Abstract

Nowadays, electronic commerce on the Internet provides many new business opportunities. The number of software applications for business on the Internet is growing very quickly. To explore the capabilities of using the Common Object Request Broker Architecture (CORBA) to develop distributed applications, a cinema tickets reservation system was developed in this project. This system was implemented in Java to make use of her portability, built-in threading, garbage collection and exception handling etc..

Acknowledgement

I would like to acknowledge my supervisor Professor Michael Lyu, who provided many valuable opinions and guidance for me throughout the project.

Additionally, my partner Mr. Andy Kam deserves many thanks for discussing with me and helping me in this project.

Finally, I would like to thanks all people who have helped me in this project.

Table of Contents

Chapter 1

Introduction

In these decades, our styles of living are changing rapidly, for example, in the past we only have telephones at home but now most of us have a mobile phone with us so that we can make a phone call to other person at any time, any where. A few years ago only a few people know about Internet and at that time it is only for academic purpose. Now Internet become a part of our daily life, we go there for chatting, shopping, searching information, sending email, buying stock, entertaining and sharing information.

We live in a world with advanced technology. People try to make the advanced technology more applicable to our daily lives. E-commerce is a direction people are working for because we want to do business through the Internet. If we can solve the existing problems of developing E-commerce
, there are many business opportunities for future.

To do business through the Internet, we need to solve some problems such as security, heterogeneous between objects, authentication and law etc. Internet is a distributed system where objects communicate with other to invoke their methods. To develop E-commerce through Internet, applications need to use distributed transaction to accomplish a service. A distributed transaction involves several objects to provide a service in which all objects must commit or abort atomically. Moreover, objects in the Internet are developed by different programming languages and reside at different platforms.

There are many solutions to these problems such as Socket programming, Microsoft's DCOM, JAVA RMI, RPC, CORBA etc. CORBA is a new technology in this area and it provides good features to develop applications on Internet. It allows objects developed in different language to communicate with others and it is platform independent. It provides many services such as locating an object in the system. It monitors the transactions to ensure that the transactions are either commit or abort as a whole. To allow objects to use others' method, only the interfaces are required to be specified. Client programs can use the interfaces to invoke other objects' methods regardless their implementation. It has the advantage that if the implementation was changed, no change in the client programs are required.

To explore the capabilities of CORBA in the project, an online cinema tickets booking center was developed by using CORBA and JAVA. The outline of the report is as follows: Chapter 1 is the introduction. Chapter 2 is the nature of distributed system. Chapter 3 is introduction to CORBA. Chapter 4 is the system requirement. Chapter 5 is the system design. Chapter 6 is the implementation. Chapter 7 is the testing. Chapter 8 is the discussion. The chapter 9 is the conclusion.

Chapter 2

Nature of Distributed System

2.1 Distributed Systems

The application developed in this project is on the world's largest distributed system – the Internet. Hence, before the introduction of CORBA and the application, I would like to introduce the concepts and key properties of distributed system.

The outline of this chapter is as follows:

- What is a distributed system?
- Examples of distributed systems
- Common characteristics

Firstly, a definition of distributed system is given. Then there is a comparison of it with the centralized systems. For a better appreciation of the issues that are involved in distributed systems, we will review several distributed systems that we have already come across in our lives. Finally, we shall then elaborate on the common characteristics of distributed systems.

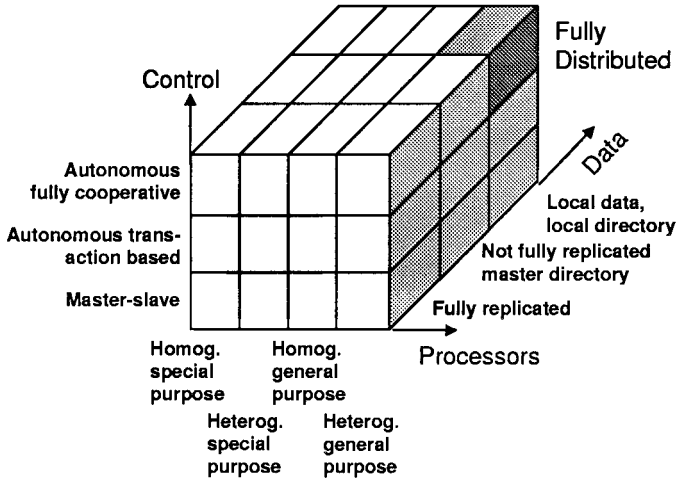


Figure 2.1 Distributed System Types (Enslow 1978)

The model of systems is involved hardware (processors), application and system software (control) and application and system information (data). This is a three dimensional model but which of these dimensions have to be distributed for the system to be a distributed system?

For a system to be distributed, Enslow requires that distribution is transparent and system users are unaware of the fact that the system is composed of multiple processors.

According to the figure 1.1, Enslow's model (1978) is fairly rigid: A system is a fully distributed system if and only if all dimensions are fully decentralized.

1. Full hardware decentralization includes multiple heterogeneous control units (as opposed to a single control unit with multiple processors and multiple homogeneous control units).

2. Control must be provided by multiple units cooperating with each other rather than in a master-slave relationship

3. Data must be partitioned and/or replicated, each part with its own local directory.

However, Enslow's definition is too restrictive in our opinion. Techniques of distributed system construction should also be employed if only a single dimension is decentralized.

Nature of Centralized System

To introduce the consequences of distributing a system, we compare its characteristics to those of centralized systems.

In a centralized system, there is a single component that may be decomposed further. However, its parts (such as classes in an object-oriented program) are not autonomous, i.e. the component possess full control over them at all times. As there are no other components, there is no need to provide an interface to the component.

If the component supports multiple users (e.g. a relational database), the users need to share the complete component at all times. A centralized system runs in a single process. There is no need to consider concurrency control and synchronization.

There is only a single point of control. The program counter of the processor, register variable contents and the virtual memory occupied by the process determine the state of the component.

As a result, the system is either running or it is not. There is no such case where part of the system or parts of its interconnection have failed and need to recover.

Nature of Distributed System

The components in a distributed system may be decomposed further. These components are autonomous, i.e. they possess full control over their parts at all times. The components, however, have to provide interfaces to be able to use each other.

In a distributed system, there may be components that are used by only some users but are not used by others. It is an advantage to have these components residing on machines that are local to the users that use them.

A distributed system runs in multiple processes. These processes are usually not executed on the same processor. Hence it is necessary for inter-process communication with other machines through a network. Different levels of abstraction (confer the ISO/OSI reference model) are involved in this communication.

Unlike centralized system, distributed systems have multiple points of control, but these are not totally independent. Components have to take into account that they are being used by other components and have to react properly to requests.

There are multiple points of failure in a distributed system.

- It may fail because a component of the system has failed.
- It may fail if the network has broken down.

- It may also fail if the load on a component is so high that it does not respond within a reasonable time frame.

However, the distributed system is more fault-tolerant than the centralized one.

2.2 Examples of distributed systems

We now review some examples of distributed systems that can provide a better understanding of what are to be tackled in the construction of distributed systems.

Local Area Network

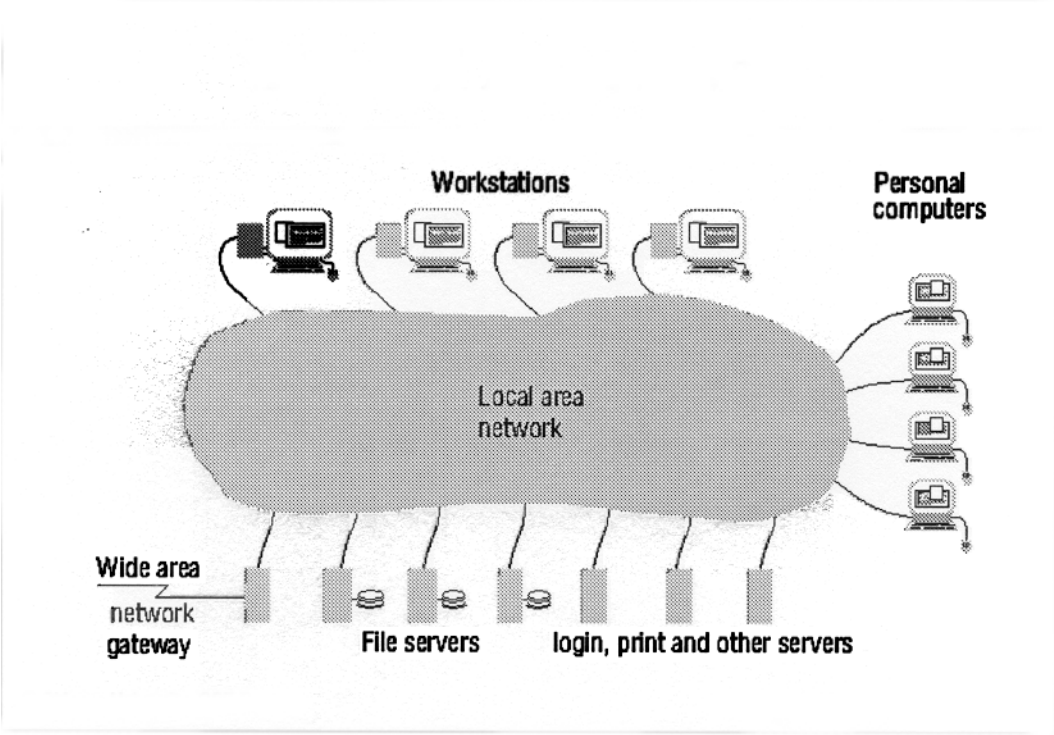


Figure 2.2 Local area network

A local area network consists of a number of different computers. Workstations and personal computers provide the front-end for network users. Different servers provide

shared services.

One or several network file servers provide data storage services. Any workstation and PC may henceforth store files on disks maintained by these file servers.

A local name server maps machine names to IP addresses, user names to user ids and group names to group ids. Any machine can request a service to resolve a certain name.

One or several print servers control the access to shared printers. Workstations and PCs have the server printing jobs for them.

Another component provides a gateway to the wide area network. As a user you need not be aware which machine provides which service.

Database Management System

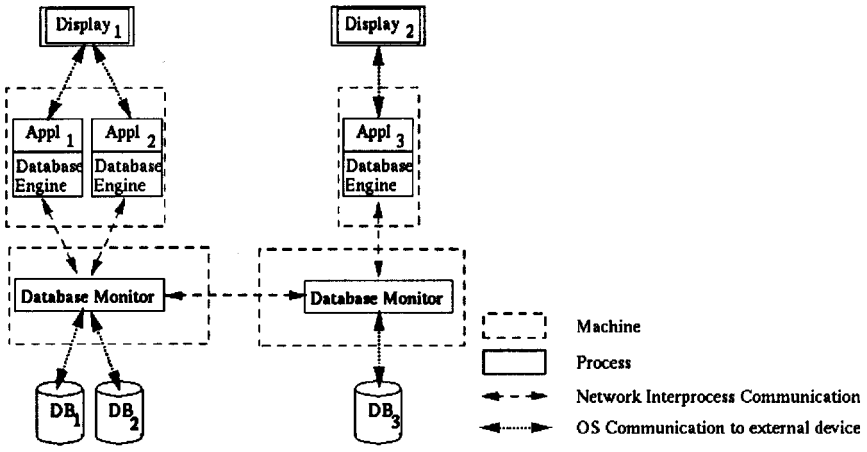


Figure 2.3 Database Management System

Many client applications want to access and update shared data in a database. The client

applications might be banking systems, real-estate agencies, airline-ticket reservation systems accessing data like balances of bank accounts, details of property that are for sale or to let, or airfares and aircraft reservation data.

The database is physically distributed over several processors to take advantage of local data accesses for increased performance of client applications.

Data may be replicated to reduce the impact of failures of a processor and/or the network. This can also reduce the bottleneck of some heavy load databases.

Each processor runs a database monitor that implements the mapping between the database seen by clients and the physical database stored on the different processors.

Database monitors have to cooperate with each other to implement client accesses to remote data, updates of replicated data and concurrency control. The physical distribution of data is therefore transparent to clients.

Automatic Teller Machine Network

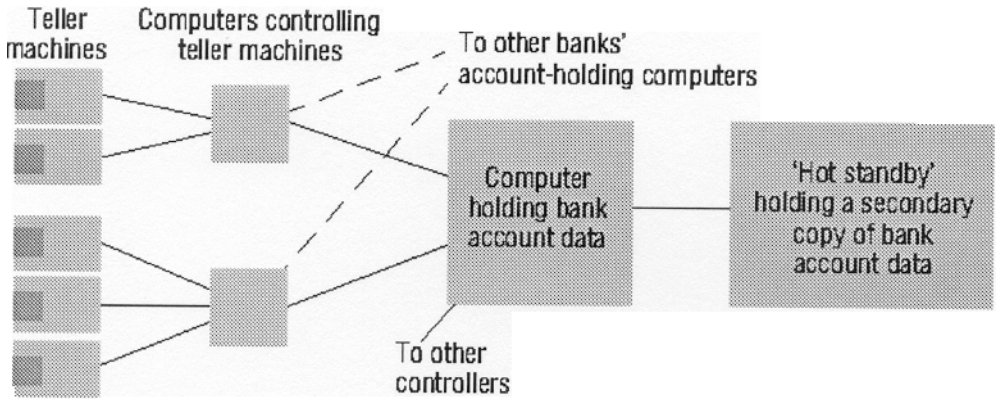


Figure 2.4 ATM network

To facilitate bank customers to withdraw cash from their bank account, an automatic teller machine network is maintained by Banks and building societies. Besides the basic requirement of cash withdrawal, customers also have high security, privacy and reliability requirements. Moreover, customers may want to withdraw cash from their account through a ‘foreign’ teller machine.

Technically, a front-end computer controls one or several tellers. It transfers withdrawal requests to the computer of the account holder's bank, it awaits the bank granting the request, and it has to be interoperable with heterogeneous computer systems, for example, Hang Seng Bank may have different account management systems than HongKong Bank and Bank of China.

Each bank has fault-tolerant systems to quickly recover from failures of their account holding computers. An example is the ‘Hot standby’ computer which maintains a copy of the account database and can replace the main computer within seconds.

World Wide Web

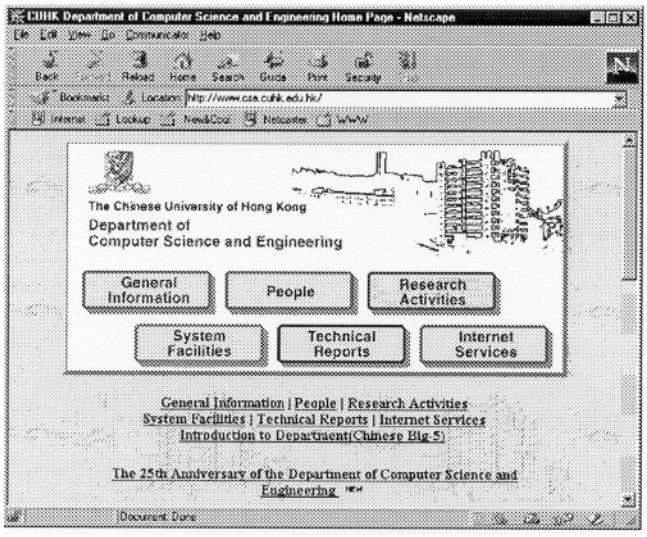


Figure 2.5 World Wide Web

A Web browser is a user interface to the world s biggest distributed system, the Internet. A Web page includes links to other Web pages. These links are specified as URLs.

A URL is the name of a protocol (ftp, http, etc.), the name of a site (gateway1.cse.cuhk.edu.hk) and the name of a file.

To follow a link to a remote Web page, your Web browser talks to the local name server to resolve the symbolic site name into an IP address (137.189.88.153). Then it talks to the http daemon running on that web site and requests the delivery of the Web page addressed by the URL.

To obtain a file from a remote ftp site, your Web browser resolves the site name with the local name server, it talks to the ftp daemon running on that site and performs an anonymous login. Then it switches the daemon into an appropriate transfer mode and obtains the file addressed by the file addressed in the URL.

To send an e-mail, your Web browser opens a new dialog window where you can enter the addressee(s) and the e-mail text. Then it talks to the local sendmail daemon to have it delivering the email to the sendmail daemons on the sites of your addressees.

2.3 Common characteristics of distributed systems

At a first glance constructing a centralized system appears to be much easier and it is

really the case. So why do we bother about constructing distributed systems?

Apparently, some properties of a distributed system cannot be achieved by a centralized system. Hence, it is worthwhile to keep those properties in mind during the design or assessment of a distributed system.

The properties are as follows:

Resource sharing: I can put all my publications on my Web site, hence sharing them with all users of the Internet.

Openness: I have credit cards from Hang Seng Bank and Wells Fargo Bank in U.S.A. and can use them at others' tellers. These banks, however, would never develop a common centralized teller system. It is because their systems are open and interoperable that I have this flexibility.

Concurrency: Multiple database users can concurrently access and update data in a distributed database system. The database system preserves integrity against concurrent updates and users perceive the database as their own copy. They are, however, able to see others' changes after they have been completed.

Scalability: Distributed systems, such as the Internet, grow each day to accommodate more users and to withstand higher load.

Fault tolerance: Two (distributed) account databases are managed by the bank to quickly recover from a break-down.

Transparency: When using a distributed system it appears to users as if it was centralized.

We will discuss the above properties one by one in details.

Resources Sharing

Hardware, software and data are the resources to be shared. It has to be defined who is allowed to access shared data in a distributed system. For the sensitive information, an access control policy has to be defined.

To implement this access control policy a resource manager is needed. As an example, for the Web, the local http daemon takes the role of this resource manager. To control access, it interprets a .htaccess file in the directory where a particular page is stored and only grants access to those sites that are listed in that file.

A more complex resource manager is the database monitor we came across in the DBMS example. Apart from access control, it provides the naming scheme for data (the mapping of data to physical storage addresses) and controls concurrent accesses.

There are different models resource managers and resource users can be deployed in a distributed systems architecture. In a client/server model, there are servers that provide certain resources and clients who use them. Servers may themselves be clients and use resources provided by other servers.

In this project, we will extensively use a more sophisticated model, the object-based model. In this model, any resource is considered as an object that encapsulates the resource by means of operations that users of the resource can invoke. This model is used by the Object Management Group (OMG) in the Common Object Request Broker Architecture (CORBA).

Openness

Openness tries to address the following question: How difficult is it to extend and improve a system?

As we all know in most cases, both functional extensions and improvements require new components to be added and these components may have to use the services provided by existing components.

Hence, the static and dynamic properties of services provided by components have to be published in detailed interfaces. The new components have to be integrated into existing components, so that the added functionality becomes accessible from the distributed system as a whole.

In distributed systems, components may not always be running on the same platforms. For instances, Hang Seng Bank, HongKong Bank, and Bank of China almost certainly do not have the same type of hosts, it's quite likely they use different programming languages and have different networks. Despite of that, their automatic teller machines have to be integrated.

To achieve such a heterogeneous integration, often different data representation formats have to be integrated. For example, if components running on a Windows-3.x PC have to be integrated with components running on a Sun SparcStation, short integers on the Sun have 64 bit, while they only have 16 bit on the PC.

Concurrency

Components in distributed systems are executed concurrently. There may be many different people at different teller machines. Likewise, there are many different users working in a local area network.

While these components access shared resources, the resources have to be protected against integrity violations that may be introduced through concurrency.

As an example for a lost update, consider that you withdraw 50 dollars. This requires the bank's account database to compute:

```
Debitbalance = balance-50; /* Opl */
Balance = debitbalance; /* Op2 */
```

If a clerk in the bank credits a check of 100 dollars the following computation has to be done:

```
creditbalance = balance+l00; /* Op3 */
balance = creditbalance; /* Op4 */
```

If these two modifications to your account are done concurrently the integrity of the account data may be violated in two ways:

1. your debit may not be recorded (bad luck for the bank) if the schedule is (Op1, Op3, Op2, Op4).

2. the credit of your check may not be recorded (bad luck for you) if the schedule is (Op3, Op1, Op4, Op2).

These situations have by all means to be avoided. Concurrency control facilities (such as locking) are needed in almost any concurrent system.

Scalability

Centralized systems often create bottlenecks as soon as a certain number of users are reached. Distributed systems can be built in a way that these bottlenecks are avoided. Then new processors can be added to accommodate new users.

For instances, the Internet grows every day by adding new sites. Other Internet sites are not affected by these additions. They do not have to be changed.

However, components in distributed systems have to be designed in a way that the overall system remains scalable. Sometimes it is required to relocate components, i.e. to migrate them to new processors. Relocation is required to populate new processors with components and to remove a certain amount of load from existing processors.

