HKIA SAS: A Constraint-Based Airport Stand Allocation System Developed with Software Components

Andy Hon Wai Chun

City University of Hong Kong
Department of Electronic Engineering
Tat Chee Avenue, Kowloon
Hong Kong
eehwchun@cityu.edu.hk

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

Advanced Object Technologies Limited
Unit 602A, HK Industrial Technology Centre
72 Tat Chee Avenue, Kowloon
Hong Kong
{steve, francis, dennis}@aotl.com

Abstract

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 airlines/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 utilization of stand resources. This paper also 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.

Task Description

Costing over US$20 billion to construct, the Hong Kong International Airport (HKIA) at Chek Lap Kok replaces 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% 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 1998's Christmas holiday, HKIA serviced around 100,000 passengers daily while maintaining equally high service standards. In January 1999, Travel & Leisure Magazine awarded HKIA the Critics' Choice Award in recognition for the levels of satisfaction and praises received from travelers to Hong Kong.

The main responsibility of ensuring that the airport operates smoothly and that travellers are satisfied rests upon the shoulders of the Hong Kong Airport Authority (AA). 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 optimise the use of inner stands. In addition, to cope with conflicts caused by changes in actual operations, AA also needs to make real-time problem solving decisions on stand reassignments.

The Stand Allocation System (SAS) was designed and developed to support AA's ramp management function. 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 non-stop 24 hours-a-day operations, since HKIA is a 24-hour airport.

Since all stands at HKIA (passenger and cargo) are centrally allocated and managed by the Airport Authority, the main objective of SAS is to be as fair as possible to all airlines and handling agents, while making efficient and safe use of airport resources. An efficient stand allocation plan can maximize the utilization of stands and thus permit additional flights during peak traffic hours and holiday seasons. Although HKIA was designed to have more stands that 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 proximity of the immigration counters will allow passengers to quickly exit the terminal. On the other hand, 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. This will allow airlines and handling agents to perform longer term macro planning with more accuracy and a certain degree of independence.

Since 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 co-ordinate multiple sets of data within a multi-user environment is also very important.

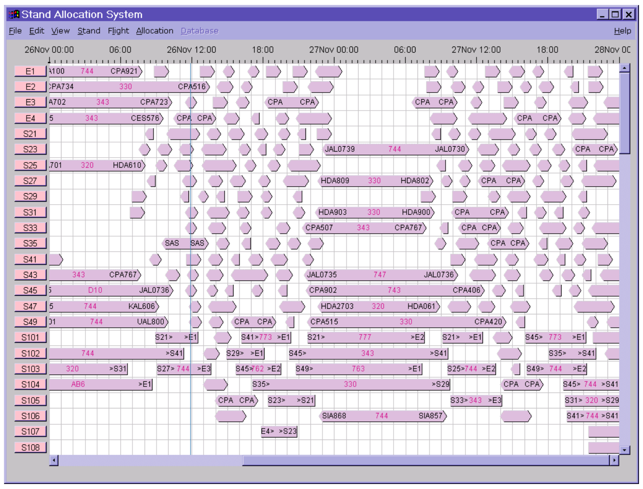


Figure 1. SAS Gantt chart with two days of assignments.

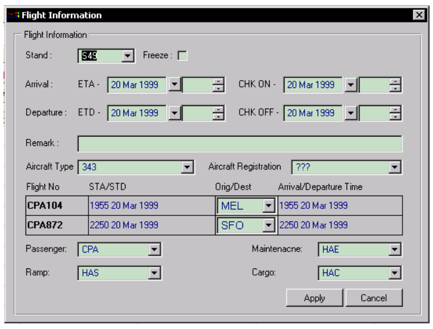
The quality of a stand allocation plan is determined by many factors – how efficient are resources being used, how convenient is it to the passengers, and most importantly, are all safety-related criteria met. Normally, a human operator must have several years of experience in order to acquire enough 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 since 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 may 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 may need to return to the airport as an air return or ground return. The ability to handle changes and events like these quickly and intelligently while minimizing impact on the rest of the committed schedule is very vital to ensure airport operates smoothly.

An AI solution allows a highly optimized allocation plan to be produced in 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 around 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 plan and generates an optimized stand allocation using an algorithm for constraint-satisfaction problems (CSP). Our scheduling algorithm considers constraints related to arrival/departure times, type of flight, aircraft type/size, airline preferences, stand configurations, etc., in producing the stand allocation plan. SAS also provides a set of menus and commands to allow the operator to enter any last minutes changes before confirming and distributing the plan. 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.


