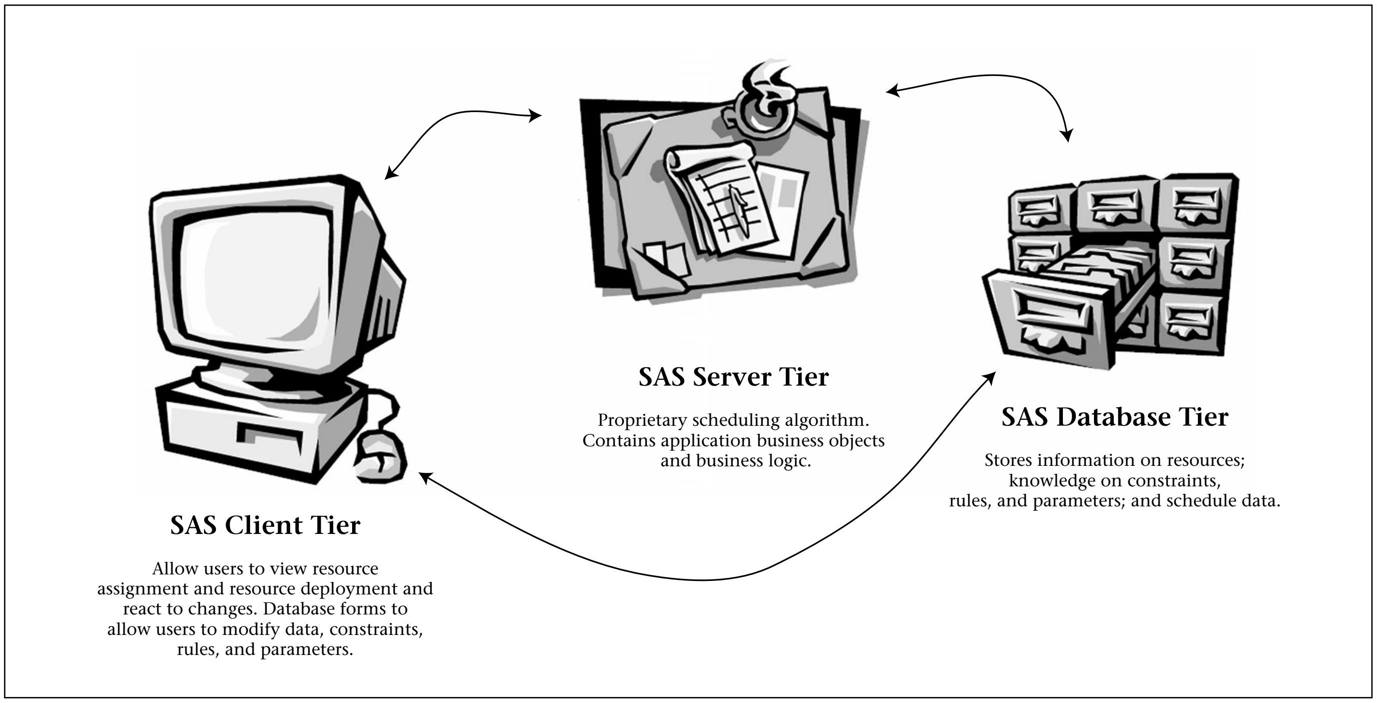

SAS Server Tier
Proprietary scheduling algorithm. Contains application business objects and business logic.
SAS Database Tier
Stores information on resources; knowledge on constraints, rules, and parameters; and schedule data.
SAS Client Tier
Allow users to view resource assignment and resource deployment and react to changes. Database forms to allow users to modify data, constraints, rules, and parameters.

Figure 4. The SAS Three-Tiered Software Architecture.

lowing discussion highlights some of the rationale in the technologies used to construct the SAS application.

SAS was designed to operate within a Microsoft NT environment with its own NT server and local Microsoft SQL server database. Figure 3 is a diagram of the SAS hardware architecture. It interfaces to other external server machines to exchange data, such as seasonal schedules, rotation information, and stand assignment.

To provide real-time performance, SAS was designed as a three-tiered architecture with an in-memory persistent business object cache within the application server. Figure 4 shows the three-tiered software architecture used by SAS. We selected an object technology approach using C++ as the implementation language because of its efficiency and the availability of numerous off-the-shelf software components.

SAS uses constraint programming as the foundation for the AI component because the stand-allocation problem can easily be modeled as a constraint-satisfaction problem. Constraints provided in constraint programming were expressive enough to capture all types of knowledge required for stand allocation. Furthermore, constraint-programming capabilities were available as third-party C++ components.

Typical to most object technology–based systems, SAS uses four main types of component software: (1) middleware components, (2) business objects, (3) business logic, and (4) graphic user interface (GUI) components (figure 5). Middleware components are used in both client and server processes. The airport business objects are implemented as CORBA objects, where the implementations reside on the server and then are distributed to clients as proxies. The business logic used for stand allocation is stored in the server process only. The user interface components are used mainly for the client GUI. Using an object technology approach, we were able to take advantage of commercially available best-of-breed off-the-shelf component software to support the functions required by SAS.