Then it is important that no or only very little assumptions are made on the location of components, both within the component itself and also within other components that use the component. Otherwise these components having explicit location information have to be changed whenever a component is relocated.

Fault Tolerance

Hardware, software and networks are not free of failures. They fail either because of software errors, failures in the supporting infrastructure (power-supply or air-conditioning), mis-use of their users or just because of aging hardware. The average life time of hard disks are between two and five years, much less than the average life-time of a distributed system.

Given that there are many processors in a distributed system, it is much more likely that one of them fails than it is that a centralized system fails. Distributed system, therefore, have to be built in a way that they continue to operate, even in the presence of failures of some of its components.

A distributed system can even achieve a higher reliability than a centralized system if distribution and replication is exploited properly. Two different means have to be deployed to achieve fault tolerance: *recovery* and *redundancy*.

- Components that are able to recover from failures have been built in a way that they react in a controlled way if they rely on services of components that have failed.

- Redundant hardware, software and data decreases the time that is needed after a failure to bring a system up again.

Transparency

The complexity of distributed systems should be hidden from their users. They should not have to be aware whether the system they are using is centralized or distributed. Thus, it is transparent for the user that s/he is using a distributed system.

For the administrators of the system, however, this is not true. For them, it may well be important (e.g. during load balancing) to know which component resides on which machine.

To make life easier for an application programmer, s/he should also not have to be aware that s/he is using distributed components.

There are many aspects of transparency such as access transparency, location transparency, concurrency transparency, replication transparency, failure transparency, migration transparency, performance transparency and scalability transparency.

Chapter 3

Introduction to CORBA

3.1 Distributed software engineering using CORBA

From the discussion in the last chapter, we have learned that any distributed system is composed of distributed components. Now the question is: *How can we effectively engineer the components that comprise a distributed system?*

A major part of the problem to engineer components for a distributed system is to find a model for these components. We shall take an object-oriented approach and see components as objects that provide services and encapsulate a component state.

The approach we have chosen to present in this topic has been developed by the Object Management Group (OMG), a consortium of leading hardware and software vendors, user organization and research institutions.

It has been developed as part of the OMG's work on the Common Object Request Broker Architecture (CORBA). We will discuss the CORBA object model, which is an object-oriented component model for distributed and heterogeneous components.

The model is implemented in terms of IDL, the OMG Interface Definition Language and we will discuss the design rationales of IDL in greater detail.

We have seen that distributed systems had multiple interacting components. Typically

these distributed components are not homogeneous. They are running on different hardware platforms with different operating systems, they have been implemented in different programming languages and use different network protocols.

For a distributed system to appear as a single system to a user, as it is required by the transparency property, these heterogeneous components have to inter-operate to contribute to achieving the overall systems objectives.

In order to construct these components in an interoperable way, there has to be a common component model.

1. The model has to define component states, i.e. the relationships the component has with other components and the data the component can store.

2. It has to be expressive enough to describe the services a component offers to other components and the services a component uses from other ones.

3. Finally, the component model has to provide primitives that can express how a component interacts with other components.

3.2 CORBA Object Model

The component model of CORBA is based on the object-oriented paradigm. Hence components are seen as objects. Object orientation seems appropriate due to its support of data abstraction and reuse.

Attributes of objects are used to model the externally accessible state of components; this component state can thus be considered as the set of its current attribute values.

A component offers a set of services to other components. These define the component's behavior. They are modeled in terms of operations exported by the object.

The interaction of a component with a remote component is modeled in CORBA in terms of operation execution requests and responses to these requests.

Service executions may fail. These failures may be due to some problem common to any distributed applications. In these cases failures may be described in a generic fashion. Failures may also be due to some application specific problem. These application specific failures have to be dealt with by the application. The CORBA object model provides exceptions for this purpose.

3.2.1 Types of distributed objects

Many different objects share the same static properties (attributes, operations and exceptions). It is, therefore, unreasonable to have a designer of a distributed system describing properties for each of these objects. They should rather be defined only once for all ‘similar’ objects.

The OORBA object model, therefore, introduces the concept of object types. They are a vehicle to define properties shared by all objects that are of that type.

Types are also used for object creation: objects are *instantiated* from a type. The type they are instantiated from is referred to as their type. Objects keep their type during their whole lifetime.

The CORBA object model incorporates a static type system. This type system is used to verify at compile time that an object has a certain property at run-time.

3.2.2 Attributes

An object type may declare an attribute by characterizing the attribute name and its type. The attribute name will be used by remote components to access or even update the attribute's value. The domain of the attribute value is defined by the attribute type.

The attribute type can denote an object or a non-object type.
If it denotes an object type, the attribute value *refers* at run-time to an instance of that object type. If it denotes a non-object type, the attribute *contains* at run-time a value of that type.

An attribute determines whether or not other components may modify the attribute value at run-time.

Attributes will be implemented in terms of operations. For modifiable attributes two operations will have to be provided. The first operation will be used to set the attribute value. The second operation will be used to read the value. For attributes that can only

be read, the set-operation is not available.

3.2.3 Exception

The CORBA object model takes a rather simple approach. Exceptions are a mechanism to inform a requester object about a failure.

Operation execution may fail due to a system error or due to an error specific for the object. Failures that are specific to an object handled differently by additional operation executions.

Exceptions may be raised explicitly by the service providing object or implicitly by the distributed operating system. If no exception has been raised, the client knows that the server has performed the request properly. An exception may have additional data that informs the client object about the nature of a failure.

3.2.4 Operations

Operations that are exported by an object are defined in terms of a signature. Part of that signature is an operation name. The name is used to identify the operation when the operation execution is being requested by a requester.

Operations may be parameterized. Parameters are used to pass specific arguments to the operation and to return results from the operation to the requester. The signature defines whether the operation expects the requester to provide a value or whether the

requester can expect the provider to return a parameter value or both.

Parameters are given a name and a type. The name can be considered as an informal explanation of the parameter's semantics. The type constraints the domain of values that can be passed through the parameter.

Operations return a value whose type is defined as part of the signature definition. Again the type restricts the domain of values that can be returned from the operation to the service requester. Operations also declare the set of specific exceptions that may be raised during the operation execution.

3.2.5 Operation execution requests

Objects can request other objects to execute a particular operation. The object that issues the request is called client object and the one that provides the operation is called server object.

Server objects may be clients themselves, because they may rely on the operations provided by other server objects. The operation execution request is expressed by sending a message to an object. The message is identical to the name of the operation that is to be executed.

The signature of the operation to be executed defines a contract between the client and the server. The parameters define the types of values that are to be provided and the types of values returned. The exceptions define which failures the requester has to

expect.

It is important that the client reacts to these failures properly: They have to catch the exception the server may have raised. Otherwise the client may not be aware that the server has not executed the operation completely.

3.2.6 Subtyping

Different types of objects in the CORBA object model are arranged in a type hierarchy. Common attributes, operations and exceptions of different types can be defined in a common super-type. Subtypes then need not define these properties again but inherit them from their super-types.

An object is *a direct instance* of a type, if and only if it has been instantiated from that type. An object is an *instance* of all the supertypes of its type. A subtypes may add specific properties to those inherited from *a* super-type.

The subtype relationship is transitive. This means that if a type B is a subtype of A and another type C is a subtype of B then C is also a subtype of A.

A subtype may *redefine* the definition of a particular property. It may give an operation, for instance, a slightly different behavior.

Sometimes operations are *abstract* (or deferred) and are not implemented by a type T, but have to be implemented in all subtypes of T. Then other types can be sure that the

operations exists in all instances of T. These abstract operations are often used to implement call-backs.

3.3 The OMG Interface Definition Language

The OMG has standardized an interface definition language (IDL) to express the abstract concepts of the CORBA object model.

IDL is not bound towards a particular programming language. The IDL can be used to declare the exported properties of object types.

IDL is not computational complete. It does not have concepts for defining variables and algorithms as these should not be exposed at the Interface level.

The OMG has standardized IDL programming language bindings for C, C++ , Smalltalk, Ada-95,Cobol and Java.

Objects whose interfaces have been declared in IDL can be implemented using these programming language bindings. A language binding is also used to request the execution of operations that are specified in IDL from a particular programming language. The advantage of this approach is that programming language interoperability is achieved.

3.4 Object Management Architecture

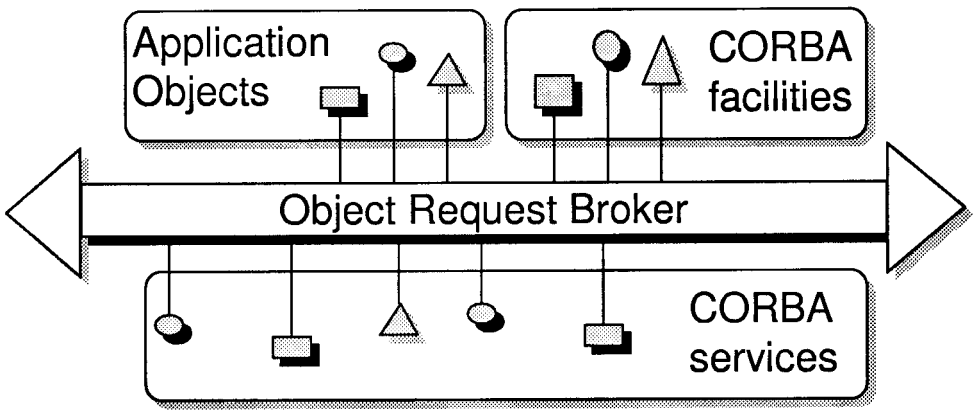


Figure 3.1

The core of the object management architecture is an object request broker. It enables an object to request an operation execution from another distributed object. The objects can be very heterogeneous in the sense that they can be running on different hardware platforms with different operating systems.

Objects that use an ORB are classified into application objects, CORBAservices objects and CORBAfacilities objects.

A number of problems occur in almost any distributed system. Examples of these problems are naming, trading, migration of components, concurrency control, transactions and the like. Object based solutions to these problems have been standardized by the OMG within the framework of the CORBAservices. It is expected that every vendor of an object request broker provides implementations for these services. This will accomplish portability and interoperability of application objects.

The OMG has started to define a number of component interfaces that may be useful but will not necessarily needed in every distributed system. These component interfaces are defined as CORBAfacilities. It is not mandatory for ORB vendors to

provide these components.

Objects that are specific for a particular application are considered as application objects. They can use or customize CORBAservices or CORBAfacilities objects and so leverage and reuse the solutions the OMG has defined and ORB vendors have implemented.

3.4.1 Accessing Remote Objects

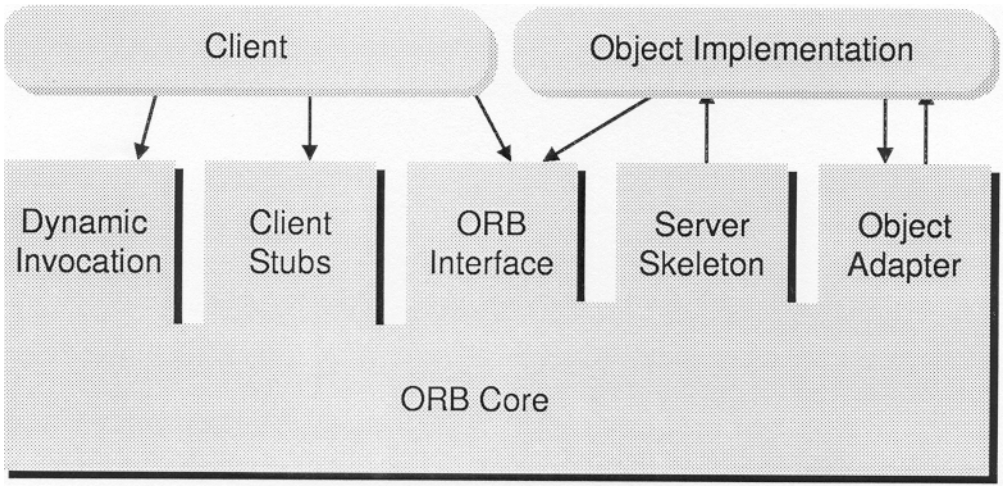


Figure 3.2

The above figure displays the various modules that are involved in a server object from a client.

Client stubs and *server skeletons* perform marshalling of operation parameters, transform them into a common format and manage the synchronization between client and server.

Client stubs are used for statically defining operation invocation requests at compile time. A *dynamic invocation interface* can be used to define operation invocation requests dynamically at run-time.

Both client and server objects get access to ORB functionality through the ORB interface. They need it, for instance, to initialize themselves properly and to get access to CORBA services.

The *object adapter* implements different object activation strategies. These strategies control the allocation of multiple or single objects to processes and determine whether objects are always available or only started on demand.

The *ORB core is* in charge of implementing session, transport and network layers and to hide these concerns from the formerly discussed modules.

3.4.2 Static vs. Dynamic Invocation

The static way of invoking a remote operation can only be used if the IDL interface is available at the point in time when the client is being developed. This is because the client stubs that are derived from the IDL interface are to be linked with the client program.

This is, however, not always applicable. There are application domains where generic clients are built before specific server objects become available. An example would be an object browser that can display an object's attributes.

The dynamic invocation interface provides facilities for objects to define requests at run-time. A request can be created for any object. Then a name of an operation to be executed can be determined and the operation parameters can be set. After that the operation request can be issued and the operation results can be obtained from the request.

A disadvantage of this dynamic invocation interface is that it is not type safe. The interface itself does not provide any guidance as to the operations and attributes that are available for an object or the numbers and types of their parameters.

Clients can limit the unsafeness of dynamic invocations by using an interface repository. The interface repository keeps information about the interfaces of an object and can be used to query which operations and attributes an object supports and can be used to find out about their types and parameters. Entries into the interface repository are made by the IDL compiler during stub generation.

3.4.3 ORB Interface

Among other things, the ORB interface defines the object type Object, the root of the object type hierarchy. Object defines a number of operations for object identification purposes that are explained later.

The ORB interface provides an operation to startup the object request broker.

Also the ORB interface standardizes the way clients register themselves with the ORB and servers to obtain an identifier of the object adapter to register the server with the object adapter.

Finally, it provides a programming interface to the interface repository that can be used to traverse and query the set of object types.

3.4.4 Object Identification

Objects are uniquely identified by object identifiers. They are the principle way to access object attributes and to execute an object's operations. Hence object identifiers achieve location transparency.

Object identifiers are persistent, i.e. once an object identifier has been assigned to an object the object will retain that identifier throughout its lifetime, irregardless whether ORB or object server are stopped.

Object identifiers can be externalized (converted into a character string) by means of operations exported from type Object. This is needed, for instance, to transmit an object identifier to a remote component through communication outside the ORB.

To obtain the object identifiers of remote components, the following ways can be employed.

1. The principle way is to use a *naming* or *trading* service. Naming is a technique that

maps external names to object identifiers (like the white pages of a telephone directory) and trading is a technique that enables clients to find objects by the services their operations implement (similar to the yellow pages).

2. Object identifiers can be obtained by reading an attribute whose type is a subtype of Object.

3. Object identifiers can be obtained as a result of an operation execution, or through an out or inout parameter of an operation if these types are subtypes of Object.

4. Finally, object identifiers can be obtained by internalizing an externalized object reference.

3.4.5 Activation Strategies

CORBA standardizes the operations of the *basic object adapter* (BOA). An BOA implementation has to be provided by any CORBA compliant object request broker.

The BOA defines four different object activation strategies, which define how objects are allocated to processes.

1. *Shared server* strategy (A) one process or thread accommodates multiple objects. Operation execution requests are queued while the process performs an operation of one of the objects.

2. *Unshared server strategy (B)* each object is executed in its own process or thread. Operation execution requests only have to be queued if another operation of the same object is currently being executed.

3. *Server per method* strategy (C) each method of each object is executed in its own process or thread. Requests only have to be queued if the method of that object is currently being executed.

4. *Persistent server* strategy (D) - activation is done outside the ORB, for instance manually by an administrator or automatically through the operating system. The process would then run until it is stopped explicitly.

With the first three strategies, the basic object adapter starts the processes/threads as soon as the ORB has transferred a request. Implementations therefore have to be registered with a so called *implementation repository.* Processes and threads are deactivated by the BOA as soon as no requests are to be handled anymore.

3.4.6 Request vs. Notification

In IDL operations the client awaits the completion of the operation on the server and the quality of service is *at most once.*

For notifications where the semantics of the client is fully independent of the result of the operations, it is not be necessary to wait until the server has handled the notification. This is the case if operations do not

- have return values (i.e. their return value is void),
- have out or inout parameters and
- raise specific exceptions.

In these cases the semantics of the further execution of the client does not depend on the server and the client may continue immediately after the ORB has accepted the request from the client. Then, however, only a *maybe* quality of service can be achieved.

The ORB cannot decide itself, whether to execute an operation as a request or a notification, because it does not know the quality of service an object implementation wants to achieve. Therefore IDL has a language construct oneway that can be used to declare that the operation is executed as a notification.

3.4.7 Failures

The invocation of a remote operation through an object request broker may fail for reasons: servers may collapse, request or reply messages may be lost.

In case of failures in CORBA, a client is informed about a failure in terms of an exception. The exception gives a very precise account as to why an operation execution request has not been handled properly.

Clients may exploit this knowledge to decide whether they want to retry the operation execution request or to report the request to an administrator or to the user.

CORBA distinguishes between generic and application specific failures. Recall that generic failures are defined in the CORBA standard and application specific failures are defined through exceptions in IDL

The different programming language bindings for IDL will have to deal with exceptions and enable programmers of both servers and clients to deal with exceptions.

Server programmers will have to determine in object implementations when to throw exceptions. Client programmers must be able to catch both, application specific and generic exceptions.

Chapter 4

System Requirement

4.1 System Description

Figure 4.1 is a flow chart describing the booking center. From the chart, you can see that if the user want to use the system, he/she must registered with it. If the user is a new customer, then he/she can perform online registration. There is no online registration for cinema manager. If the cinema want to add, remove and change manager, she must contact us personally.

If the user login as cinema manager, there are two things he/she can do. Firstly, the manager can select a film registered in the booking center and add the schedule of showing to the cinema for booking. Secondly, if the manager want to create an unregistered film for booking, he/she can register the film with the booking center before creating the showing schedule for booking.

If the user login as a customer, there are also three things he/she can do. Firstly, he/she can select films to add comments to them or view the comments of those films. Secondly, he/she can select films for booking, the payment of booking can be made by credit card. Thirdly, the customers can check their booking information.

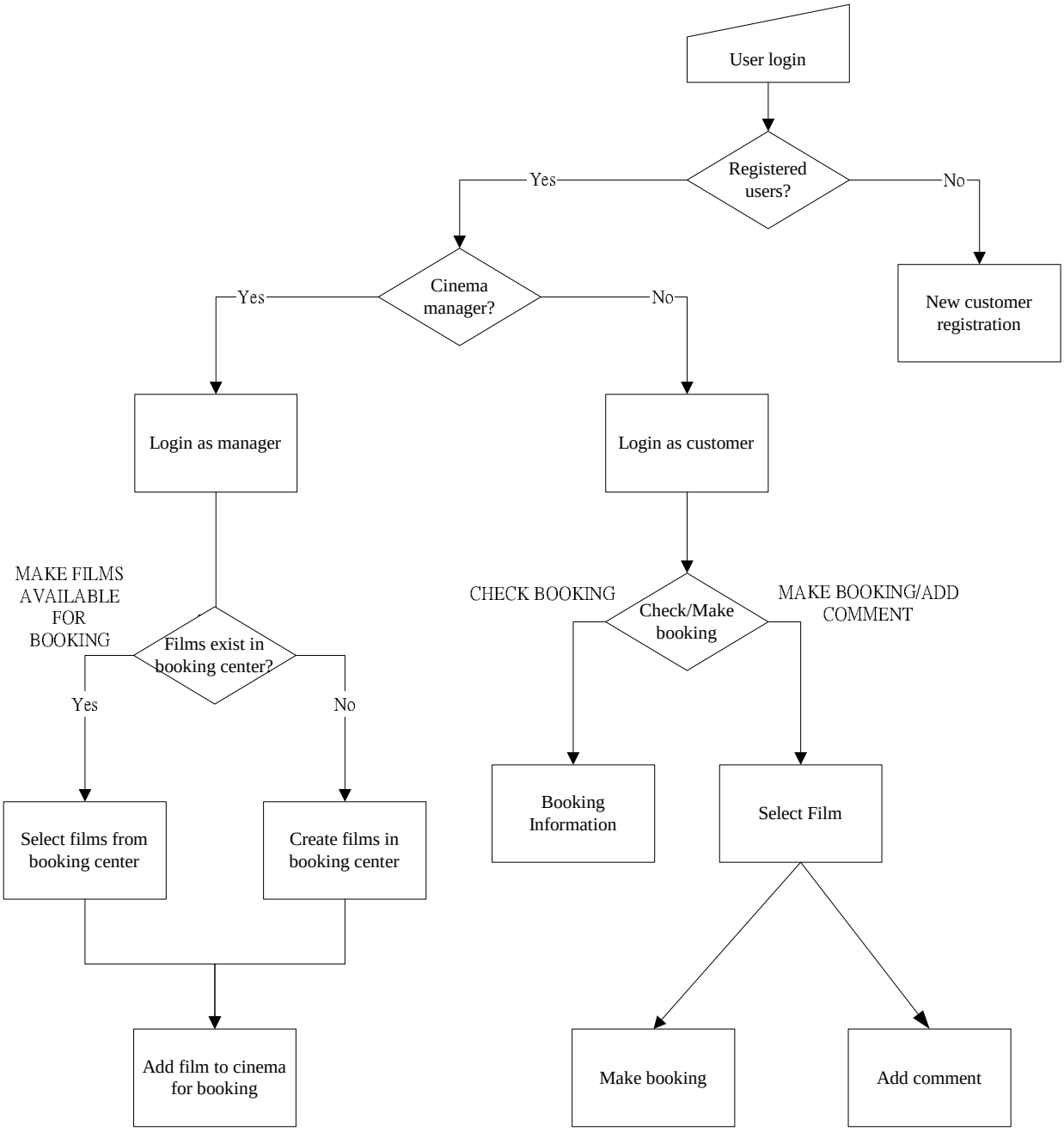


Figure 4.1 Booking Center

4.2 System Requirement and Capabilities

From the system description, we can summarize the following system requirements.