Flight Information

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.

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 statuses are displayed using different icons, symbols, and color-coded shadings in SAS. SAS automatically re-evaluate 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 multi-user 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 may prevent the next aircraft from parking, or a delay in arrival may 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 may request SAS to automatically resolve the conflict using a reactive scheduling algorithm that solves conflicts while minimizing impact on 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 – design criteria related to the AI implementation and criteria related to the IT 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.
Optimizing – The reasoning mechanism must be able to optimize on a given set of potentially contradicting constraints and criteria since 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 in order to be able to consider several hundred constraints for each assignment. There may be several hundred assignments per day.
Problem Solving – Since 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 IT point-of-view, SAS was designed to operate as a mission critical application. This added additional requirements on top of those related to AI and scheduling. The following highlights some of the key IT considerations:

Real-time Performance – Mission-critical implies that SAS response time must be close to real-time and must be able to cope with real-time changes very quickly. The implementation technology selected must be highly efficient. Interpreted programming languages might perform poorly in this respect. Multi-thread support is a must.
Multi-user Support – Most mission critical operations will require several human operators co-operating together. The system architecture must support the simultaneous use by several operators and be able to synchronous 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 that 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% 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. This also includes the ability to easily access potentially different types of relational or object databases. The degree of interoperability will depend on the technology selected.

A Solution Based on Object Technology

Based on the AI and IT 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.

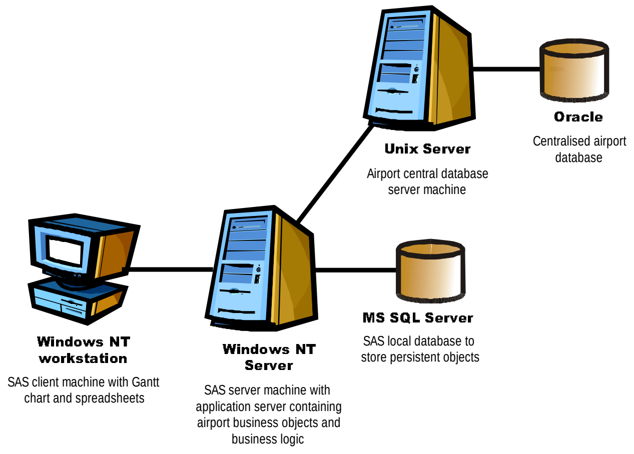

Oracle
Centralised airport database
Unix Server
Airport central database server machine
Windows NT workstation
SAS client machine with Gantt chart and spreadsheets
Windows NT Server
SAS server machine with application server containing airport business objects and business logic
MS SQL Server
SAS local database to store persistent objects

Figure 3. The SAS hardware architecture.

There is, of course, a lot of flexibility in selecting technologies if we are designing a standalone 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 multi-user n-tiered architecture, the choices must be carefully made. The following 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 MS SQL Server database. It interfaces to other external server machines to exchange data, such as seasonal schedules, rotation information, stand assignment, etc.

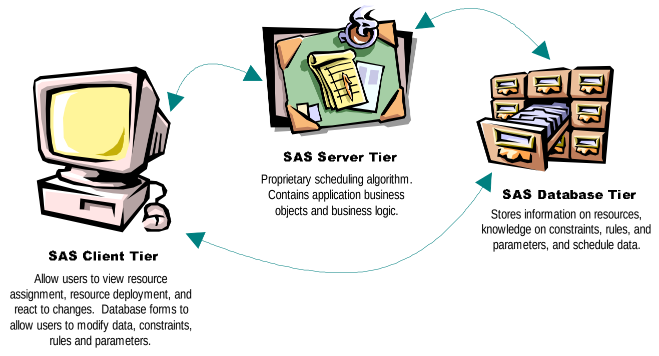

SAS Server Tier
Proprietary scheduling algorithm. Contains application business objects and business logic.
SAS Database Tier
Stores information on resources, knowledge on constraints, rules, and parameters, and schedule data.
SAS Client Tier
Allow users to view resource assignment, resource deployment,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

Figure 4. The SAS 3-tiered software architecture.