The Middleware Components

SAS uses two types of middleware component: (1) a CORBA middleware and (2) a database middleware. We selected CORBA as the core middleware technology for SAS because it has a relatively low performance and implementation overhead. CORBA also has a broad range of services and is readily available on many platforms. To provide for multiuser support, we used push-type asynchronous messaging. For scalability, the asynchronous messaging can easily scale up to use IP MULTICAST. SAS uses an off-the-shelf CORBA object request broker (ORB) for development.[1,2,3,4]

The SAS application server generates a finite set of events to notify clients of changes and

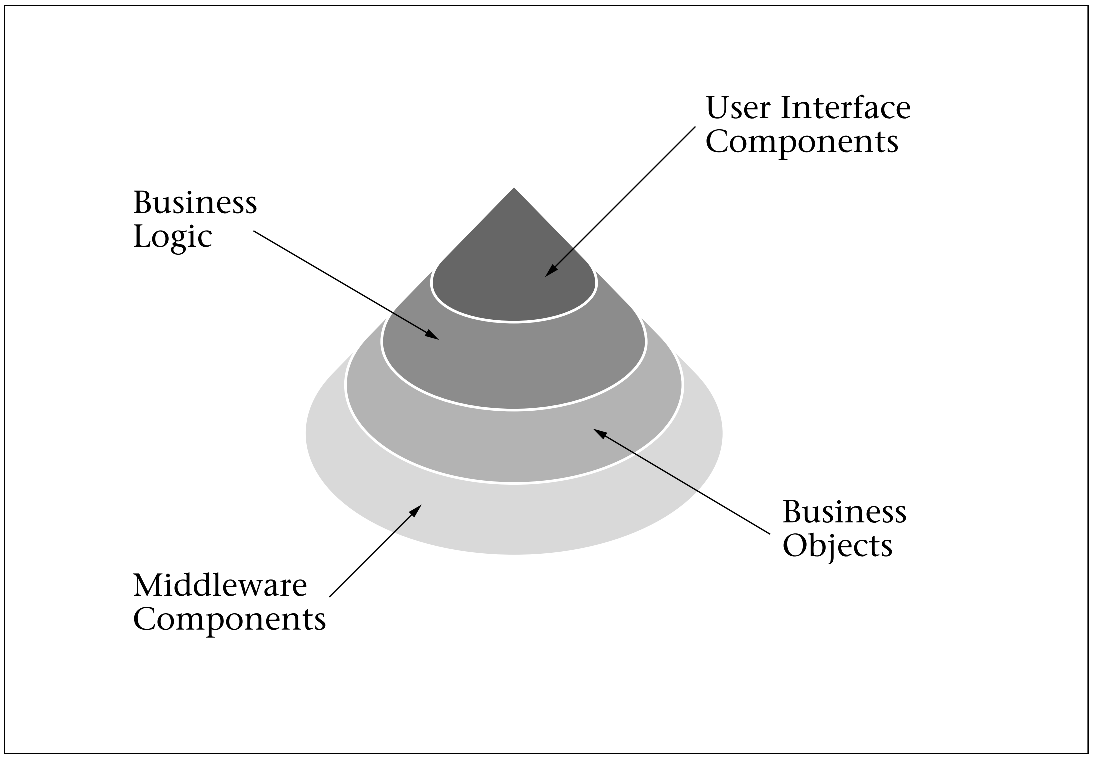


Figure 5. Four Types of Component Software Used by SAS.

updates. These events are distributed using the CORBA event services (figure 6). This infrastructure gives us the added advantage of potentially using the same mechanism to notify other information technology systems at the airport of stand-related events, such as stand-gate changes or stand closures.

Most database vendors readily support some form of fault tolerance at the third tier. Fault tolerance in the second tier can easily be handled by the CORBA middleware itself. Several CORBA middleware products also support load-balancing techniques. SAS also uses a database middleware to insulate the system from the databases we communicate with. SAS uses off-the-shelf software components to provide object-oriented access to relational databases.[5]

The Business Objects

We were at an advantage in building the SAS application because most of the SAS business objects were reused from an earlier system we had built for the old Kai Tak Airport. Business objects represented entities such as flight legs, aircraft, stands, airlines, and handling agents. These business objects were developed on top of our proprietary OPTIMIZ! scheduling framework that was implemented using off-the-shelf foundation classes.[6] The business objects are made "constraint aware" by incorporating C++ software components that supported constraint programming (Kumar 1992; Cohen 1990; Van Hentenryck 1989; Steele 1980). In addition, the whole system was made multithread safe using third-party multithread components.[7]

Because all key business objects in SAS are packaged as CORBA objects, there is a great degree of interoperability with other systems using the ORB or simple bridging techniques. Although not all systems at HKIA are CORBA based, a CORBA "wrapper" can easily be added to other information technology systems to facilitate integration, as illustrated in figure 7.