- Support multiple cinemas
- Support multiple films within booking system
- Support multiple customers booking films
- Support customers booking multiple films
- Support the capabilities for a person to login as a customer or cinema manager
- Support the capabilities for a person to register as a new customer
- Support the capabilities for cinema to create films in booking center
- Support the capabilities for cinema manager to add films to cinemas for booking
- Support the capabilities to enumerate films in booking center
- Support the capabilities to acquire seats' status in arenas of cinemas and prompt the customer if the film is fully booked
- Support the capabilities to check the bookings of customers
- Support the capabilities to add comments to films
- Support the capabilities to retrieve comments of films
- Support the capabilities to use credit card for ticket payment
- Support the capabilities for the booking center to refresh the grade of a film

Chapter 5

System Design

5.1 IDL Design and Description

To accomplish the requirement of the system, the servers need to provide a number of operations and attributes for the clients to access. The utilization of CORBA architecture to implement the systems requires us to define the sets of operations and attributes by mean of an interface language, the **Interface Definition Language** (IDL). The IDL of the systems are described in the next pages.

The interface definitions of the server object are listed below:

```
// TicketBooking.idl

module TicketBooking {

// 1. exception declarations.
   exception InvalidCustomerException{};
   exception InvalidSeatException{};

// 2. structure for films' information.
   struct createFilmStruct
   {
      string title;
      string director;
```

```
        string casting;
        short hr;
        short min;
        string ctgy;
        string lang;
        string desc;
    };

// 3. structure for a string.
    struct filmNameStruct
    {
        string name;
    };

// 4. structure for customer’s information.
    struct createCustomerStruct
    {
        string uname;
        string hkid;
        string addr;
        string email;
        string hphone;
        string ophone;
        string cdtyp;
        string cdnum;
        short expyyear;
```

```
        short expymth;
        short expyday;
        string issucomp;
        string userid;
        string userpwd;
    };

// 5. structure for creating film’s showing schedule.
    struct createScheStruct
    {
        string cinemaname;
        string arenname;
        string filmname;
        string showdt;
        string showtme;
        string prices;
        string seatstus;
    };

// 6. stucture for bookings’ information.
    struct bookStruct
    {
        string filmname;
        string cinemaname;
        string arenname;
        string showdt;
```

```
        string showtme;
        string bookdt;
        string booktme;
        string bookseat;
        string prices;
    };

// 7. Array of string and schedule structure.
    typedef sequence<filmNameStruct> filmNameSeq;
    typedef sequence<createScheStruct> createScheSeq;

// 8. Interface definition and signatures of methods of //
the BookingCenter server object.
    interface BookingCenter {
// 8a.)
        boolean authenticateBooker(in string bookerId, in
        string pwd) raises(InvalidCustomerException);
// 8b.)
        boolean createFilm(in createFilmStruct
        createFilmData);
// 8c.)
        filmNameSeq allFilmName();
// 8d.)
        boolean createCustomer(in createCustomerStruct
        createCustomerData);
// 8e.)
```

```
        boolean createSche(in createScheSeq createScheData);
// 8f.)
        filmNameSeq getCineName(in string fnData);
// 8g.)
        filmNameSeq getArenaName(in string fnData, in string
        cnData);
// 8h.)
        filmNameSeq getDates(in string fnData, in string
        cnData, in string anData);
// 8i.)
        filmNameSeq getTimes(in string fnData, in string
        cnData, in string anData, in string aDate);
// 8j.)
        boolean makeBooking(in string userId, in filmNameSeq
        seatData, in string afilm, in string aCinema, in string
        aArena, in string aDt,  in string aTime, in string
        aPrice, out string reason)
        raises(InvalidSeatException);
// 8k.)
        string getSeatStatus(in string afilm, in string
        aCinema, in string aArena, in string aDt, in string
        aTime);
// 8l.)
        string getFilmPrice(in string afilm, in string aCinema,
        in string aArena, in string aDt, in string aTime);
// 8m.)
```

```
        bookStruct getBooking(in long bookRef, in string
        userId);
// 8n.)
        long getMaxBookRef(in string userId);
// 8o.)
        createFilmStruct getFilmInfo(in string filmName);
// 8p.)
        boolean createComment(in string filmName, in string
        comment, in long grade);
// 8q.)
        long getMaxCommentRef(in string filmName);
// 8r.)
        string getComment(in string filmName, in long
        cmmtRef);
// 8s.)
        long getAvgGrade(in string filmName);
// 8t.)
        string authenticateCinema(in string managerId, in
        string password);
// 8u.)
        filmNameSeq allArenaName(in string cinemaName);
// 8v.)
        boolean logout(in string userId);
// 8w.)
        boolean logoutMan(in string managerId);
    };
```

```
};
```

This is a point by point explanation of the meaning of the IDL.

1. **Exception declarations** – it defines two exceptions, invalid customer and invalid seat. The server object will throw these exceptions as long as the clients require the server to process on a customer or seat which is actually not existed.

2. **CreateFilmStruct** defines a structure, which contains a film's information. This structure allows the clients to pass films' information to server for films creating, it also let clients to retrieve information of films from the server.

3. **FilmNameStruct** defines a structure that contains only a string. When it is used together with filmNameSeq, which is a sequence of FilmNameStruct, allows clients and server to pass array of string to and forth.

4. **CreateCustomerStruct** defines a structure that contains information for registering a new customer to the cinema tickets reservation system.

5. **CreateScheStruct** is a structure for creating the showing schedule of films by a cinema manager. It contains important information such as show date, show time and prices.

6. **BookStruct** is a structure for customers to make booking or retrieve their bookings' information from the server. It contains important information to the clients such as show date, show time and their booked seats.

7. **CreateScheSeq** defines an array of CreateScheStruct which allows the cinema managers to create more than one schedule at a time. FilmNameSeq was explained in point 3.

8. **BookingCenter** server object – it is the heart of the system. It is responsible for all methods invoked by the clients and all interactions with the database. All server methods will be explained now.

8a.) Method **authenticateBooker** passes a user's ID and password to the server for authentication. If the password is correct, the method will return true to the clients, otherwise it will return false. If the user had not registered to the system, it will raise a invalid customer exception.

8b.) Method **createFilm** passes a film's data to the server for film creation. If film creation is successful, it returns true, otherwise it will return false.

8c.) Method **allFilmName** returns all films' names of films registered in the server. It returns all films' name by a string array.

8d.) Method **createCustomer** passes the personal particulars to the server for new customer registration. If the registration is successful, it returns true, otherwise it will return false.

8e.) Method **createSche** passes an array of schedule to the server to create showing schedule. If the schedule creation is successful, it returns true, otherwise it will

return false.

8f.) Method **getCineName** passes a film's name to server to get the names of all cinemas that are showing this film.

8g.) Method **getArenaName** passes a film's name and a cinema's name to the server to get the names of all arenas that are showing this film by the cinema.

8h.) Method **getDates** passes a film's name, a cinema's name and an area's name to the server to get the exact showing dates.

8i.) Method **getTimes** passes a film's name, a cinema's name, an area's name and a date to the server to get the exact showing times.

8j.) Method **makeBooking** passes the user ID, the seats' label a film's name, a cinema's name, a arena's name, a date, a time and the price/seat to the server to make booking. If the booking is successful, it returns true, otherwise it will return false. If the booking failure was due to the seats had been occupied, it will return the reason of failure. If the seat label is invalid, it will raise an invalid seat exception.

8k.) Method **getSeatStatus** passes a particular film's name, a cinema's name, a arena's name, a date, a time to the server to get the current seats' status.

8l.) Method **getFilmPrice** passes a particular film's name, a cinema's name, a arena's name, a date, a time to the server to get a seat's price of a film.

8m.) Method **getBooking** passes a booking reference number and a user's ID to get a booking's information from the server.

8n.) Method **getMaxBookRef** gets the maximum booking reference of a user.

8o.) Method **getFilmInfo** gets a film's information by its name.

8p.) Method **createComment** passes a film's name, the comment and the grade given by the user too the server to create a comment. It returns true if success, otherwise it returns false.

8q.) Method **getMaxCommentRef** gets the maximum comment reference of a film.

8r.) Method **getComment** gets a comment of a film by a comment reference number.

8s.) Method **getAvgGrade** gets the average grade of a film from the server.

8t.) Method **authenticateCinema** passes a manager's ID and password to the server for authentication.

8u.) Method **allArenaName** gets the names of arenas of a cinema.

8v.) Method **logout** passes a user's ID to the server to logout. It returns true if success, otherwise it returns false.

8w.) Method **logoutMan** passes a manager’s ID to the server to logout. It returns true if success, otherwise it returns false.

5.2 User Interfaces

For the client to interact with the system, some user interfaces are required in the system. The screen design is shown in the following figure.

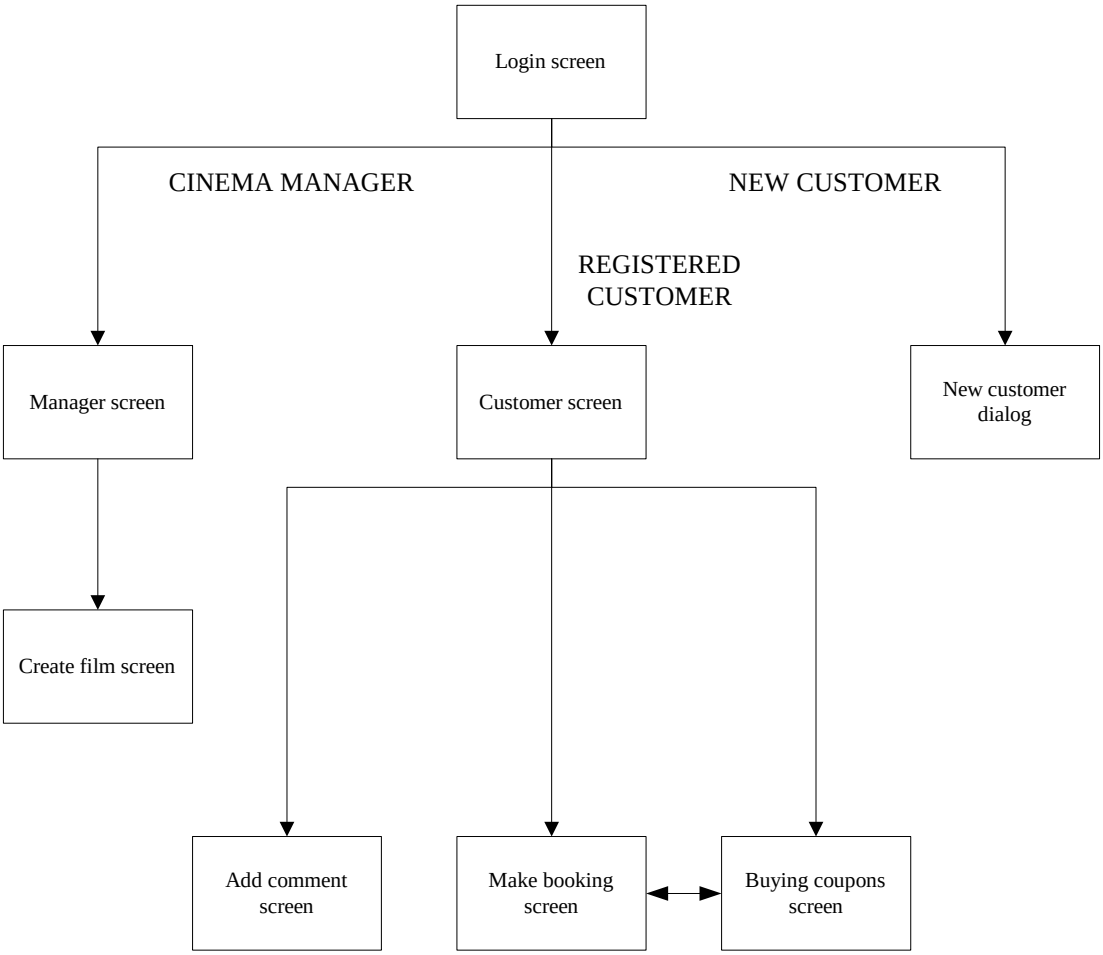


Figure 5.1 Screen Flow

User interfaces are the front-end that responsible for the interaction between users and the system. It allows users to perform input and retrieval of information. This section will introduce the system's user interfaces in detail.

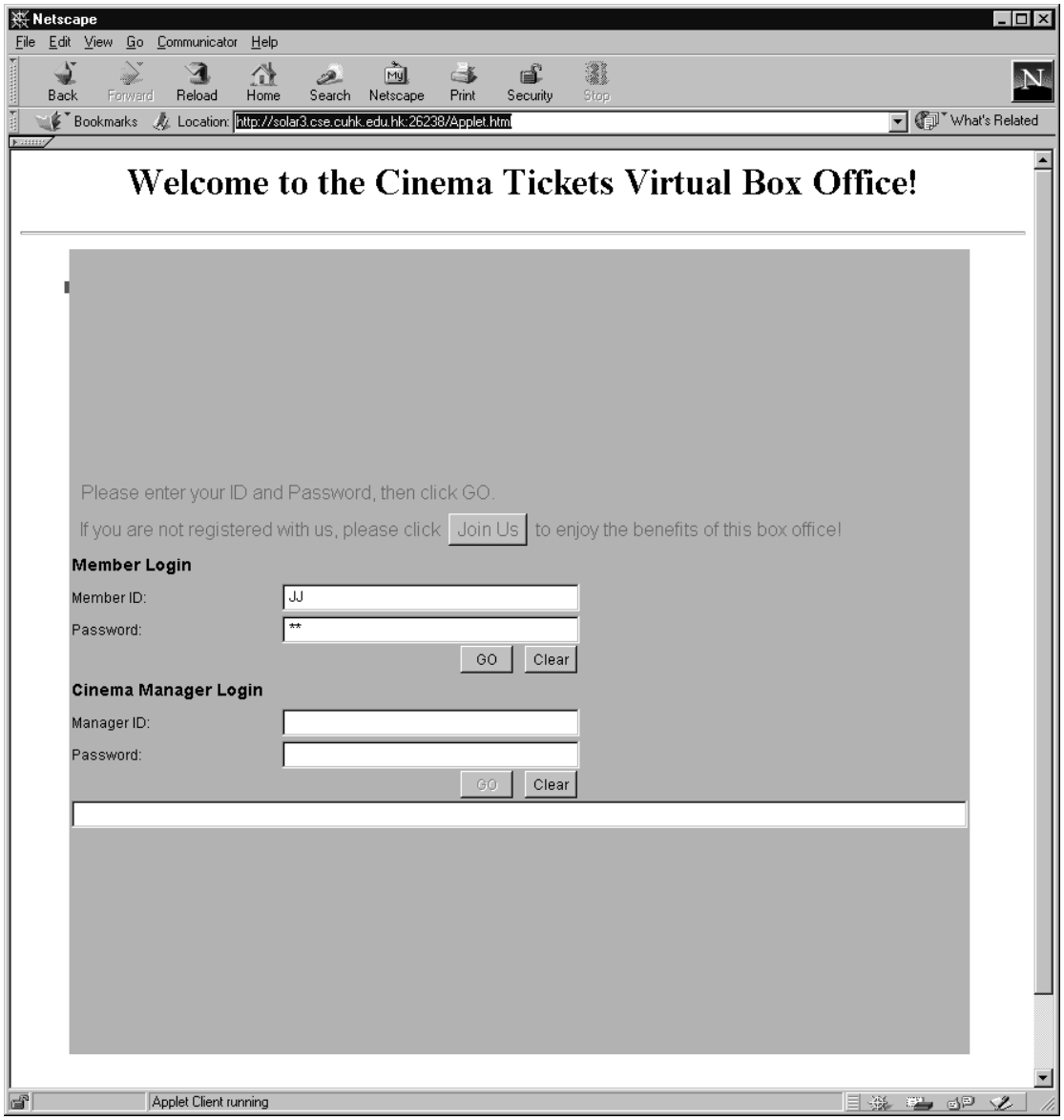


Figure 5.2

Figure 5.2 shows the login screen of the cinema ticket reservation system. As the

system allows general users to buy tickets or cinema manages to create and schedule films for show on behalf of cinemas. Hence, it allows the login of a user or a cinema manager. If a new user comes to visit this web page, he/she can register as a new user by pressing the Join Us button. After pressing the Join Us button, a dialog figure 5.3 will show up for the new user to fill in some personal information. If the new user has completed filling the information and pressed the OK button, he/she can use the system immediately with the new user ID and password. He/she can cancel the registration by pressing the Cancel button.

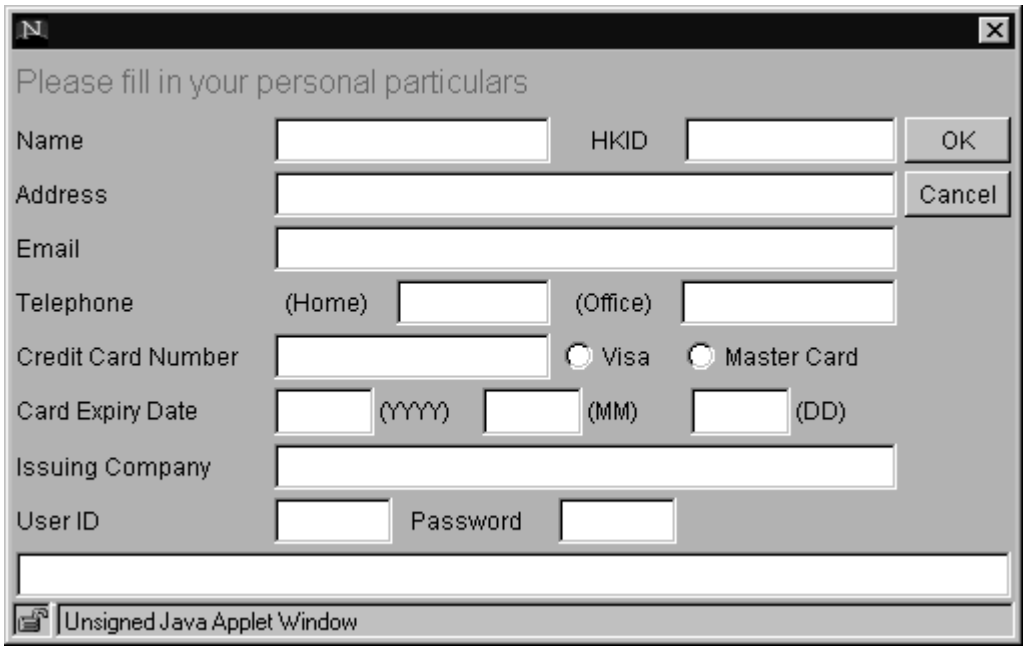


Figure 5.3

After a user has login the system, figure 5.4 is the first screen shown up. From this screen, the user can select a film to see its information. Besides, the official information, users can know more about the films by viewing the comments posted by other users. The Overall Grade shows the average grade of this film given by all users. Point 5

represents excellent, point 1 represents bored, then the user will has some idea about this film. The user can click the Last or Next buttons to retrieve a comment. The user also can post a new comment by clicking the Add Comment button.