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. We selected an OT approach using C++ as the implementation language due to its efficiency and the availability of numerous off-the-shelf software components.

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

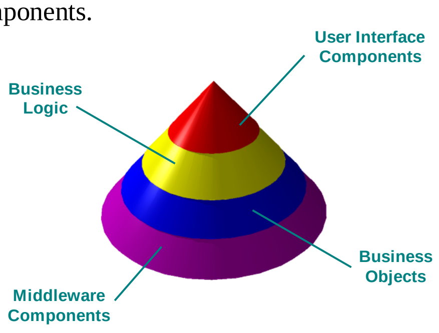


Figure 5. Four types of component software used by SAS.

Typical to most OT-based systems, SAS uses four main types of component software – middleware components, business objects, business logic, and graphic user interface (GUI) components. 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 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 OT-approach, we were able to take advantage of commercially available best-of-breed off-the-shelf component software to support the functionality required by SAS.

The Middleware Components

SAS uses two types of middleware component – a CORBA middleware and a database middleware. We selected CORBA as the core middleware technology for SAS since 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 multi-user 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 ORB [3, 4, 5, 10] for development.

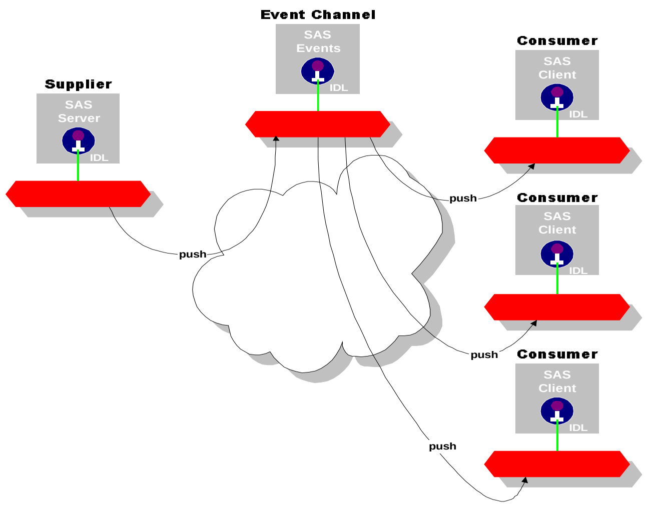


Figure 6. CORBA Event Services push updates to clients for multi-user support.

The SAS application server generates a finite set of events to notify clients of changes and updates. These events are distributed using the CORBA Event services. Incidentally, this infrastructure gives us the added advantage of potentially using same mechanism to notify other IT 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 loading balancing techniques.

SAS also uses a database middleware to insulate our system from the databases to communicate with. SAS uses off-the-shelf software components [10] to provide object-oriented access to relational databases.

The Business Objects

We were at an advantage in building the SAS application since 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, handling agents, etc. These business objects were developed on top of our proprietary **Optimiz!** scheduling framework that was implemented using off-the-shelf foundation classes [10]. The business objects are made "constraint-aware" by incorporating C++ software components that supported constraint programming [1, 6, 7, 11]. In addition, the whole system was made multithread-safe using third-party multithread components [10].

Since all key business objects in the 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 IT systems to facilitate integration as illustrated in Figure 7.

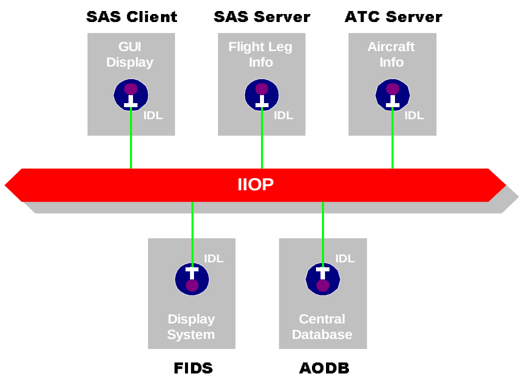


Figure 7. SAS as a CORBA-based system.

The SAS in-memory business object cache is created from a local MS 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, to be automatically committed to database through the database middleware.

The Business Logic

SAS business logic consists mainly of constraints defined using our pre-built C++ **Optimiz!** scheduling framework. Our framework provides a software infrastructure upon which a general class of scheduling systems can be built. The framework contains features to facilitate problem modeling and scheduling algorithm implementation. 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, swap, etc. Other facilities include reactive scheduling, what-if analysis, explanation generation, warning message generation, audit-trail logging, an event generator, and auto-testing facility.