The SAS in-memory business object cache is created from a local Microsoft SQL server relational database using the database middleware. The database schema design is identical to that of the HKIA's airport operational database (AODB). All relational tables relevant to ramp operations are mirrored and updated regularly from the AODB. Business objects are made persistent by having any changes, triggered by the SAS event mechanism, be automatically committed to the database through the database middleware.

The Business Logic

SAS business logic consists mainly of constraints defined using our prebuilt C++ OPTIMIZ! scheduling framework. Our framework provides a soft-

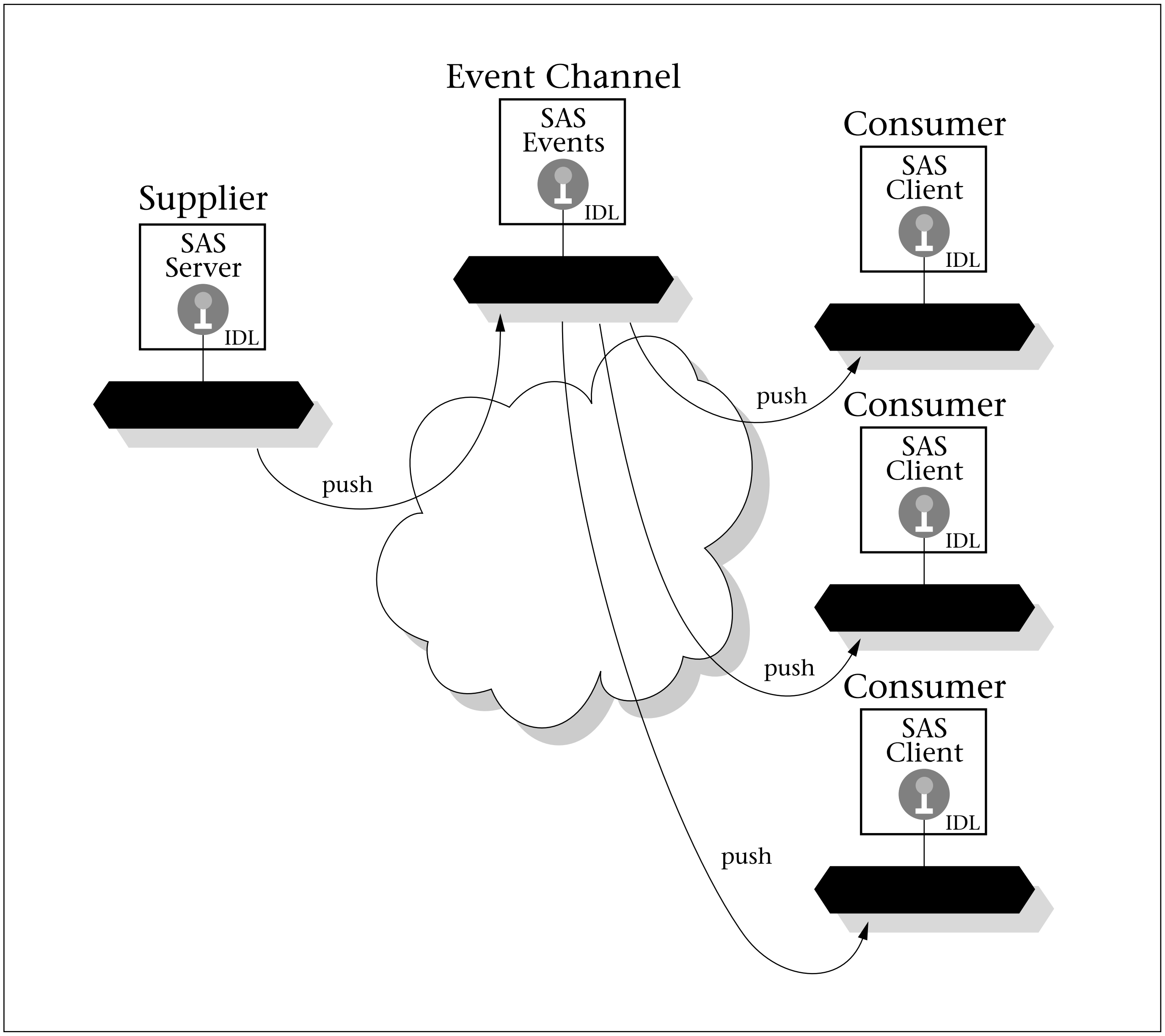


Figure 6. CORBA *Event Services Push Updates to Clients for Multiuser Support.*

ware infrastructure on which a general class of scheduling systems can be built. The framework contains features to facilitate problem modeling and the scheduling of algorithm implementations. It contains software components to represent generalized concepts such as allocatable objects, hard constraints, soft constraints, and a set of scheduling algorithms. Many generic allocation operations are built into the framework, such as freeze, move, cancel, split, and swap. Other facilities include reactive scheduling, what-if analysis, explanation generation, warning message generation, audit-trail logging, an event generator, and an auto-testing facility.

The OPTIMIZ! framework has been used for other projects and was ready, off the shelf, prior to SAS development. As part of actual SAS development, we defined constraints related to stand allocation using the framework and established a set of constraint parameters that reflected actual operations at Chek Lap Kok.