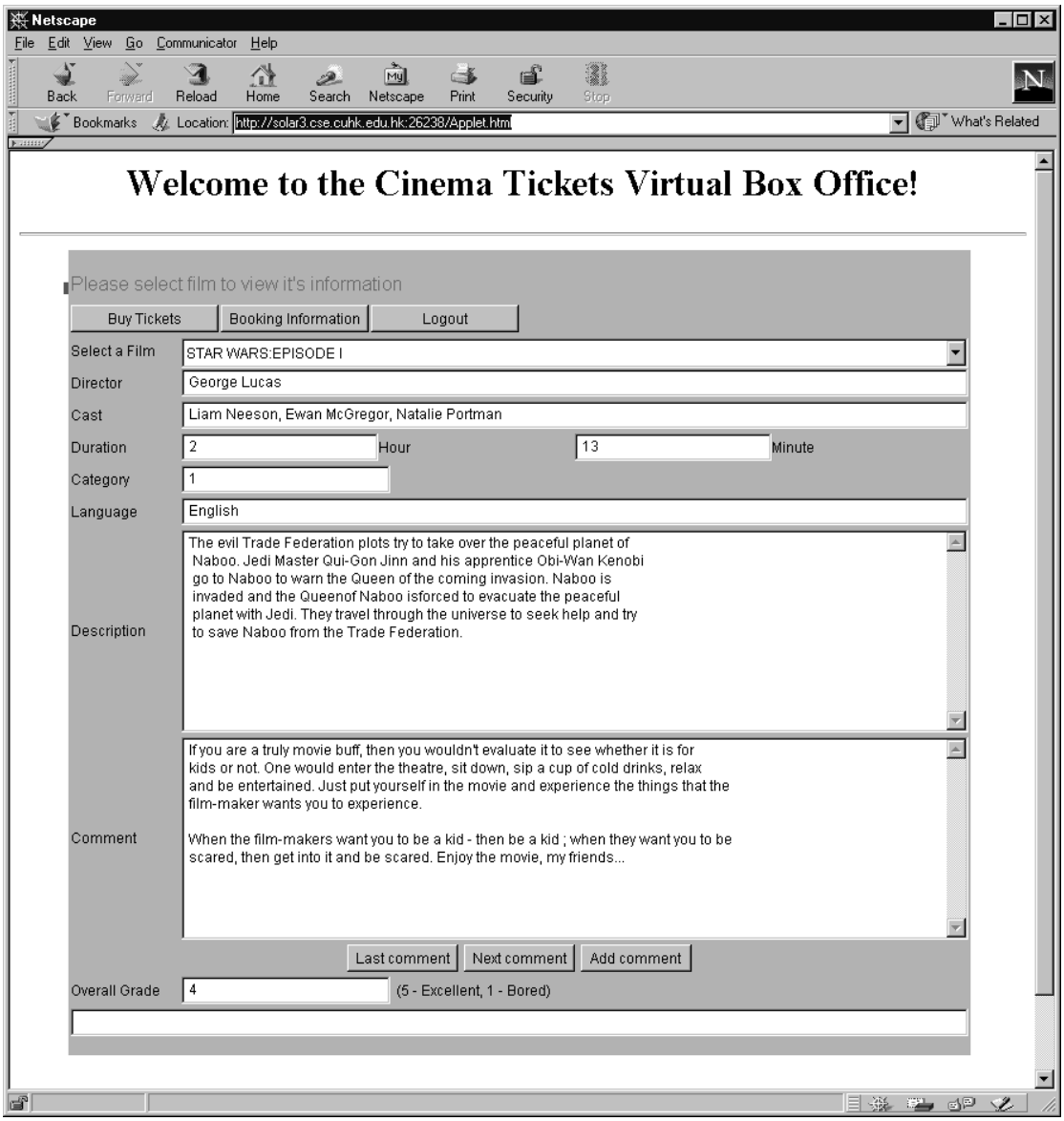


Figure 5.4

Figure 5.5 is the screen for the user to post comment and grade to a film. The user can press OK button to post it or press Cancel button to cancel.

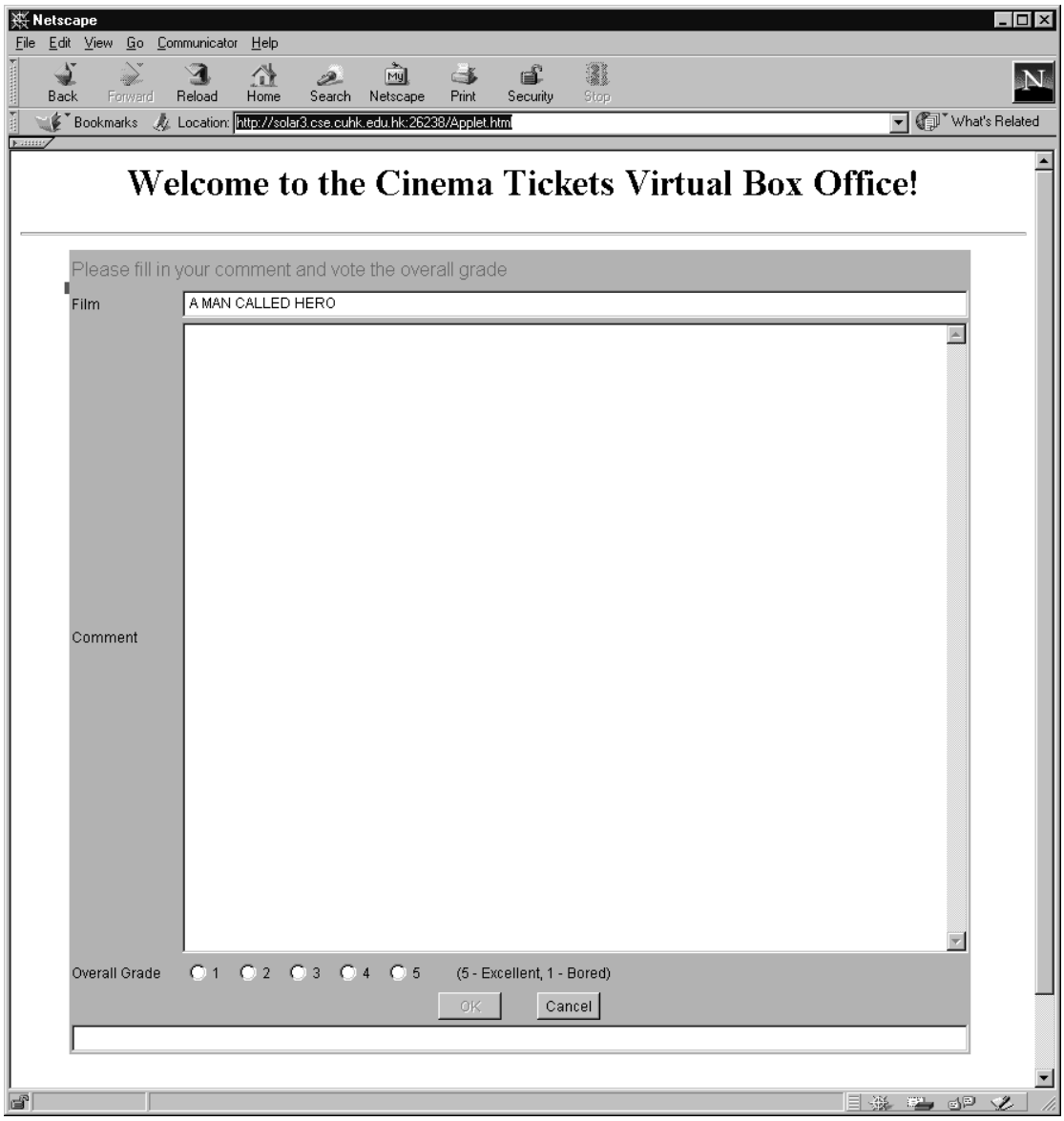


Figure 5.5

From figure 5.4, if the user presses the Buy Tickets button, the screen below will show up. To buy tickets, the users need to select a film first, secondly, he/she needs to select a cinema, then a area of that cinema. After that he/she need to select a date and time. If all selection has been made, he/she can go to the seating plan by clicking the Seating Plan button.

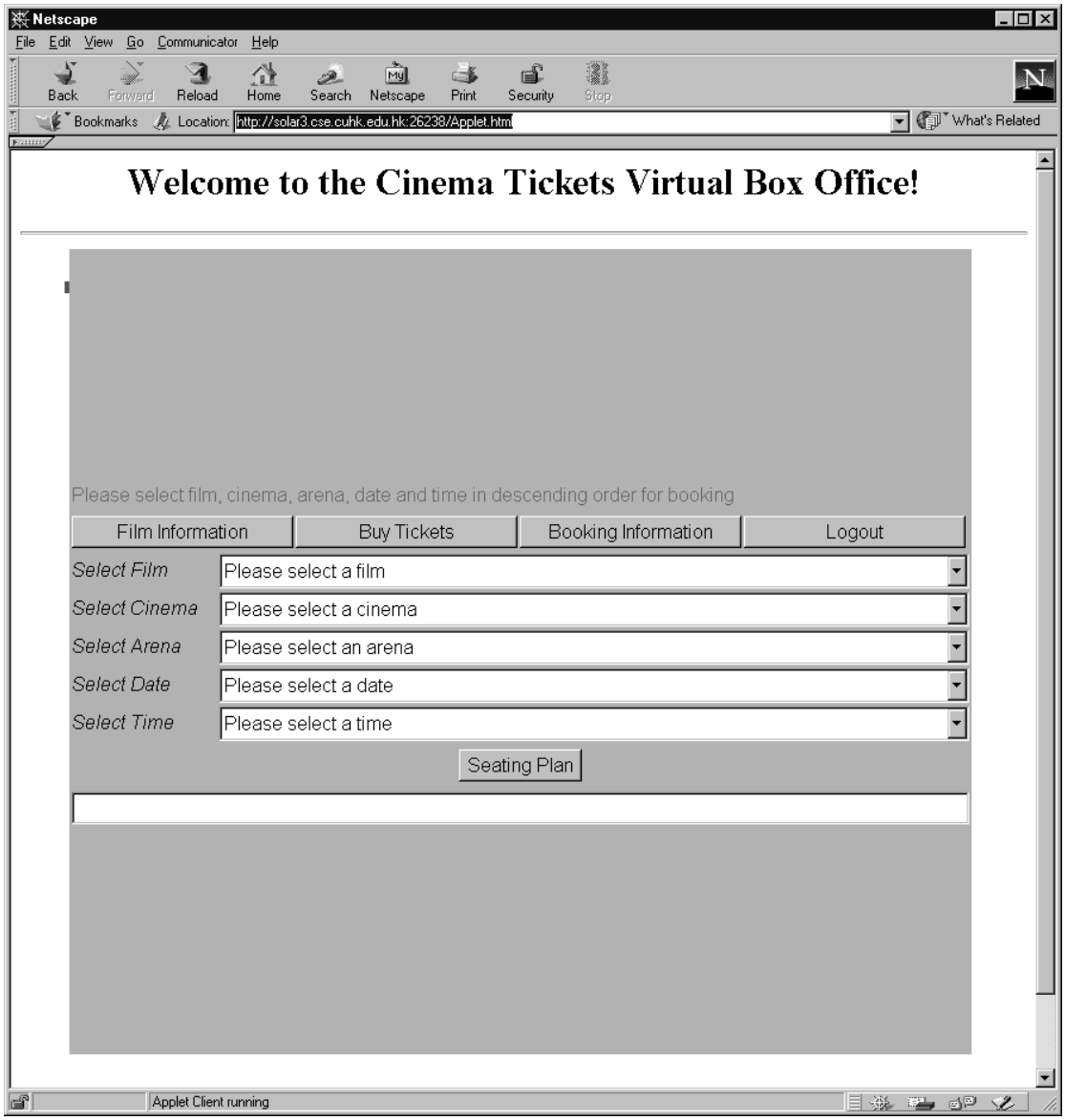


Figure 5.6

Figure 5.7 show the seating plan the user selected. The top shows the information of the show. The middle shows the seats, seats available are yellow in color while seats booked are in deep blue color. To select a seat, the user just need to click on that seat, its color will change to deep blue and the seat's label will be added to the list at lower left corner. The total amount of all selected seats will accumulate automatically. The user can deselect a seat by clicking on that seat again. The user click Confirm Booking button to confirm or Cancel button to cancel all selection.

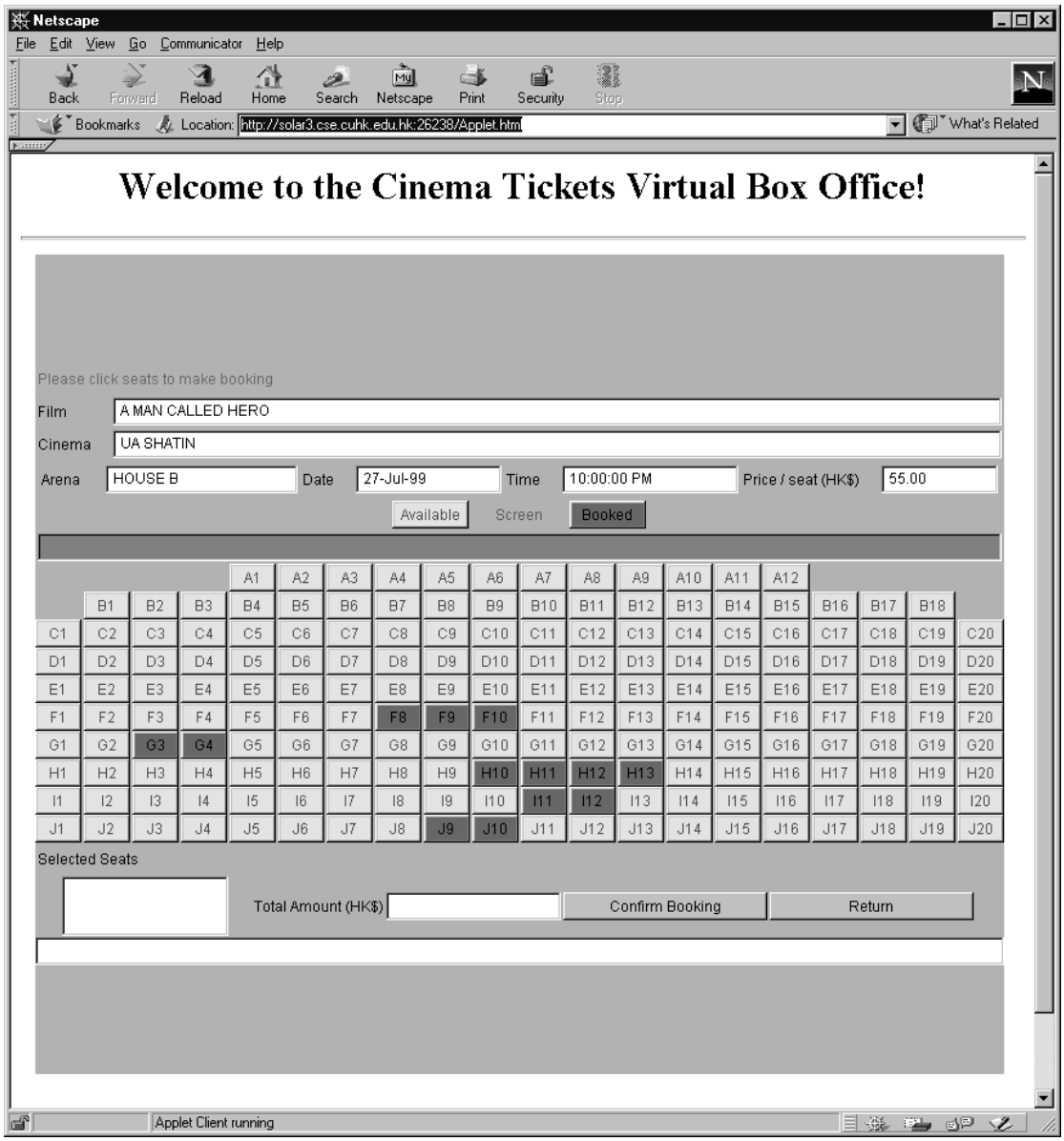


Figure 5.7

After buying tickets, the user can check the booking by pressing the Booking Information button. Figure 5.8 is the booking information screen that shows all the booking made by a user, the latest booking shown first and the oldest booking shown last. The user can click the Last or next buttons to view the booking details. The charges of booking of the user will be settled monthly by credit card payment.

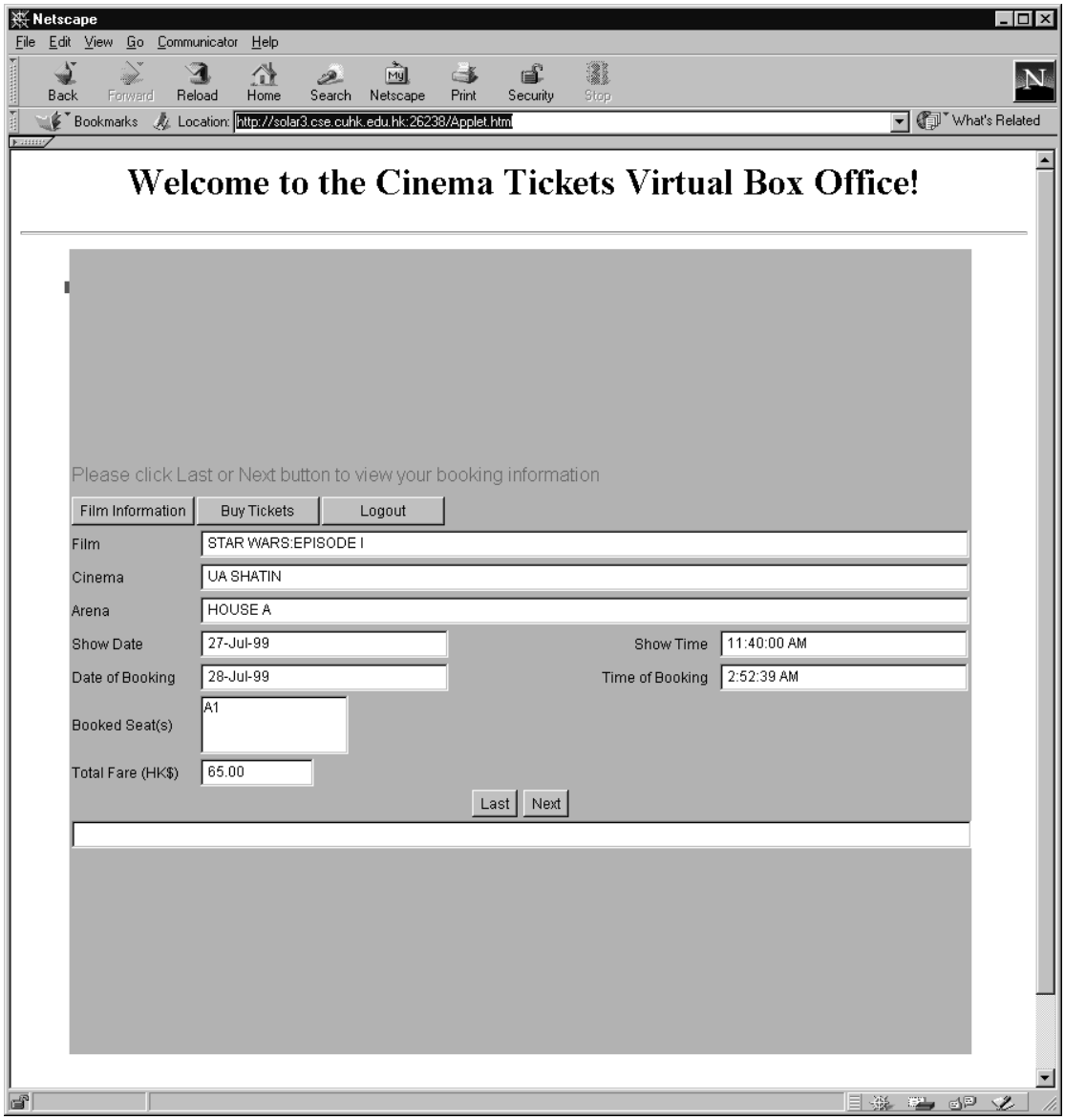


Figure 5.8

The cinema managers can login into the system to schedule the films' show on behalf on a cinema. A film must be registered before it can be scheduled. Hence, when a manager login, the first screen show up contains information of all registered films. Figure 5.9 is the screen that allows the manager to select a film and view its information.