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

The User Interface Components

The graphic user interface for SAS client machines was developed using highly optimized C++ graphic components [2]. The user interface consists mainly of interactive Gantt charts and spreadsheets. Additional administrative user interface facilities were developed using Microsoft Visual Basic. All SAS user interface screens follow standard Windows interaction with context-sensitive menus and on-line 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.

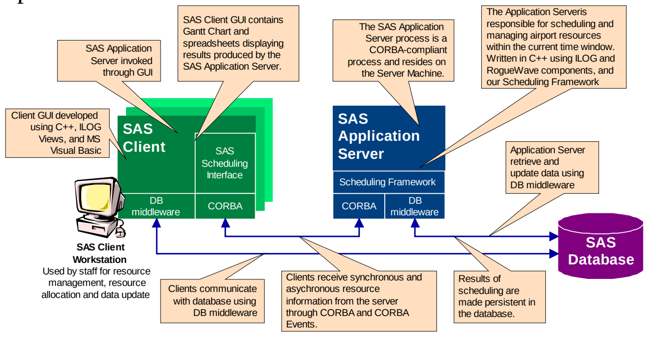

SAS Application Server invoked through GUI
SAS Client GUI contains Gantt Chart and spreadsheets displaying results produced by the SAS Application Server.
The SAS Application Server process is a CORBA-compliant process and resides on the Server Machine.
The Application Serveris responsible for scheduling and managing airport resources within the current time window. 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 retrieve and update 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 asychronous 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.

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, this 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 very efficiently as a constraint satisfaction problem (CSP) [1, 6, 7, 11]. CSP algorithms have been used successfully to solve a wide variety of transportation related scheduling problems [8, 9]. 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 may be associated with several variables since an aircraft may 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 constraints – hard constraints for domain reduction and soft constraints for value selection. These constraints are implemented using constraint components from our **Optimiz!** scheduling framework. The following highlights the key constraints for stand allocation:

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 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 may disallow any passenger flights from de-planing or boarding at cargo areas.

International/Domestic Constraint – Some stands may 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 may 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 may 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 may still use the stand for de-planing 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 may 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 are soft preferences on which stands should be assigned to which aircraft types. This 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, on the other hand, will be towed to maintenance stands, which might be further away. The tow times are defined by several parameters – the minimum time needed to off-load passengers, the minimum time an aircraft should be assigned to a stand, and the minimum time needed for boarding, and the towing time from stand to stand.

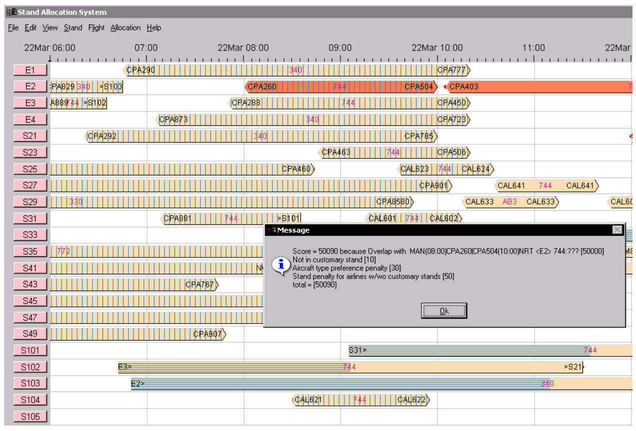
Towing Preferences – This defines preferences as to where aircraft should be towed for temporary parking. For example, certain areas may be more desirable to be used as "pad" areas. On the other hand, there might be a simple preference to just tow the aircraft to the nearest outer stand, in terms of towing time.

Customary Stand Preferences – Airlines or handling agents may have preferences as to which stands their aircraft should be assigned. 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 may have different options as to the orientation that the aircraft be parked. This constraint may be related to convenience of aircraft maneuvering or safety of de-planing 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 will be highlighted in red. Figure 9 is an example where a flight's departure (CPA504) is delayed and 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 arrival flight. The user can invoke the SAS "Why" command to get an explanation for the constraint scoring. Figure 9 also shows how different shadings and colors are used to indicate different flight statuses, such as confirmed chock on/off and new eta/etd.


Message

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.