The User Interface Components

The GUI for SAS client machines was developed using highly optimized C++ graphic components.[8] The user interface consists mainly of interactive Gantt charts and spreadsheets. Additional administrative user interface facilities were developed using Microsoft VISUAL

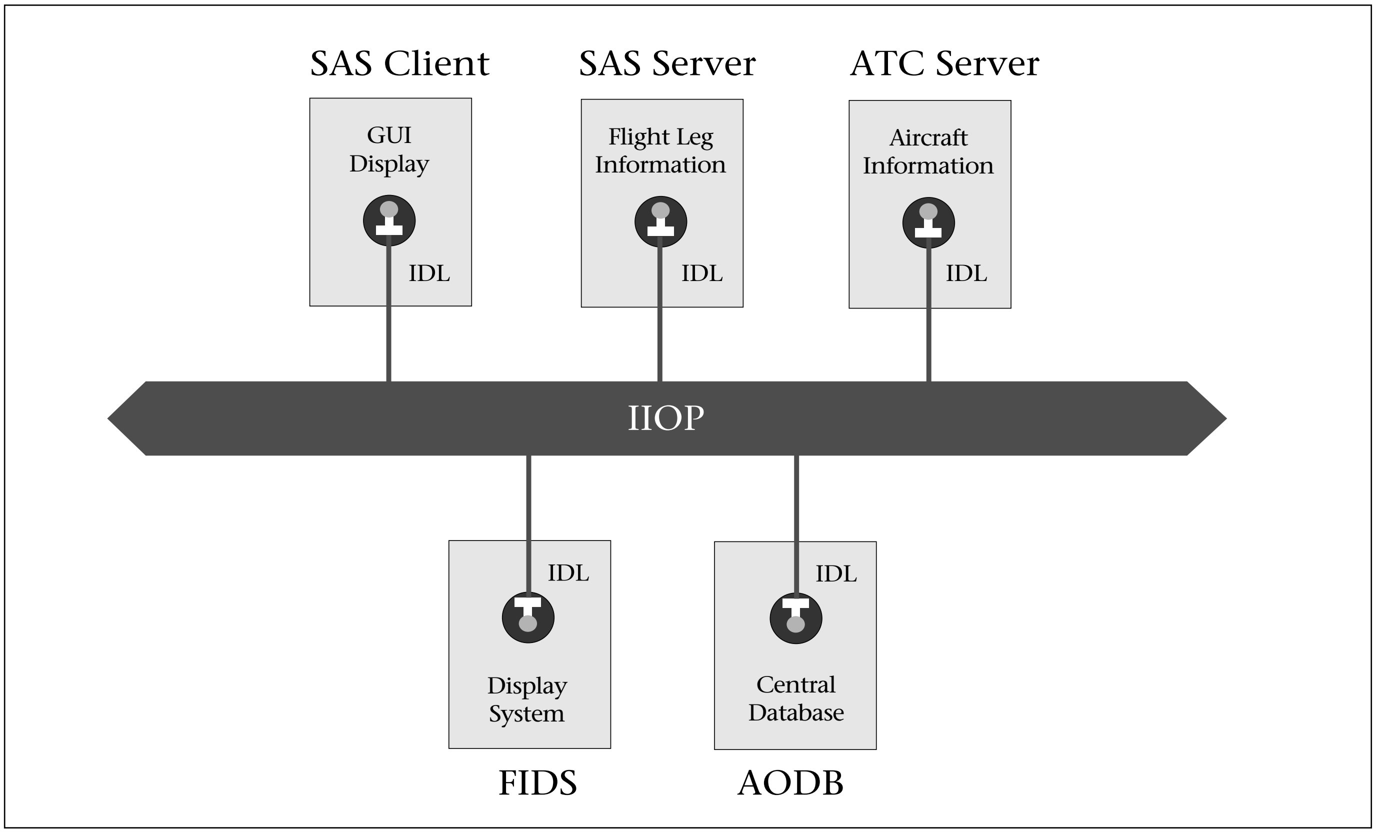


Figure 7. SAS *as a* CORBA*-Based System.*

BASIC. All SAS user interface screens follow standard WINDOWS interaction with context-sensitive menus and online help, thus making the system very user friendly. SAS training only takes a day's time for operators who are already familiar with airport operations.

Putting It All Together

Figure 8 documents the final system architecture once all the software components are in place and integrated into the CORBA infrastructure. This diagram represents an orchestration of many software components working efficiently in unison. As an architecture for AI applications, it is highly elegant and is made possible only through the combination of highly efficient C++, CORBA, and constraint programming.

Uses of AI Technology

The problem of stand allocation is a typical resource-assignment problem that can be solved efficiently as a CSP (Kumar 1992; Cohen 1990; Van Hentenryck 1989; Steele 1980). CSP algorithms have been used successfully to solve a wide variety of transportation-related scheduling problems (Puget 1994a, 1994b). In general, scheduling and resource-allocation problems can be formulated as a CSP that involves the assignment of values to variables subjected to a set of constraints.