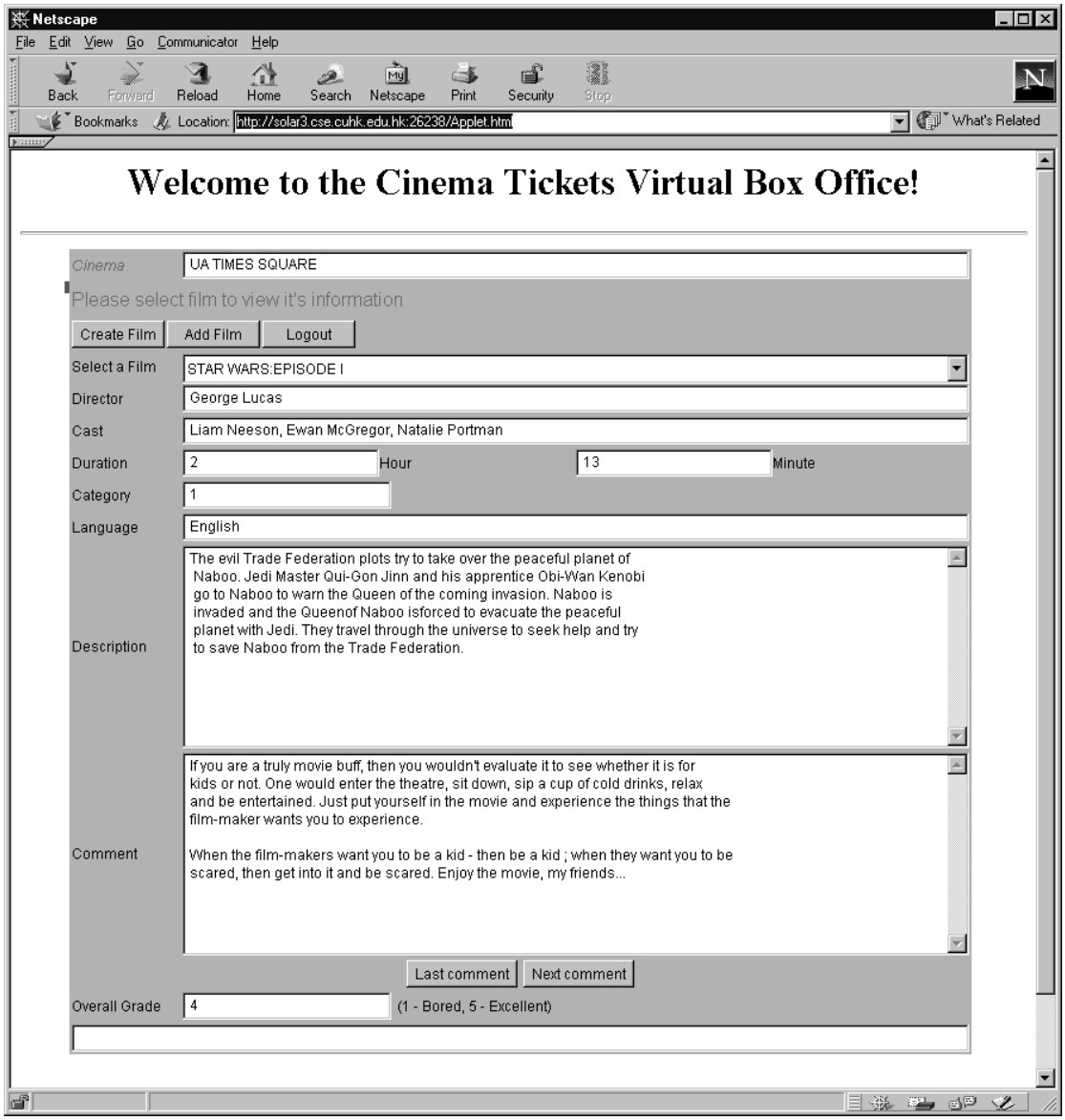


Figure 5.9

If the manager cannot find the film for scheduling from the previous screen, then he/she needs to create the film. Figure 5.10 is the screen for film creation and registration. The managers just need to fill in the information of a new film and press the OK button. Cancellation of film creation can be done easily by pressing the Cancel button.

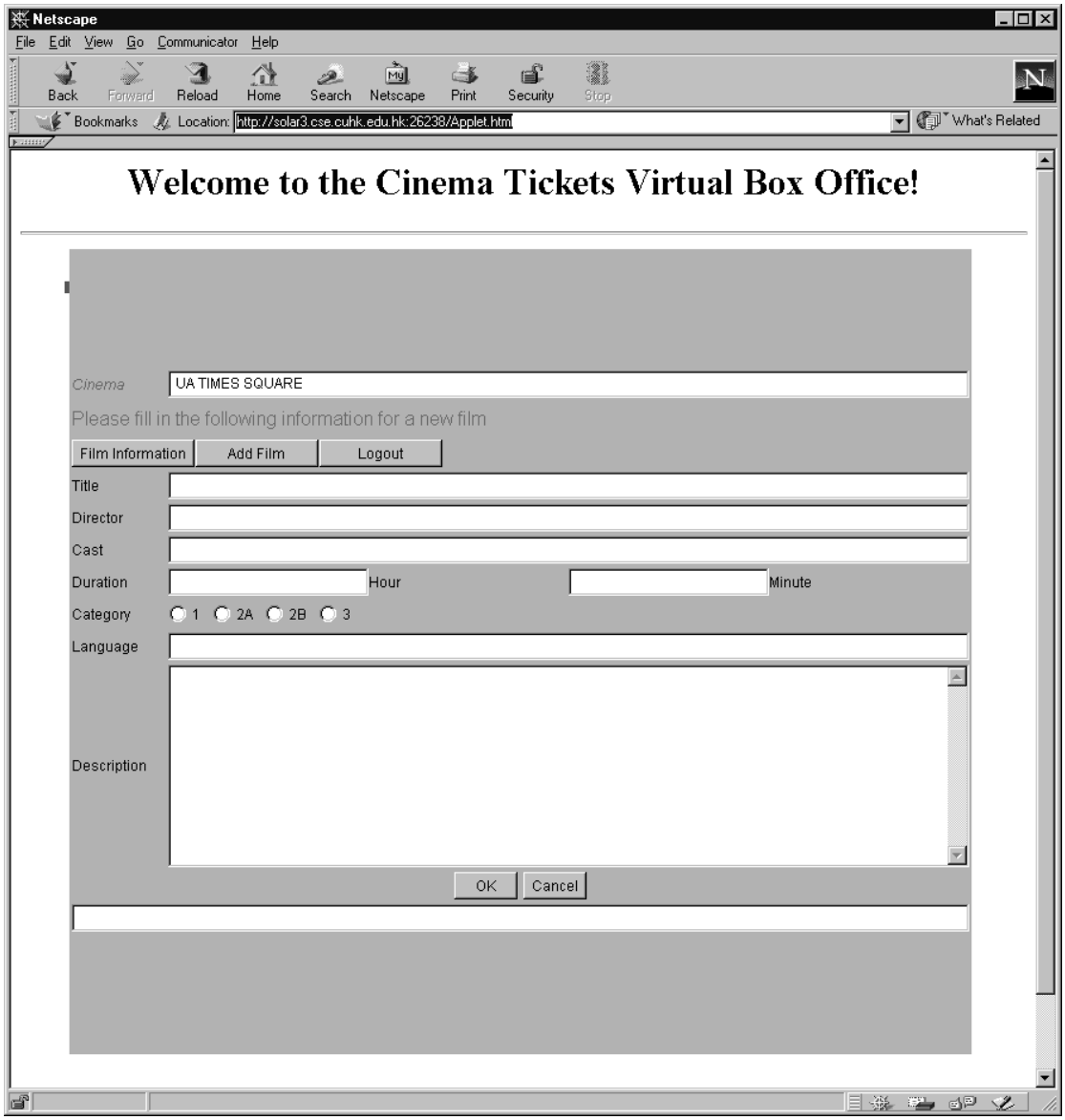


Figure 5.10

Figure 5.11 shows the screen for scheduling. The manager first select a film and an arena, then he/she need to fill in the future range of showing dates. Afterwards, the manager need to fill in the available showing times within each showing date. Moreover, the managers also need to fill in the price per seat before pressing the OK button to confirm the schedule. Managers can cancel the scheduling simply by pressing the Cancel button.

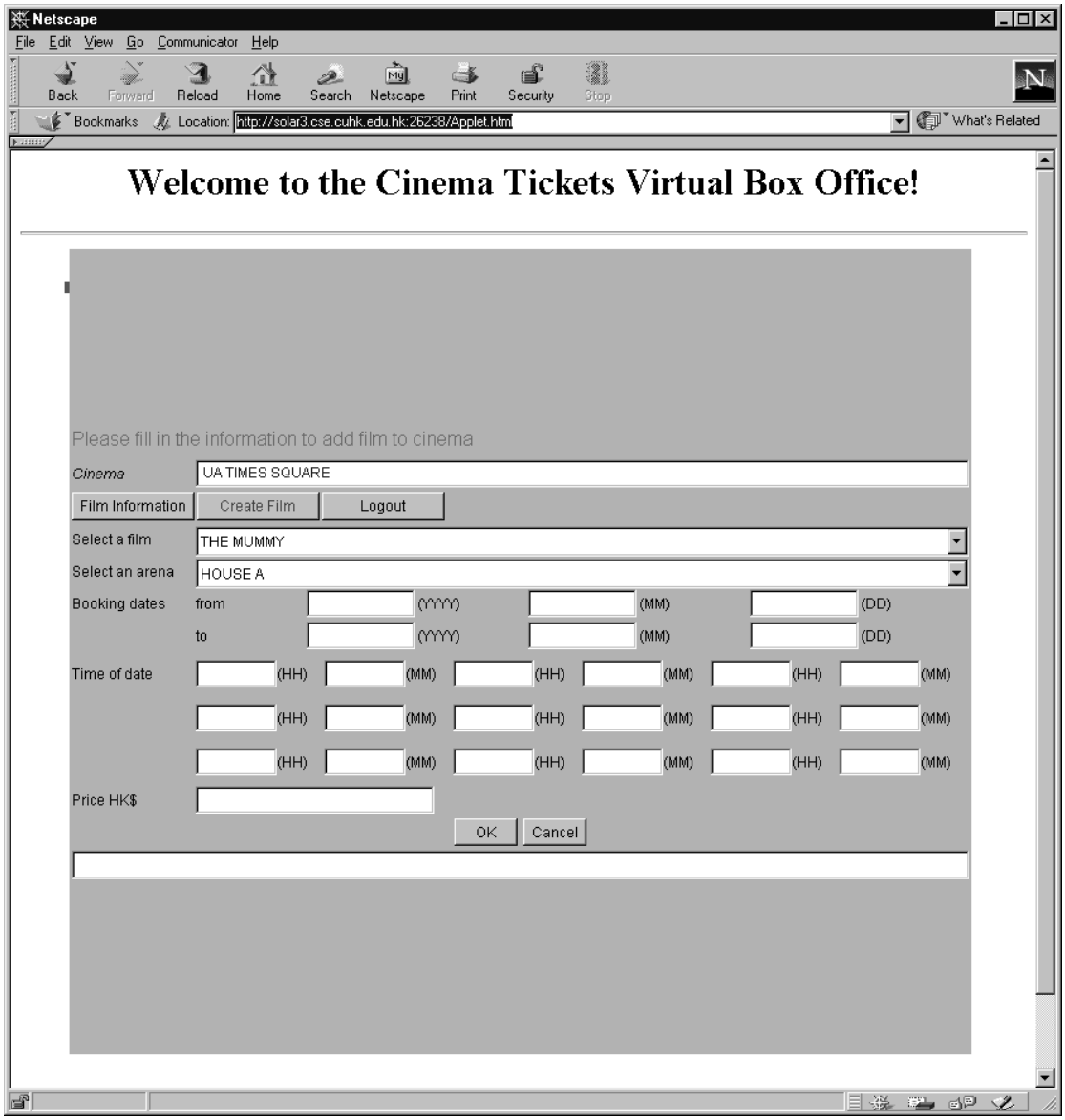

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://solar3.cse.cuhk.edu.hk:26238/Applet.html What's Related
Welcome to the Cinema Tickets Virtual Box Office!
Please fill in the information to add film to cinema
Cinema
UA TIMES SQUARE
Film Information
Create Film
Logout
Select a film
THE MUMMY
Select an arena
HOUSE A
Booking dates
from
to
(YYYY)
(MM)
(DD)
Time of date
(HH)
(MM)
Price HK$
OK
Cancel

Figure 5.11

5.3 Database Design

The following database tables are used to store the data of the application.

Table: Film

A record in this table stores the information of a film.

Primary key (filmrefno)

Attribute	**Type**	**Length**
filmrefno	int not null	38
title	char not null	50
director	char not null	50
casting	varchar2 not null	200
hr	smallint not null	38
min	smallint not null	38
ctgy	char not null	2
lang	char not null	100
description	varchar2 not null	3000

Table: Customer

A record in this table stores the information of a customer.

Primary key (userid)

Attribute	**Type**	**Length**
username	char not null	50
hkid	char not null	20
addr	varchar2 not null	200
email	char not null	50
hphone	char not null	10
ophone	char not null	10
cdtyp	char not null	20
cdnum	char not null	16
expyyear	smallint not null	38

expymth	smallint not null	38
expyday	smallint not null	38
isscomp	varchar2 not null	200
userid	char not null	20
userpwd	char not null	20

Table: Schedule

A record in this table stores the information of a showing schedule.

Primary key (scheduleno)

Attribute	**Type**	**Length**
scheduleno	int not null	38
cinemaname	char not null	50
arenaname	char not null	20
filmname	char not null	50
showdt	char not null	10
showtime	char not null	15
price	char not null	10
seatstus	char not null	300

Table: Booking

A record in this table stores the information of a booking.

Attribute	**Type**	**Length**
userid	char not null	20
bookref	int not null	38
filmname	char not null	50
cinemaname	char not null	50
arenaname	char not null	20
showdt	char not null	10
showtime	char not null	15
bookdt	char not null	10
booktime	char not null	15
bookseat	char not null	1000
totprice	char not null	10

Table: Comment

A record in this table stores the information of a comment.

Attribute	Type	Length
commentrefno	int not null	38
title	char not null	50
commt	varchar2 not null	3000
grade	int not null	38

Table: Cinema

A record in this table stores the information of a cinema.

Attribute	Type	Length
cinemaname	char not null	50
arenaname	char not null	20
managerid	char not null	20
managerpwd	char not null	20

Chapter 6

Implementation of the system

6.1 Development environment

The system was developed using Java to implement both the client side and the server side. Inprise VisiBroker for Java 3.4 was used as the ORB product. VisiBroker provides pure Java implementation of ORB and a complete IDL-to-Java language binding. Moreover, VisiBroker comes with an interesting and useful utility – the Gatekeeper. It is an IIOP proxy and can be used to wrap IIOP messages into HTTP messages. I used SGI as the development platform. The client was tested on Windows and UNIX platform with Netscape Communicator 4.5 and 4.61.

6.2 Top-level view

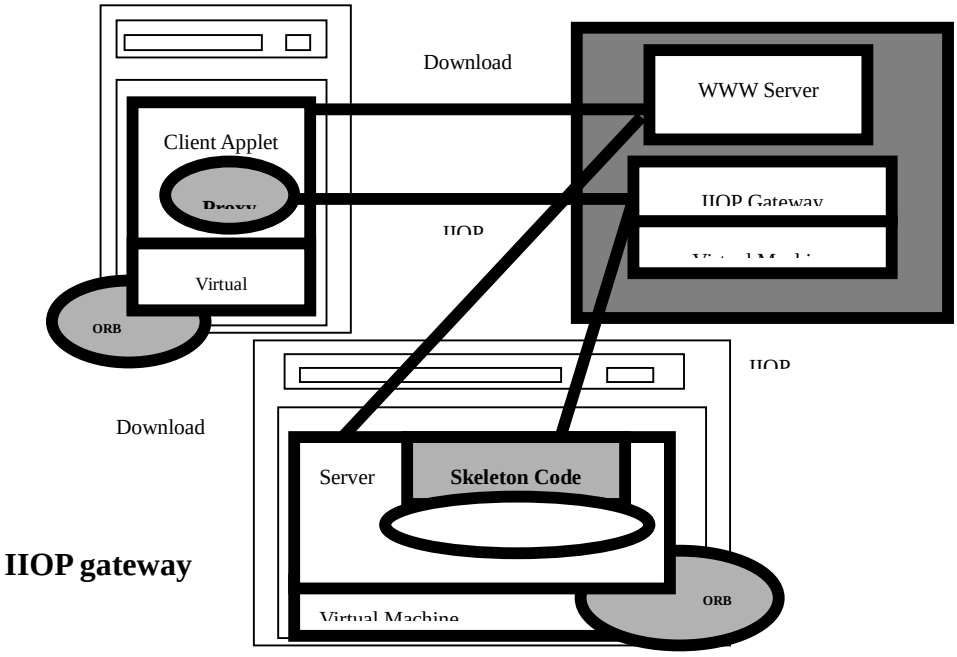
The Cinema Tickets Reservation System consists of the following components:

Database server – all persistent data includes films' information, customers' information and cinemas' information are stored in a relational database for permanent storage. I use Oracle8 release 8.0.5 as the database management system. Java DataBase Connectivity (JDBC) was used as an interface for the communication between the server object and the database. The database is the third-tier, only the server object will interact with it and it is therefore transparent to the clients.

Object server – the object server provide services to the clients. The clients use those services by method invocation, it need the help of IIOP proxy which will be explained later. It must be started up and running before the clients can start to use the services, once the server object is up, it is connected to the database server.

Web server - It lets clients to download web pages and the client applet through HTTP.

IIOP proxy — There is a security restriction, known as “sandbox security restriction”, imposed on each Java applet that the applet is allowed to have network communication only with the host from which the applet is originated. This restriction



is fatal to a distributed system since the essence of distributed computing comes from the cooperation of hosts on a network. A solution to this restriction is to use IIOP proxy on the web server host and let it routes all IIOP messages on behalf of the applet. From the view of applet, it is only communicating with the web server host. However, to other CORBA objects, the applet is perceived as if it resides on the same with network

and they do not aware that they are communicating with an applet. The VisiBroker version of the IIOP proxy, named Gatekeeper, have an extra function built-in—HTTP tunneling. This function allows the client to be executed in a firewall-protected network. The idea of IIOP proxy is illustrated in the figure 6.1.

Figure 6.1

Client applet – This is the front-end to the users. For the cinema tickets reservation system, this is the only visible part to users. It is mainly used to accept user command, transform the user request into invocations of remote methods and displayed the result return from the object server.

The top-level system architecture is shown on the following diagram.

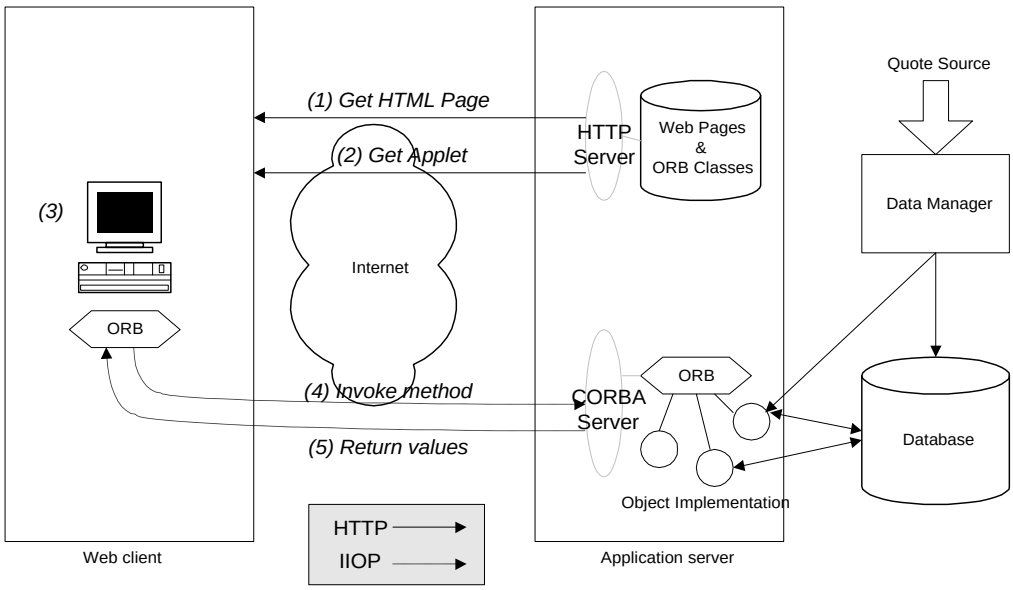


Figure 6.2

1. **Web browser downloads HTTP page** — In our case, the page includes reference to embedded client Java applet.
2. **Web browser retrieves Java applet and ORB classes from HTTP server** — The HTTP server retrieves the applet and ORB classes into the browser in the form of Java bytecodes.
3. **Web browser loads and starts applet** — The applet is first run through the Java run-time security gauntlet and then loaded into memory.
4. **Applet invokes CORBA server objects** — The Java applet includes IDL-defined objects. Invoking methods on these objects will be directed to the server implementation through IIOP. In fact, the client does not communicate with the object server directly due to the Java sandbox model restriction. The Gatekeeper makes it possible to let client to invoke an object server on a host other the one from which the applet originated. The Gatekeeper also wraps IIOP messages into HTTP message while transmitting through Internet.
5. **CORBA server return result values** — The return value of the method and values of parameters defined to be “out” type are sent to client through IIOP. Again, the IIOP messages are wrapped in HTTP messages during the transmission.