SAS automatically solves conflicts with a click of a button. It takes less than a minute to solve all conflicts for one day. The following shows the new schedule after 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 while minimizing the impact of changes to original schedule and hence minimizing the amount of inconvenience to waiting passengers.

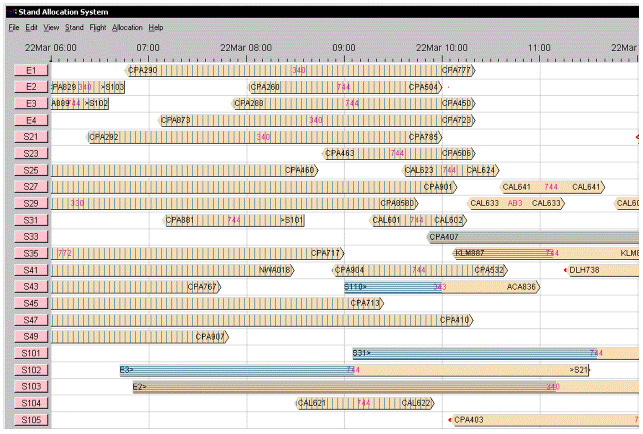


Figure 10. Result of performing reactive scheduling.

Application Use and Payoff

SAS is used by AA airfield operation staff in the Airport Control Center (ACC) at HKIA. Roughly twenty operators, who work in shifts, were trained on the use of SAS. The system has been deployed since June 1998.

Since 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, allocation plan is optimized, and high quality is maintained each time. Since 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 very quickly.

Swift Decision Making - Humans might not perform well under extreme stress and pressure such as those faced during real operations. Trying to perform problem solving under those conditions may result in less than ideal or even wrong decisions. SAS performs reactive problem solving in around five seconds and guarantees the solutions to be correct each time every time. A quick response time allows SAS operators to reply to air traffic controllers immediate while online if needed.

Guarantee Safe and Smooth Operation - Since all constraints are considered all the time, SAS guarantees that no safety-related constraints are overlooked. Over a hundred different types of aircraft lands 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 configuration or equipment failure may 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 worst, 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 travelers' satisfaction, is not easy. Many different factors contribute to this success. Stand management also 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 close to unmanned transport vehicles to allow faster exit from 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 runaway is due to operate soon and air traffic will increase. However, the 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 begins operation. 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 (OO) software components is that standard OO software engineering practices can also be followed. For SAS development, we followed an OO methodology that is based on Rational's Unified Process. User requirements were documented using Use Case Analysis and the OO 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 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 part under extensive testing and trial use by AA operators with actual data. Since 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 very peculiar sequences of events.

The coding of SAS took roughly four months elapsed time total with a development team of ten 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 AA's own staff members. AA senior operators maintain the knowledge base through a set of MS Visual Basic database forms. All site-specific knowledge, constraints, parameters, and data are stored in the local MS SQL Server database. Any change in domain knowledge can be performed without any SAS source code change. Knowledge related to stand operations do 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 AA's IT staff members. In addition, we provide 24-hour technical and user support both through telephone and on-site if needed.

Conclusion

This paper 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 paper provides some insights on the design and development of mission critical component-based AI systems.

Acknowledgements

The authors would like to thank AA operations and IT staff for the tremendous co-operation, support and enthusiasm received throughout this project.

References

[1] J. Cohen, *Constraint Logic Programming,* Communications of the ACM, 33(7), pp.52-68, 1990.

[2] http://www.ilog.com

[3] http://www.iona.com

[4] http://www.inprise.com

[5] http://www.ooc.com

[6] G.L. Steele Jr., *The Definition and Implementation of a Computer Programming Language Based on Constraints,* Ph.D. Thesis, MIT, 1980.

[7] V. Kumar, "Algorithms for Constraint Satisfaction Problems: A Survey," In *AI Magazine,* 13(1), pp.32-44, 1992.

[8] J.-F. Puget, "A C++ Implementation of CLP," In *ILOG Solver Collected Papers*, ILOG SA, France, 1994.

[9] J.-F. Puget, "Object-Oriented Constraint Programming for Transportation Problems," In *ILOG Solver Collected Papers*, ILOG SA, France, 1994.

[10] http://www.roguewave.com

[11] P. Van Hentenryck, *Constraint Satisfaction in Logic Programming,* MIT Press, 1989.