For the stand-allocation problem, each variable represents the stand assignment for one aircraft. Each aircraft can be associated with several variables because an aircraft might need to be towed several times during its stay at the airport. The domain of each variable will initially contain the set of all stands in the airport. The CSP constraints are restrictions on how these stands can be assigned to an aircraft. The same set of constraints might be used during reactive scheduling to solve dynamic problems.

SAS uses mainly two main types of constraint: (1) hard constraints for domain reduction and (2) soft constraints for value selection. These constraints are implemented using constraint components from our OPTIMIZ! scheduling framework. The key constraints for stand allocation are outlined as follows:

No overlap constraint: This constraint ensures that no two aircraft will be assigned the same stand at the same time. This time is measured between the chock-on and chock-off times and a "clearance" to allow for aircraft movements

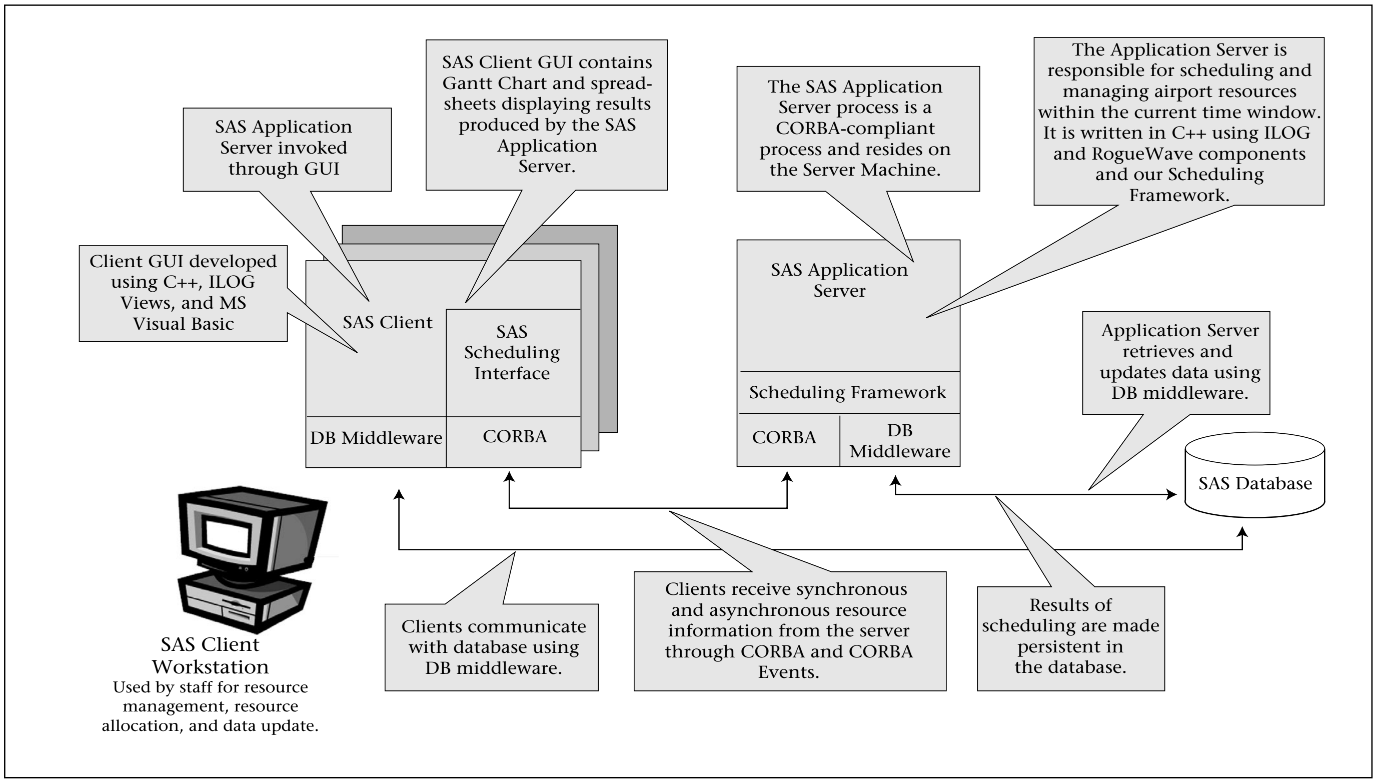

SAS Application Server invoked through GUI
SAS Client GUI contains Gantt Chart and spread-sheets displaying results produced by the SAS Application Server.
The SAS Application Server process is a CORBA-compliant process and resides on the Server Machine.
The Application Server is responsible for scheduling and managing airport resources within the current time window. It is written in C++ using ILOG and RogueWave components and our Scheduling Framework.
Client GUI developed using C++, ILOG Views, and MS Visual Basic
SAS Client
SAS Scheduling Interface
DB Middleware
CORBA
SAS Application Server
Scheduling Framework
CORBA
DB Middleware
Application Server retrieves and updates data using DB middleware.
SAS Database
SAS Client Workstation
Used by staff for resource management, resource allocation, and data update.
Clients communicate with database using DB middleware.
Clients receive synchronous and asynchronous resource information from the server through CORBA and CORBA Events.
Results of scheduling are made persistent in the database.