6.3 Server Objects

The cinema tickets reservation system is implemented as a 3-tier client/server application, Java-driven client applets invoke operations on the CORBA middle-tier server object via an IIOP ORB. The server object provides the business logic and stores their persistent data in a JDBC-compliant SQL database.

In the 3-tier application CORBA clients talk to CORBA server objects. The server objects in turn talk to one DBMS via JDBC. A new applet class called Client provides the client. The middle-tier server consists of three main classes: 1) a **BookingCenterMain** that start and manage a server object, 2) a server object of class **BookingCenterImpl**, and 3) a helper object of the class **BookDb** – this worker object with persistent JDBC connection. The third tier consists of the JDBC database – this is where the persistent state is stored.

Here are the Java classes a 3-tier cinema tickets server supports:

BookingCenterMain provides the main method for the server. It provides the following functions: 1) initializes the ORB, 2) obtains a reference to the BOA, 3) invokes *obj_is_ready* to register the newly created **BookingCenterImpl** object, and 4) invokes *impl_is_ready* to tell the ORB this server is ready for business.

BookingCenterImpl implements the IDL-defined interface. The class constructor creates a new **BookDb** object and then connects to it. The object is preconnected to the database. The class implements all IDL-defined methods. Each of these methods services a client request with a corresponding helper method on the **BookDb** object.

BookDb is a database-encapsulator class: it handles all the interactions with JDBC. The class provides twenty-two methods: *connect, closeConnection, createFilm, getFilms, createCustomer, createSchedule, getBookerPwd, getCinemaNames, getArenaNames, getDates, getTimes, getSeatStatus, serSeatStatus, getFilmPrice, insertBooking, getMaxBookRef, getBooking, getFilmInfo, createComment, getMaxCommentRef,*

getComment, *getAvgGrade*. Notice that the last twenty methods have a one-to-one correspondence to their **BookingCenterImpl** counterparts.

Chapter 7

Testing

This section shows some test cases and the expected results for testing the correctness and efficiency of the system. As all the testing conditions and expected results are satisfied, the correctness of the system is quite satisfactory. The speed of the system is fine if running on UNIX machine, but it runs even faster on Windows NT with Netscape browser version 4.07.

Test Cases	Conditions	Result
1	Correct login ID and password	Login successfully

Test Cases	Conditions	Result
2	Wrong password	Login fail, display message Customer password incorrect, please checks.

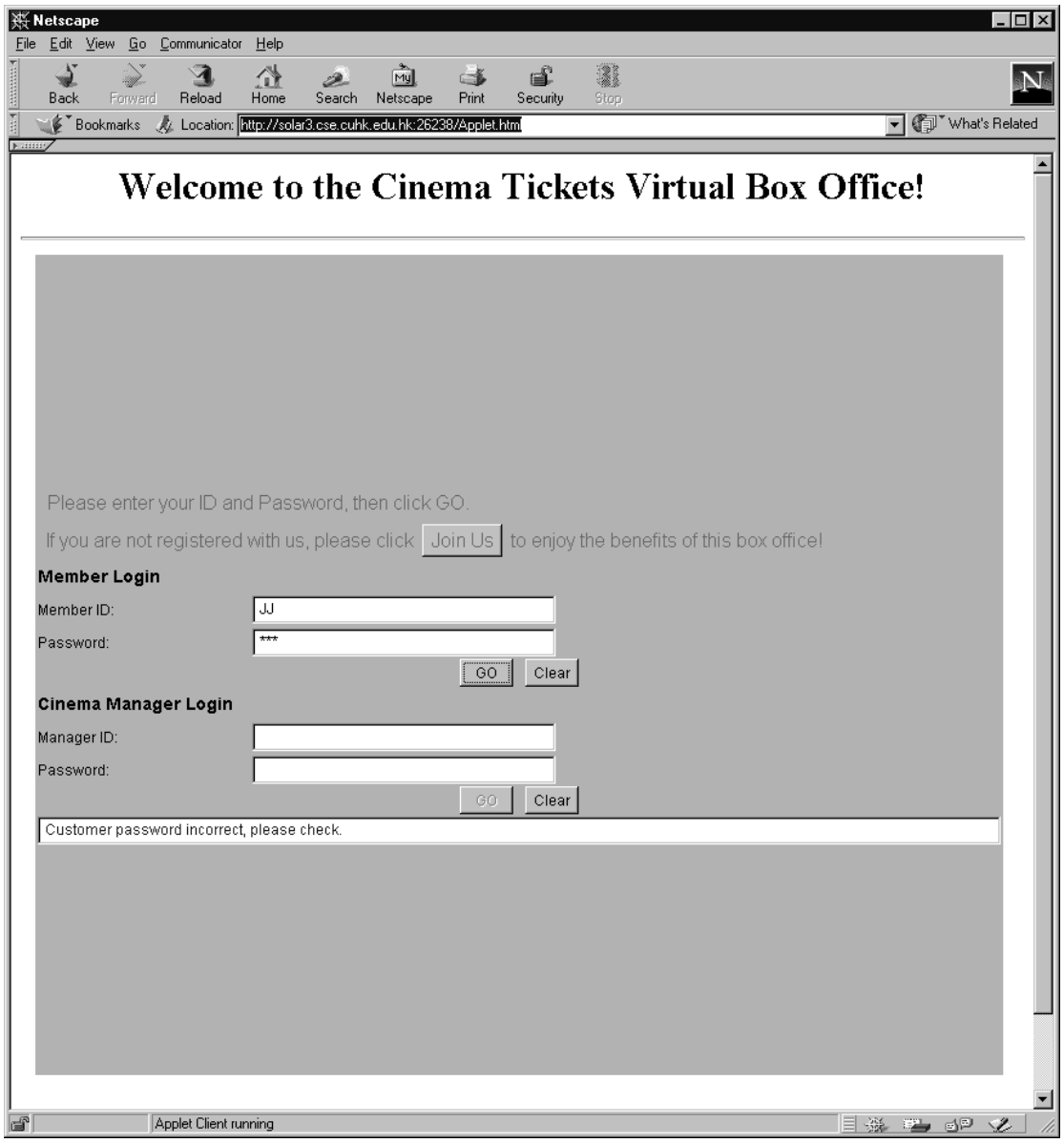


Figure 7.1

Test Cases	Conditions	Result
3	Users or managers not registered	Login fail, display message Manager ID not found

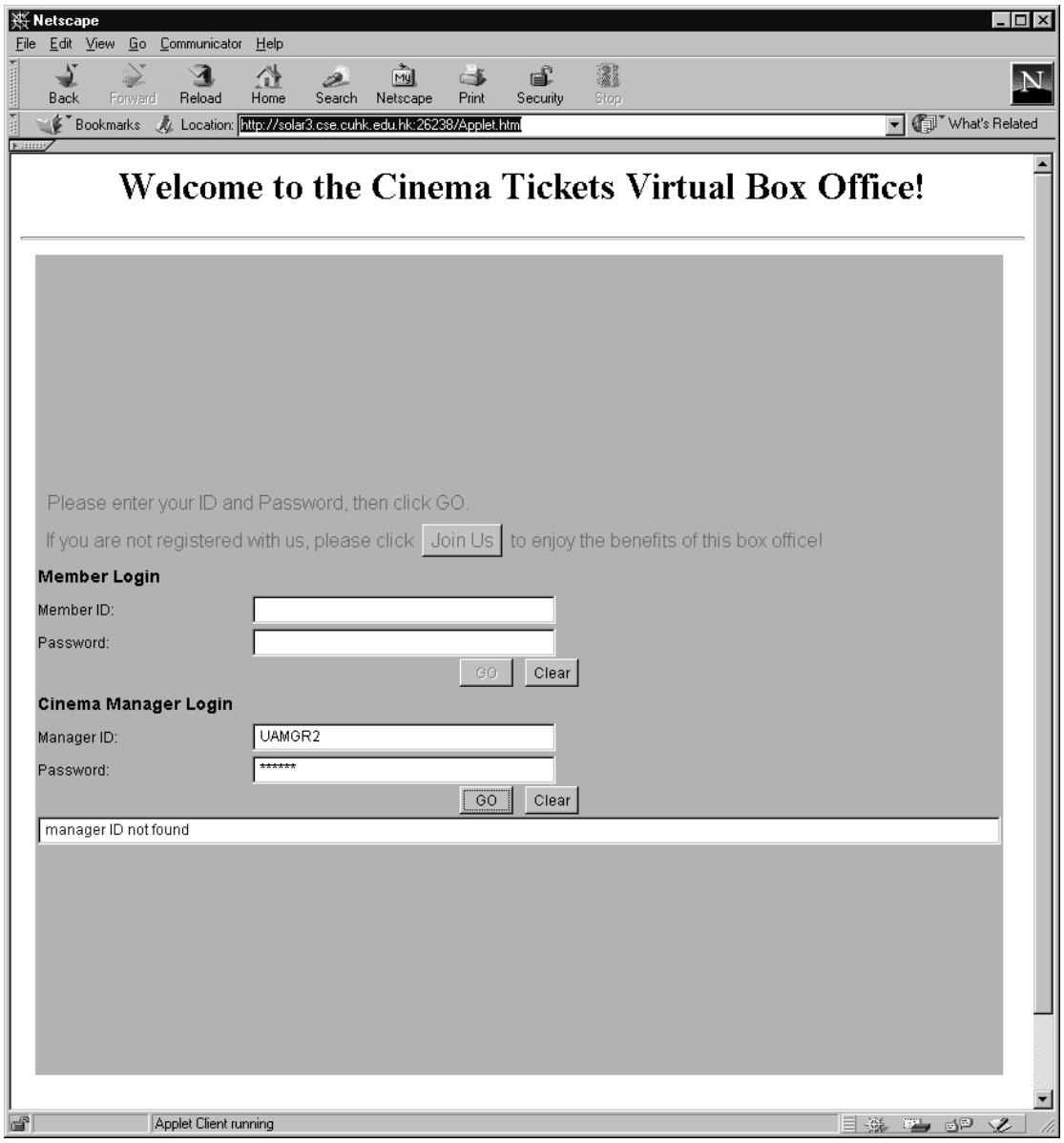


Figure 7.2

Test Cases	Conditions	Result
4	Users or managers login two clients concurrently	First login success, second login fail, display message Manager has already login.

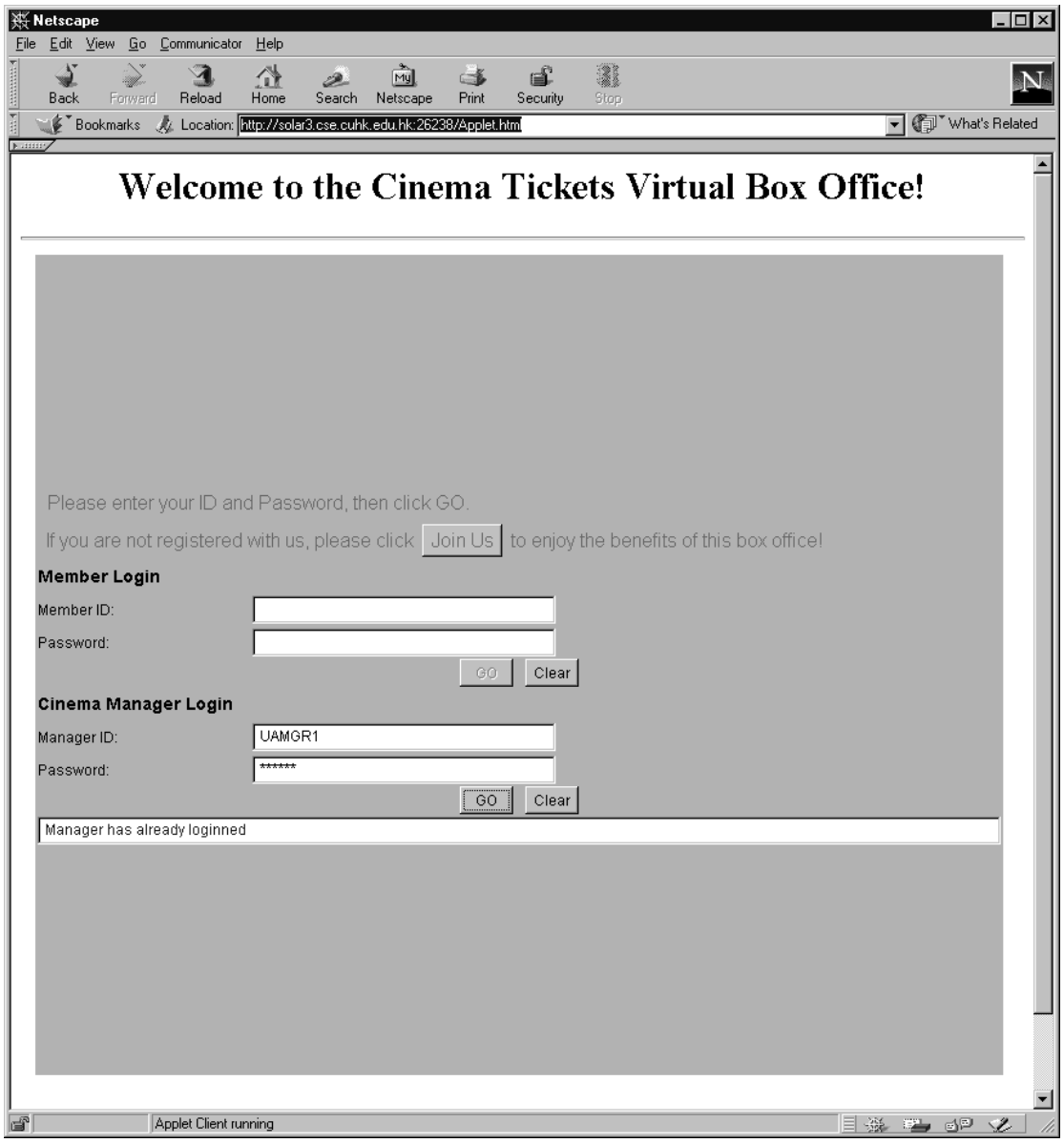


Figure 7.3

Test Cases	Conditions	Result
5	Select different film	Display different films' details

Test Cases	Conditions	Result
6	Select last/next comment	Display different comments

Test Cases	Conditions	Result
7	Add a new comment and grade	Other users can view the comment

		and grade immediately

Test Cases	Conditions	Result
8	Click on seats	Seats' labels are added to list and total amount = no. of seat clicked × price/seat. Seats' color changed to deep blue

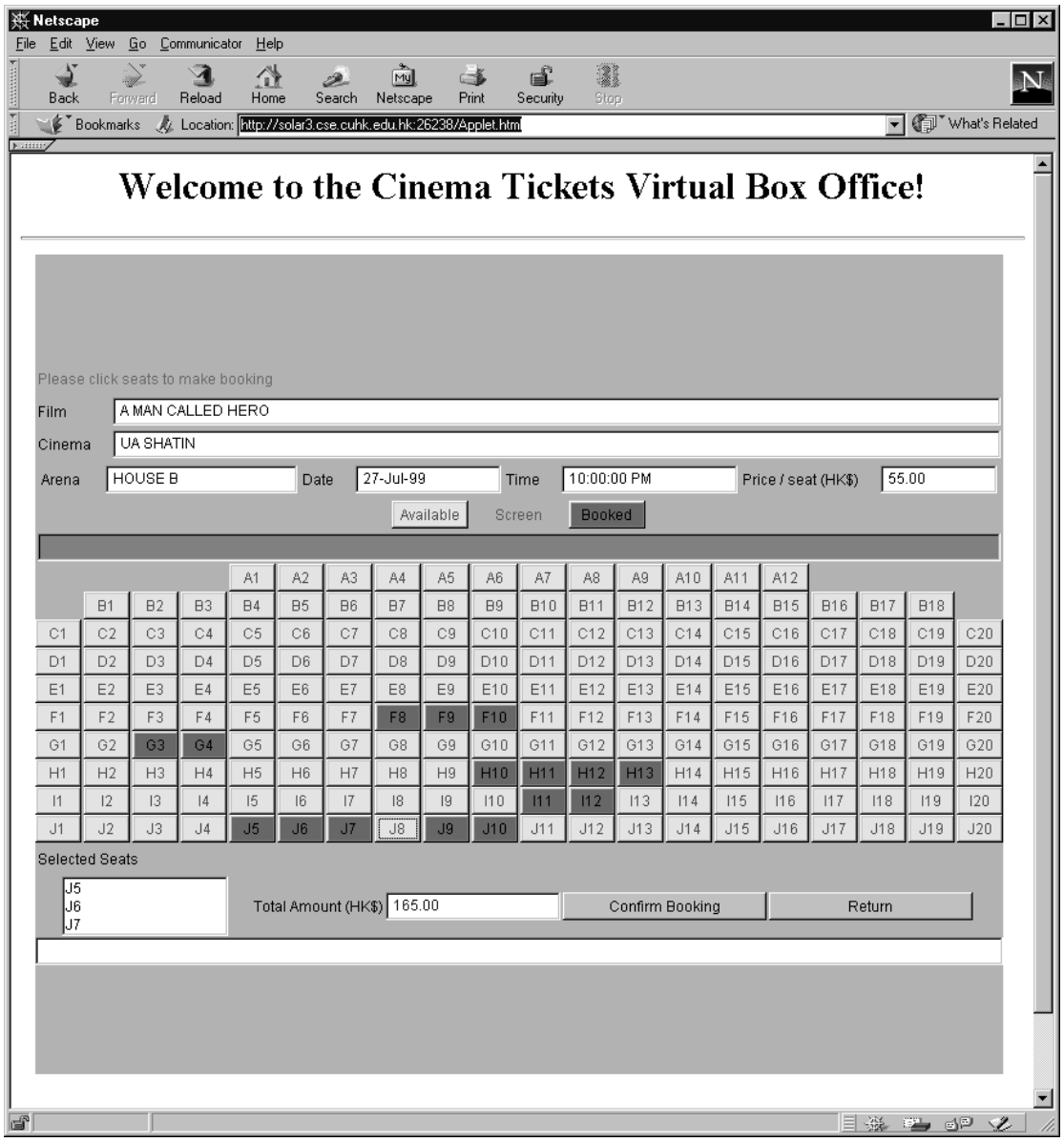


Figure 7.4

Test Cases	Conditions	Result
9	Click on previous selected seats	Seats' labels are removed from list and total amount is reduced correctly. Seats' color changed to yellow

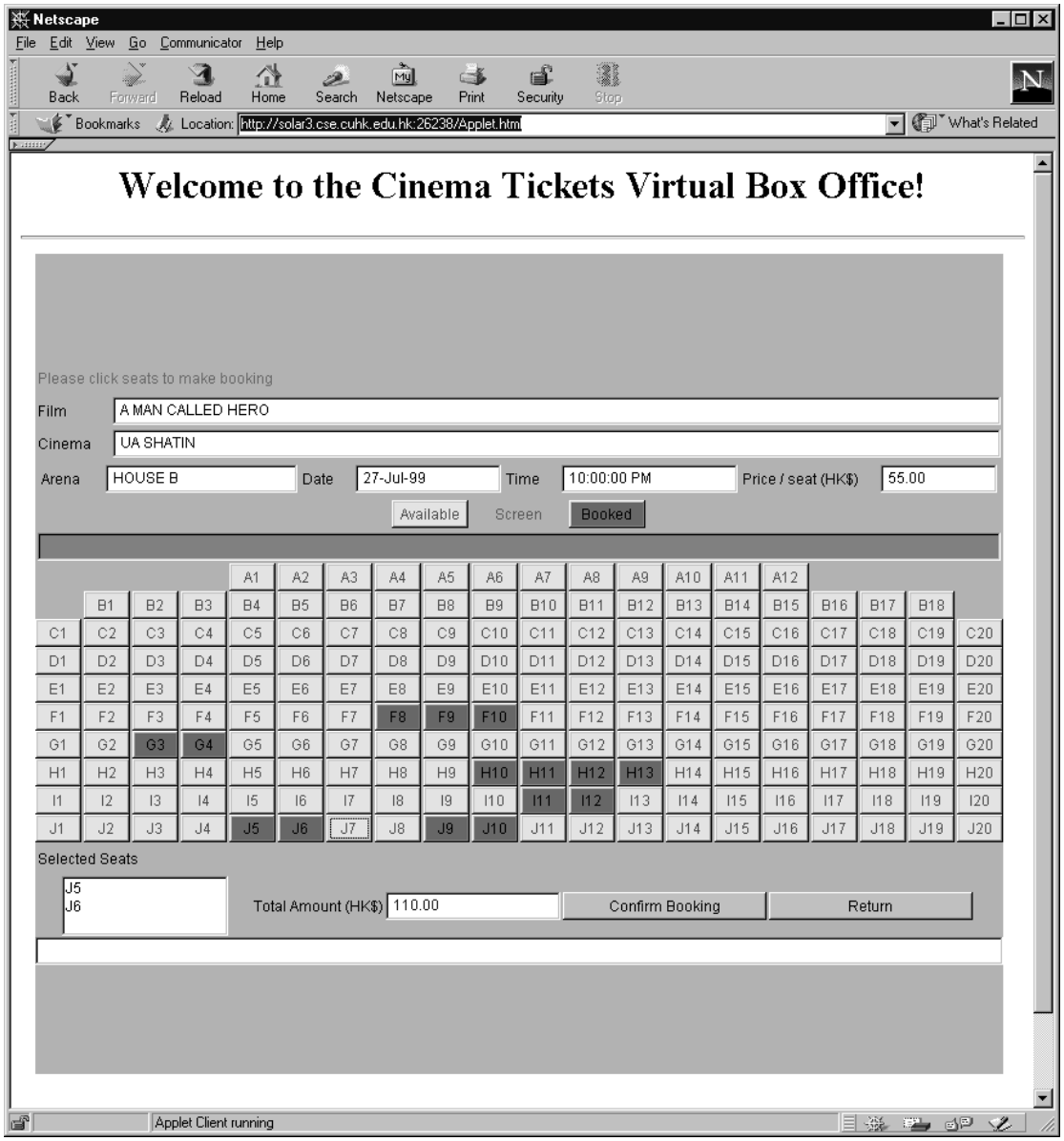


Figure 7.5

Test Cases	Conditions	Result
10	Two users select same seats one	The first user who press confirm

	after another	button success while the second one fail. Correct booking information displayed for the first user.