Figure 8. Distribution of Software Components through CORBA *Infrastructure.*

in and out of the stand.

Stand-combination constraint: Some stands can be combined with adjacent or nearby stands to accommodate larger aircraft or divided into several stands to accommodate smaller aircraft. This constraint ensures that these stand combinations are considered during allocation and that only one combination is active at any one time.

Passenger-freighter aircraft constraint: Cargo and passenger flights have different constraints on where the aircraft can be allocated. For example, an airport can disallow any passenger flights from deplaning or boarding at cargo areas.

International-domestic constraint: Some stands can be dedicated to serving international or domestic flights only. In Hong Kong, despite the fact it is now SAR China, all flights are still considered as international.

Stand-closure constraint: Stands can be closed from time to time for maintenance. This constraint ensures that no aircraft is assigned a stand when it is closed.

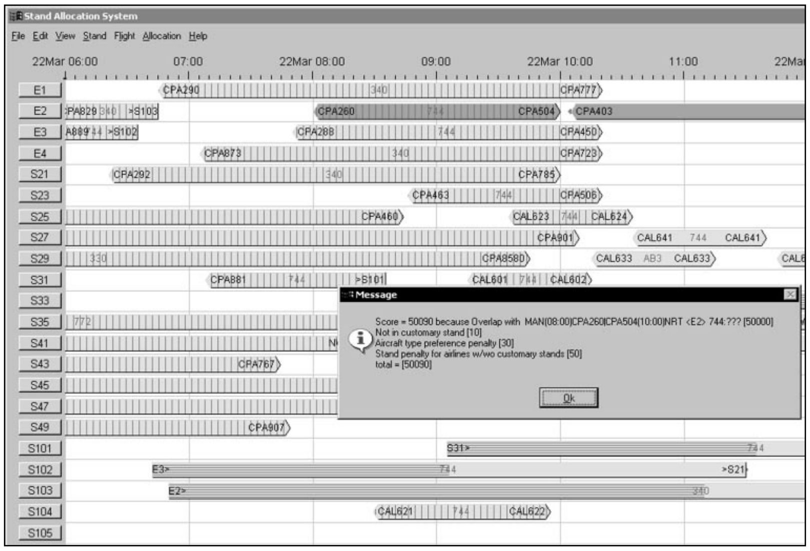
Stand-warning constraint: Sometimes stands can have minor equipment failures that restrict certain types of aircraft from parking there. For example, if one of the bridges is down, a smaller aircraft type can still use the stand for deplaning and boarding.

Size constraint: This is a physical constraint that ensures the assigned stand is large enough to fit the aircraft. In Hong Kong, most stands are large enough to fit any type of aircraft.

Adjacency constraint: Sometimes the size of an aircraft can affect which type of aircraft can use the adjacent stands. For example, a wide-body aircraft might prohibit another wide body from being assigned to adjacent stands. At CLK, there are no adjacency constraints for this new airport.

Aircraft-type preferences: These constraints are soft preferences on which stands should be assigned to which aircraft types, which might be for convenience of aircraft maneuvering, convenience of passenger, or ground-equipment availability.

Auto-tow constraint: This constraint determines when aircraft should be towed and where it should be towed. It ensures that aircraft with long ground time should be towed to temporary parking areas to optimize use of inner stands. For example, aircraft staying on ground for several hours will be towed to remote stands and then towed back for departure. Longer staying aircraft, however, will be towed to maintenance stands, which might be farther away. The tow times are defined by several parameters: the minimum time needed to offload passengers, the minimum time an air-



Score = 50090 because Overlap with MAN(08:00)CPA260|CPA504(10:00)NRT <E2> 744:??? [50000]
Not in customary stand [10]
Aircraft type preference penalty [30]
Stand penalty for airlines w/wo customary stands [50]
total = [50090]

Figure 9. Explanation Facility Provided in SAS.

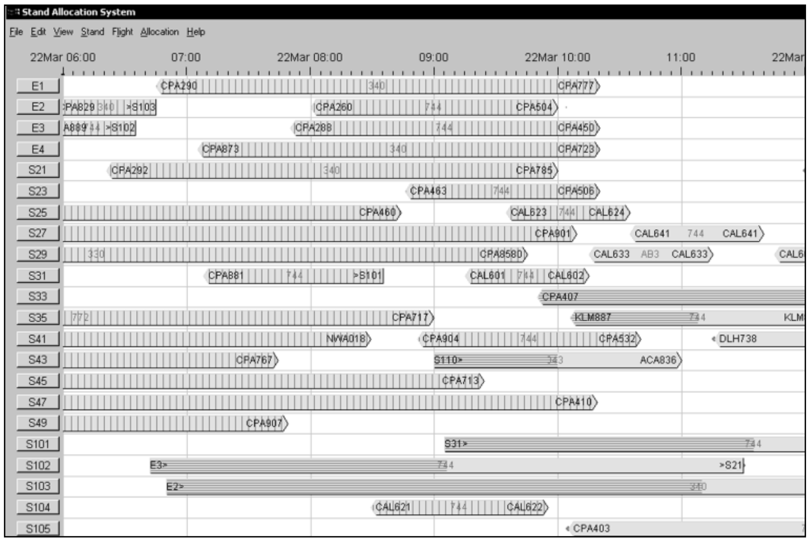


Figure 10. Result of Performing Reactive Scheduling.

craft should be assigned to a stand, and the minimum time needed for boarding, and the towing time from stand to stand.

Towing preferences: This constraint defines preferences about where aircraft should be towed for temporary parking. For example, certain areas can be more desirable to be used as "pad" areas. However, there might be a simple preference to just tow the aircraft to the nearest outer stand in terms of towing time.

Customary stand preferences: Airlines or handling agents can have preferences about which stands their aircraft should be assigned to. These preferences are usually related to the location of equipment or the transit or service counters of the airline or handling agent.

Aircraft orientation constraint: Aircraft assigned to remote stands can have different options about the parking orientation of the aircraft. This constraint can be related to convenience of aircraft maneuvering or safety of deplaning passengers at nearby stands. Unlike the old airport, at CLK, all aircraft parking orientations are fixed.

Connecting passenger constraint: This constraint tries to optimize the allocation of aircraft such that flights with transit passengers will be assigned closer together to minimize passenger walking time.

Whenever constraint violations reach a given threshold, the flights involved are highlighted in red. Figure 9 illustrates the delay of a flight's departure (CPA504), which causes a conflict with the next flight (CPA403) that is scheduled for the same stand (E2). There is not enough time for the departure flight to maneuver out of the stand to make room for the arriving flight. The user can invoke the SAS why command to get an explanation for the constraint scoring. Figure 9 shows how different shadings and patterns are used to indicate different flight statuses, such as confirmed chock on-off and new estimated times of arrival and departure (ETA-ETD).

SAS automatically solves conflicts with the click of a button. It takes less than a minute to solve all conflicts for one day. Figure 10 shows the new schedule after a conflict has been resolved by SAS. The conflicting flight (CPA403) has been moved to stand S105, which is a less desirable remote stand. Our reactive scheduling algorithm solves conflicts and minimizes the impact of changes to the original schedule, thus reducing the amount of inconvenience to waiting passengers.

Application Use and Payoff

SAS is used by airport authority airfield operation staff in the ACC at HKIA. Roughly 20 operators, who work in shifts, were trained on the use of SAS. The system has been deployed since June 1998. Because HKIA is a new airport, quantitative measurements on performance before and after system implementation cannot be made. However, we can still clearly identify the key benefits expected from using SAS:

A dynamic organization: Producing an optimized stand-allocation plan manually will take at least half a day. The quality of the resulting allocation plan will vary from person to person depending on experience. SAS produces a plan in roughly three minutes and ensures that all constraints are considered, the allocation plan is optimized, and high quality is maintained each time. Because scheduling time is reduced to only a few minutes, the airport authority becomes more dynamic as an organization and can handle last-minute flight and rotation changes quickly.

Swift decision making: Humans might not perform well under conditions of extreme stress and pressure such as those faced during real operations. Trying to perform problem solving under these conditions can result in less than ideal or even wrong decisions. SAS performs reactive problem solving in about five seconds and guarantees that the solutions are correct each and every time. A quick response time allows SAS operators to reply to air traffic controllers immediately, while online if needed.

Guaranteed safe and smooth operation: Because all constraints are considered all the time, SAS guarantees that no safety-related constraints are overlooked. Over 100 different types of aircraft land in Hong Kong. Each aircraft has a slightly different physical dimension and equipment requirement. Bridges normally can accommodate many types of aircraft. However, certain bridge configurations or equipment failures can restrict some types of aircraft from using the bridge. There is a potential that a human operator might oversee these differences and mistakenly assign an aircraft to an improper stand. The consequence might be a disruption of aircraft ground traffic or, worse, a minor collision. Even if the safety violation was identified early on, towing the aircraft to another stand will delay the plane by at least half an hour. The plan produced by SAS ensures that aircraft ground movements are safe.

Better passenger service: Being recognized as one of the best airports in the world in terms of traveler satisfaction is not easy. Many different factors contribute to this success. Stand management plays an important role. For example, a good allocation will allow passengers to get to their destination as quickly as possible. For arrival flights, SAS tries to assign aircraft close to immigration counters or unmanned transport vehicles to allow faster exit from the terminal building. For transit flights, SAS tries to assign aircraft close to airline transit counters. If a stand change must be made, SAS tries to reassign aircraft to another stand in close proximity to the original assignment.

Capacity for growth: In the long term, SAS will allow HKIA to continue to grow and accommodate more traffic in the coming years. Although HKIA is a new airport with more resources, it is still expanding. A new runway is due to begin operations soon, and air traffic will increase. However, construction of the second terminal building has yet to begin. It is most likely that the airport will gradually become resource stressed in the years before the second terminal opens. SAS will be able to help reduce this stress by optimizing the utilization of stand resources.