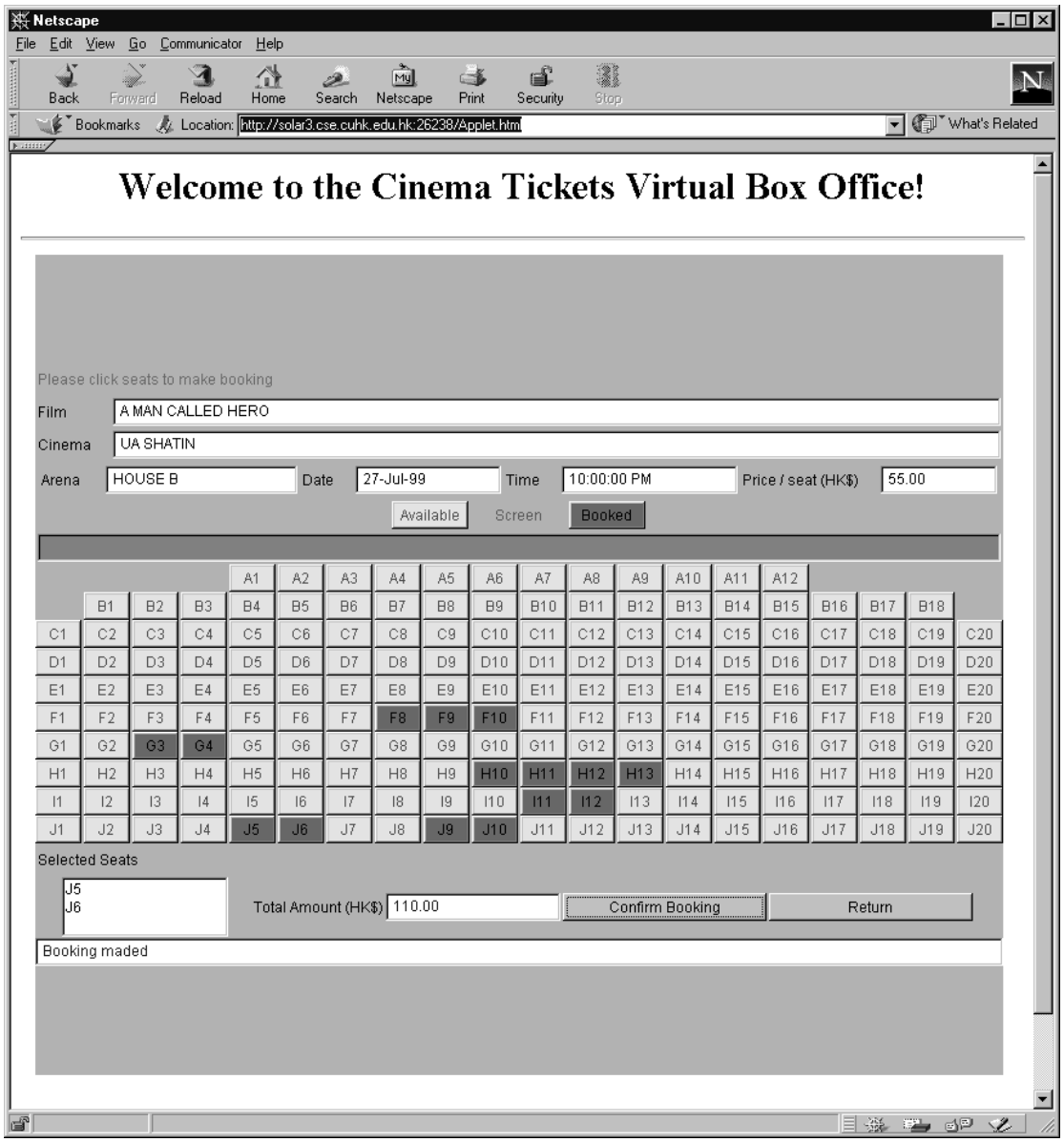


Figure 7.6

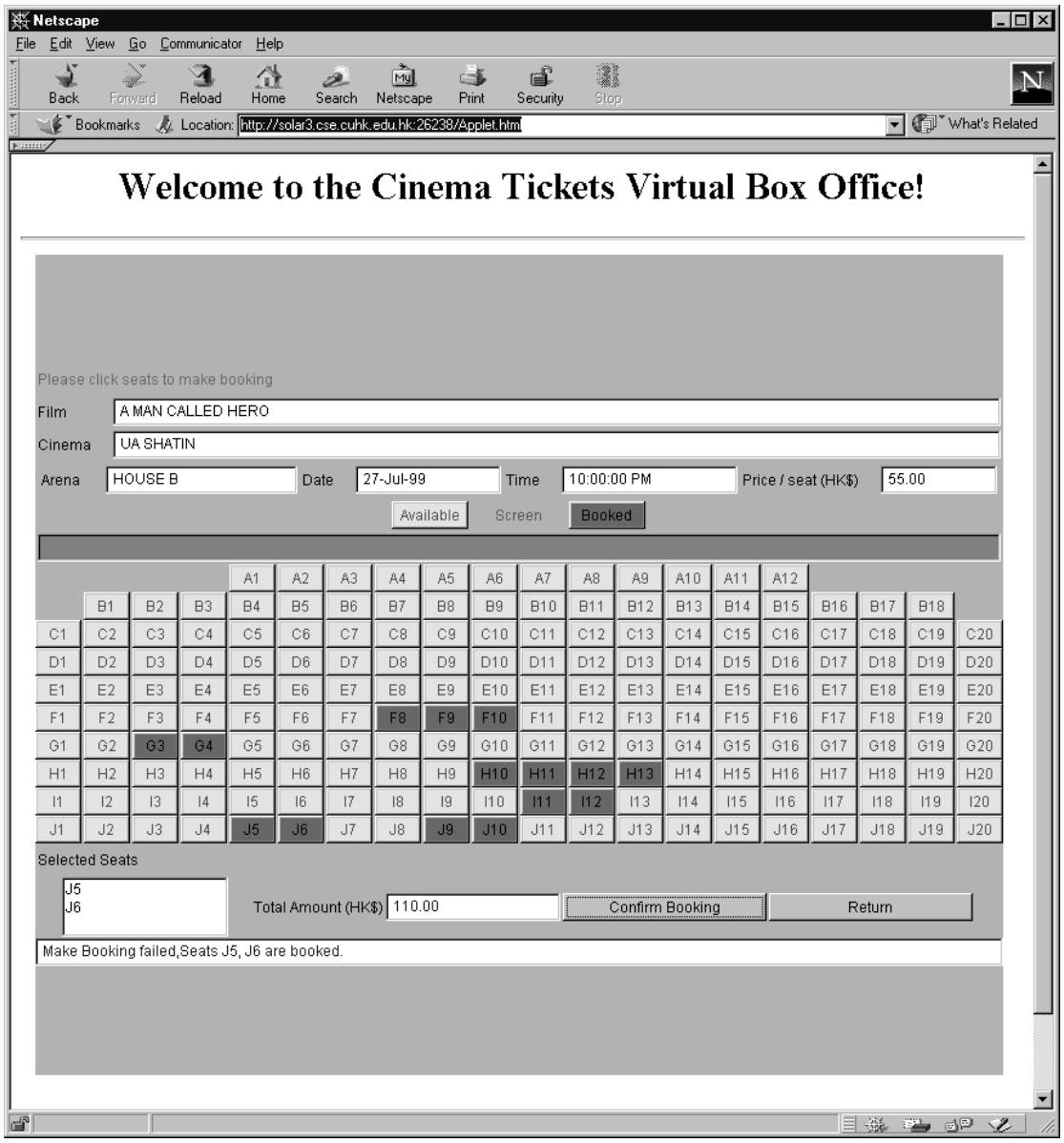

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://solar3.cse.cuhk.edu.hk:26238/Applet.html
What's Related
Welcome to the Cinema Tickets Virtual Box Office!
Please click seats to make booking
Film A MAN CALLED HERO
Cinema UA SHATIN
Arena HOUSE B
Date 27-Jul-99
Time 10:00:00 PM
Price / seat (HK$) 55.00
Available
Screen
Booked
A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12
B1 B2 B3 B4 B5 B6 B7 B8 B9 B10 B11 B12 B13 B14 B15 B16 B17 B18
C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11 C12 C13 C14 C15 C16 C17 C18 C19 C20
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12 E13 E14 E15 E16 E17 E18 E19 E20
F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12 F13 F14 F15 F16 F17 F18 F19 F20
G1 G2 G3 G4 G5 G6 G7 G8 G9 G10 G11 G12 G13 G14 G15 G16 G17 G18 G19 G20
H1 H2 H3 H4 H5 H6 H7 H8 H9 H10 H11 H12 H13 H14 H15 H16 H17 H18 H19 H20
I1 I2 I3 I4 I5 I6 I7 I8 I9 I10 I11 I12 I13 I14 I15 I16 I17 I18 I19 I20
J1 J2 J3 J4 J5 J6 J7 J8 J9 J10 J11 J12 J13 J14 J15 J16 J17 J18 J19 J20
Selected Seats
J5
J6
Total Amount (HK$) 110.00
Confirm Booking
Return
Make Booking failed,Seats J5, J6 are booked.

Figure 7.7

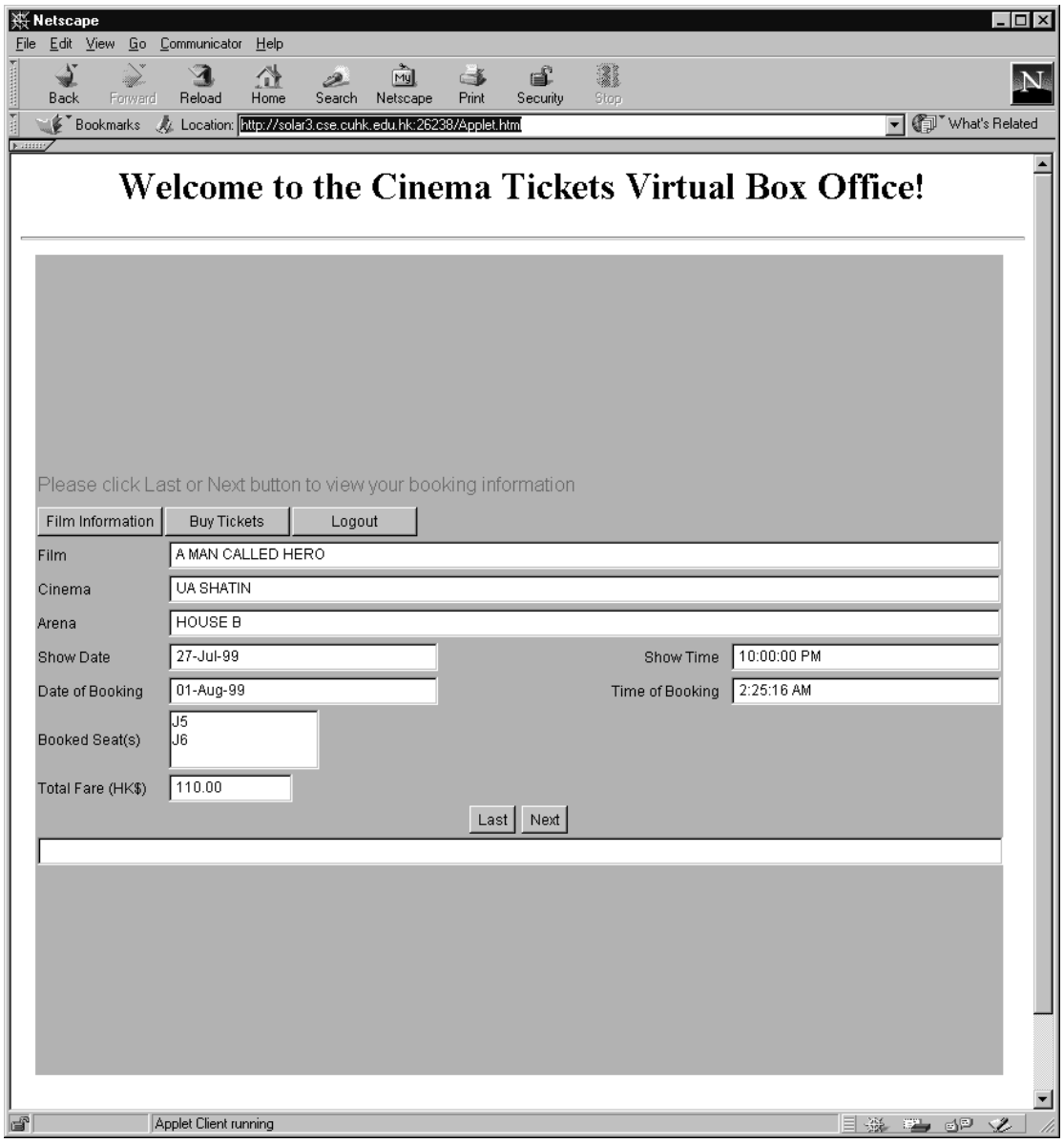


Figure 7.8

Test Cases	Conditions	Result
11	Cinema managers create a new film	New film available for scheduling

Test Cases	Conditions	Result
12	Scheduling for a film	New schedule available for users to select

Chapter 8

Discussion

This section will discuss some pros and cons of CORBA and Java.

- **A solid distributed object foundation:** A CORBA object reference is a very powerful unit of distributed service negotiation. It points to an object interface – that is, a set of related methods that operate on an individual object. In contrast, an RPC only returns a reference to a single method.

- **Callbacks:** we were able to usr CORBA callbacks very effectively to control clients from server side. You can also use callbacks to create client applications (and applets) that dynamically receive content, state, news, status, alerts, and instructions from their servers.

- **Excellent CORBA/Java Integration:** CORBA interfaces map nicely to their Java counterparts.

- **A modern 3-tier client/server foundation:** CORBA objects make ideal server objects in a 3-tier (or n-tier) distributed architecture. They provide a middle-tier object-to-object infrastructure that you can use to encapsulate data from multiple sources. In addition, you can use CORBA IDL encapsulate existing systems and connect them to the ORB.

- **CORBA works just fine from within applets:** Applets make wonderful

downloadable clients. A Java applet can interact directly with CORBA server objects. In addition, CORBA server objects can call back the applet to update its state.

- **You can create multi-panel applets using AWT:** the clients created in this project shows that you can create very functional stacks of business forms using AWT's card layout mechanism. In addition, you can give these panels a professional look using the grid bag layout mechanism. Our applet was able to navigate the panels and invoke CORBA methods to populate the fields.

- **A portable operating system for servers:** The Java language offers many operating system features. It lets you write very portable server objects. For example, Java provides automatic garbage collection and powerful error-handling facilities. With Java, you do not have to re-target your code to obtain facilities on a variety of server platforms.

Despite CORBA is quite powerful, there are still some questions on it.

- **Where's the load balancing?** Is there a ORB that can distribute server loads across multiple processors and provide a single system to the clients. The ORB should be able to pre-start server object and cache their state.

- **Where's the fault-tolerance?** Is there a ORB that provides automatic switchover during failure to an object replica.

Chapter 9

Conclusion

Nowadays, more and more systems are developed in a distributed fashion. In the expansion of a distributed system, expandability and interoperability between software are certainly the key issues. CORBA provides sound solution to many areas of distributed system, so in the long run, CORBA may play an important role in the distributed software industry.

In this report, I have examined the CORBA and how it facilitates the construction of modern distributed systems. I have developed a sample cinema tickets reservation 3-tier system using CORBA and Java for demonstration. I have used some time to learn how to adopt the CORBA into a distributed system. As CORBA is quite powerful in developing distributed system, so I believe it worth to take some time on learning it.

Although most of the desired functions are provided in the system, there are some areas of improvement to the system. 1) The server can use the callback facility to periodically update the seats' status to the clients. This can improve the accuracy and efficiency of tickets selling as the server actively notifies the clients about the seats' status. 2) The server can be started on demand. With this improvement the server need not to be running all the time to wait for the request of clients. On the other hand, if there are requests come from the clients, the server will wake up to serve the requests.

Appendix

User Guide

- Managers can login the system to register films and schedule show times. Managers just need to input correct manager ID, password and click button GO to login. If there is a typing mistake, managers can press the Clear button to clear the input fields.

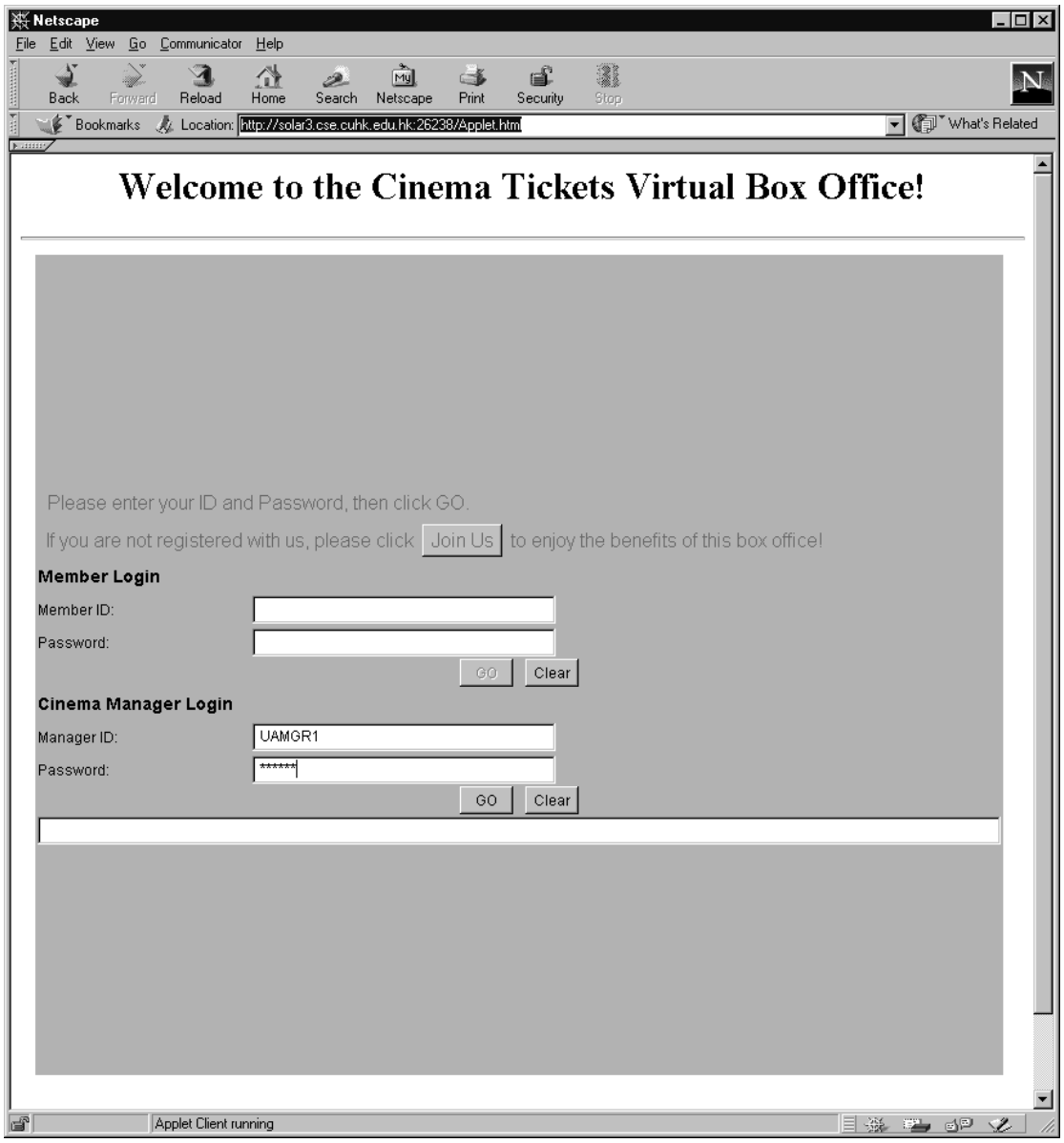


Figure A.1

- Managers can know more about a film before creating the schedule by selecting a film to view its information, comment and overall grade. Managers can select a film from the choice besides the *select a film* label.