Application Development and Deployment

One of the advantages of implementing an AI application using standard off-the-shelf object-oriented software components is that standard object-oriented software-engineering practices can also be followed. For SAS development, we followed an object-oriented methodology that is based on Rational's UNIFIED PROCESS. User requirements were documented using use-case analysis, and the object-oriented design was documented using standard UNIFIED MODELING LANGUAGE (UML).

The coding of SAS was performed in iterative cycles to reduce risk. Development time allocated for SAS was extremely tight. To minimize risk, we planned the iterative cycles very carefully. The first iteration took only two weeks. The objective of the first development cycle was to implement an initial prototype that tested the integration of all the software components working together within the CORBA infrastructure. This process involved hooking up software components from several third-party vendors and ensuring the whole infrastructure was solid and working from end to end.

The second iteration took another month and a half, and the result was the first release of SAS with basic stand-allocation capabilities. This version was used to test the completeness of the standing allocation knowledge. By integrating off-the-shelf software components, we were able to dramatically shorten our development time. We were able to further shorten the development time by reusing business objects and an object-oriented scheduling framework that we had developed earlier for the old airport.

A third iteration took another month and encoded the remaining user requirements. This version was then under extensive testing and trial use by airport authority operators with actual data. Because time allocated for testing was short, automatic testing software was built that automatically simulated thousands of different operational scenarios. This test suite was particularly useful early on in the project to identify coding errors that would have taken weeks to find if tested manually; some coding errors can only be revealed under peculiar sequences of events. The coding of SAS took roughly 4 months elapsed time total with a development team of 10 C++ software developers who were already familiar with all the software components used in this project.

Maintenance

SAS was designed to be fully maintainable by the airport authority's own staff members. AA senior operators maintain the knowledge base through a set of Microsoft VISUAL BASIC database forms. All site-specific knowledge, constraints, parameters, and data are stored in the local Microsoft SQL server database. Any change in domain knowledge can be performed without any SAS source code change. Knowledge related to stand operations does not really change that often. The knowledge base is usually updated only when a new stand, airline, or aircraft type is put into operation. Other routine database maintenance is performed by the aiport authority's information technology staff members. In addition, we provide 24-hour technical and user support both through telephone and on-site visit if needed.

Conclusion

This article provided an overview of the stand-allocation system that we have designed and built for the new Hong Kong International Airport at Chek Lap Kok. It described the hardware and software architecture of the SAS and the constraints of the new airport. It documents our rationale in selecting the technologies we did and our experience in constructing an advanced AI application using off-the-shelf software components. Hopefully, this article provides some insights into the design and development of mission-critical component-based AI systems.

Acknowledgments

The authors would like to thank airport authority operations and information technology staff for the tremendous cooperation, support, and enthusiasm received throughout this project.

Notes

1. www.iona.com
2. www.inprise.com
3. www.ooc.com
4. www.roguewave.com
5. www.roguewave.com
6. www.roguewave.com
7. www.roguewave.com
8. www.ilog.com

References

Cohen, J. 1990. Constraint Logic Programming. *Communications of the ACM* 33(7): 52–68.

Kumar, V. 1992. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine* 13(1): 32–44.

Puget, J.-F. 1994a. A C++ Implementation of CLP. In *ILOG Solver Collected Papers*. Paris, France: ILOG SA.

Puget, J.-F. 1994b. Object-Oriented Constraint Programming for Transportation Problems. In *ILOG Solver Collected Papers*. Paris, France: ILOG SA.

Steele, Jr., G. L. 1980. The Definition and Implementation of a Computer Programming Language Based on Constraints, Ph.D. dissertation, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

Van Hentenryck, P. 1989. *Constraint Satisfaction in Logic Programming*. Cambridge, Mass.: MIT Press.

Andy Chun is an associate professor in the Department of Electronic Engineering at the City University of Hong Kong and chief executive officer of BonVision Technology (Hong Kong) Ltd., formerly Advanced Object Technologies Ltd. His interests include software architectures, object technologies, AI, optimization, and constraint programming. He received a B.S. from the Illinois Institute of Technology and an M.S. and a Ph.D. in electrical engineering from the University of Illinois at Urbana-Champaign. His e-mail address is eehwchun@cityu.edu.hk.

Dennis Yeung is chief technical officer of BonVision Technology (Hong Kong). His interests are in internet technologies, infrastructure design, software architectures, distributed object technology, and database systems. He received a B.S. in information technology at the City University of Hong Kong. His e-mail address is dennis.yeung@aotl.com.

Francis Tsang was a software consultant at Advanced Object Technologies Ltd. His interests are in user interface design and internet Java development. He received a B.S. in information technology at the City University of Hong Kong.

Steve Chan is now a senior e-commerce project manager at iSquare Asia. Previously, he was a senior software consultant at Advanced Object Technologies Ltd. His interests are in internet technologies, user interface design, optimization, and distributed objects. Chan received a B.S. in information technology at the City University of Hong Kong. His e-mail address is steve.chan@isquare-asia.com.