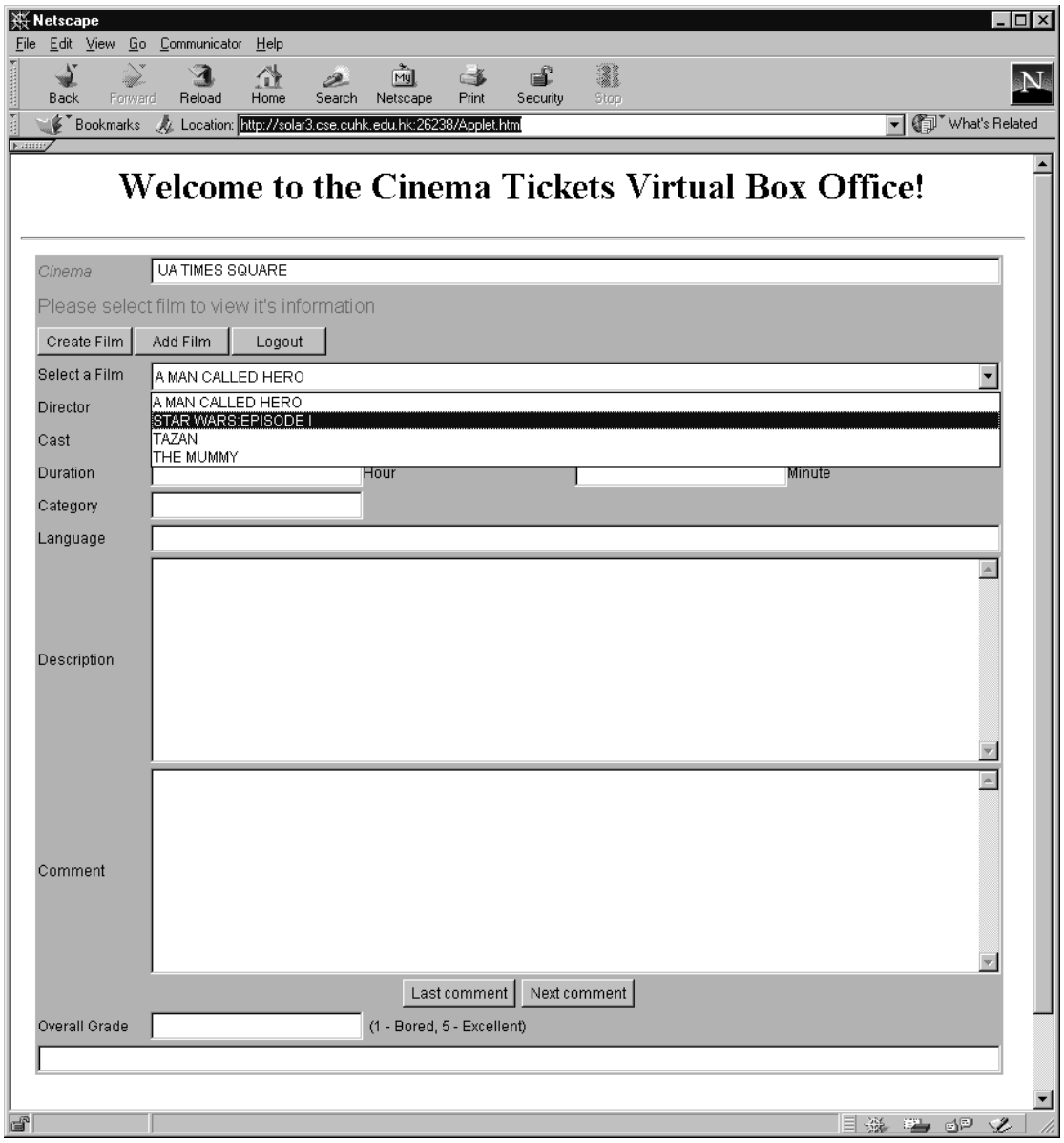

Netscape
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://solar3.cse.cuhk.edu.hk:26238/Applet.html What's Related
Welcome to the Cinema Tickets Virtual Box Office!
Cinema
UA TIMES SQUARE
Please select film to view it's information
Create Film
Add Film
Logout
Select a Film
A MAN CALLED HERO
Director
A MAN CALLED HERO
STAR WARS:EPISODE I
TAZAN
THE MUMMY
Cast
Duration
Hour
Minute
Category
Language
Description
Comment
Last comment
Next comment
Overall Grade
(1 - Bored, 5 - Excellent)

Figure A.2

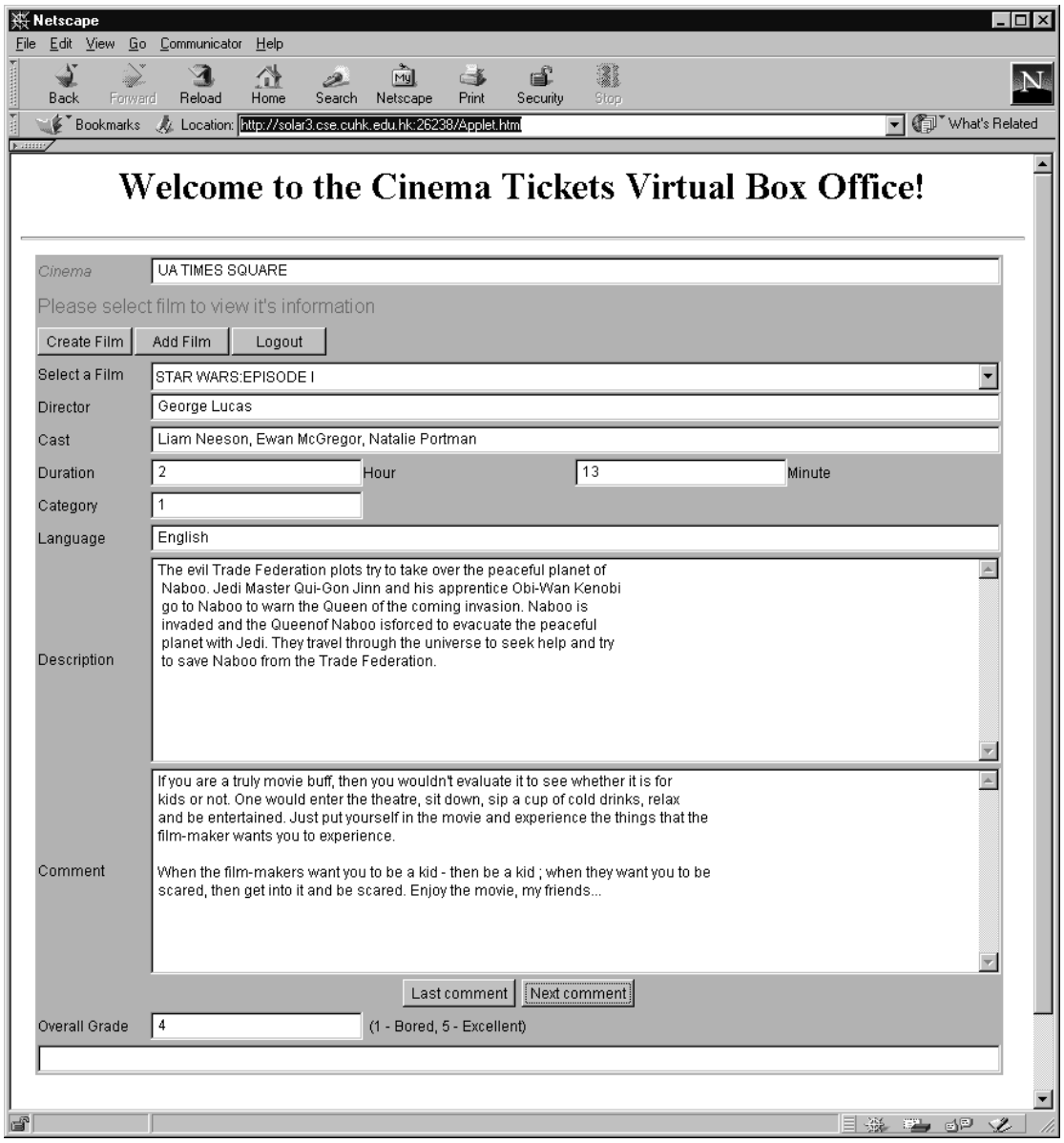


Figure A.3

- If managers cannot find a film to schedule for show, that film need to be registered first. To create a new film, managers can press the Create Film button. A screen like figure A.4 will show up. Managers can fill in a film's information and press OK button to complete the creation. Managers can cancel the creation by pressing the Cancel button.

Figure A.4

- Managers can make schedule for film shows through screen A.5. First, managers can select a film and a showing arena. Secondly, managers can input the start and end available booking dates of the film. Thirdly, managers can input the times of each day one row after another. The maximum time slot for each day is nine. Finally, managers need to input a price/seat and press OK to complete. Managers can cancel the scheduling by pressing Cancel button.

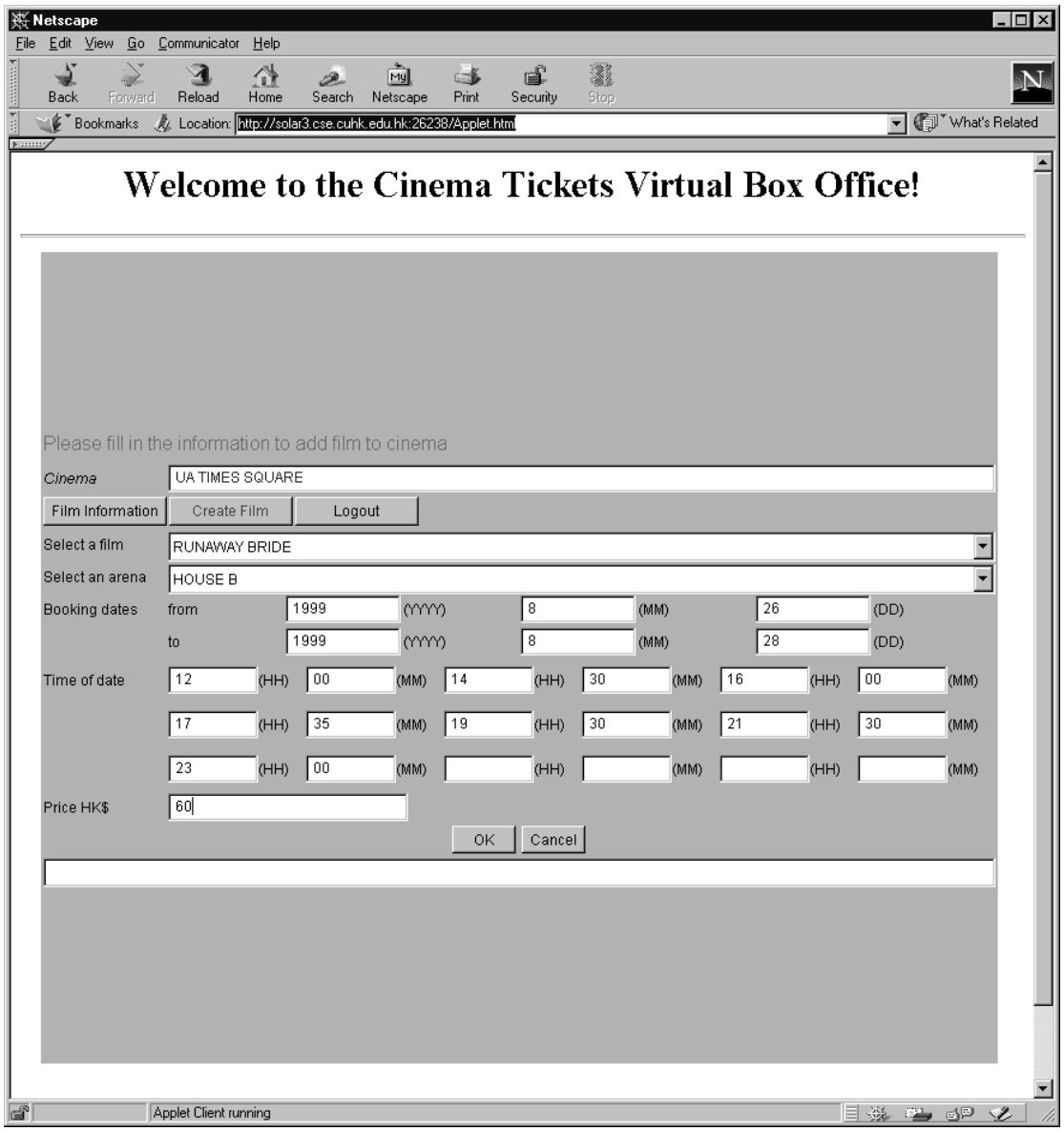


Figure A.5

- When managers has completed the films creation and scheduling, they can logout by pressing the logout button.

- Users can login if they have registered to the system by inputting the user ID, password and click the GO button. New users can register to the system by pressing the Join Us button.

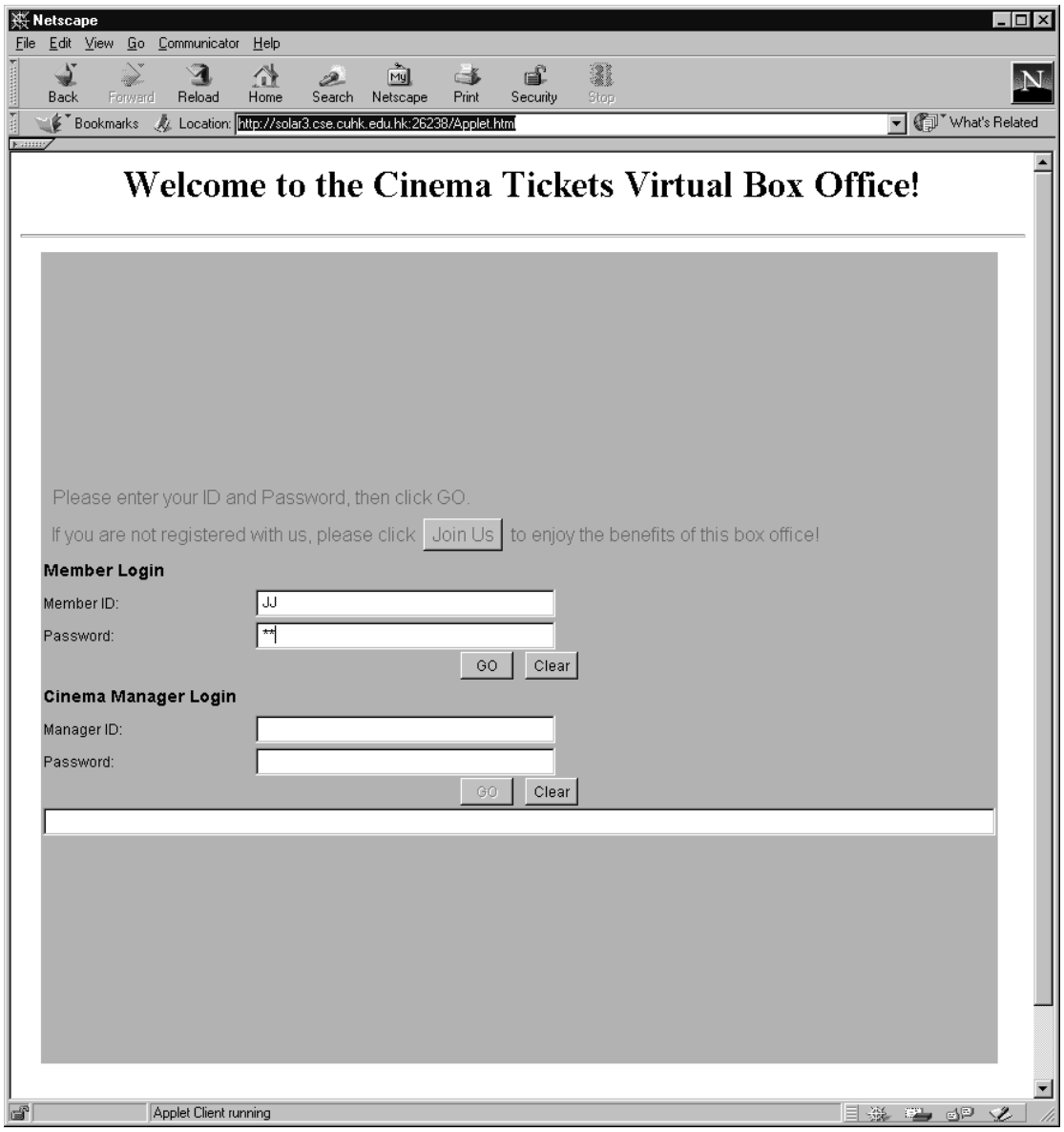


Figure A.6

- To complete the registration, new user need to fill in all fields in screen A.7 and press the OK button. To cancel the registration, users can press the Cancel button.

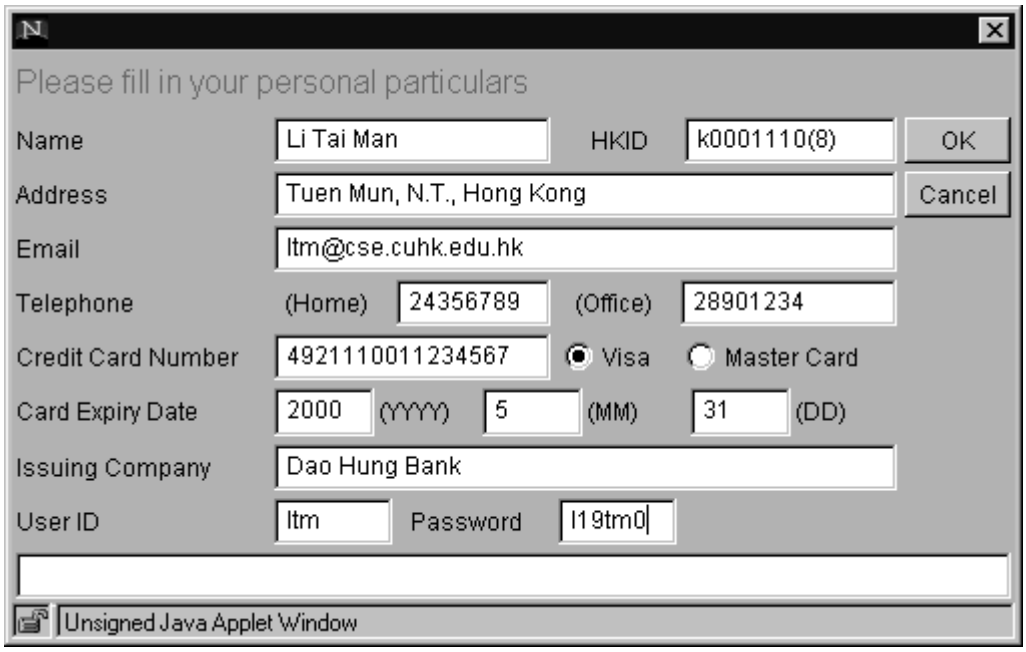


Figure A.7

- Users can select a film to view its information, comment and overall grade from the choice besides the *Select a Film* label.

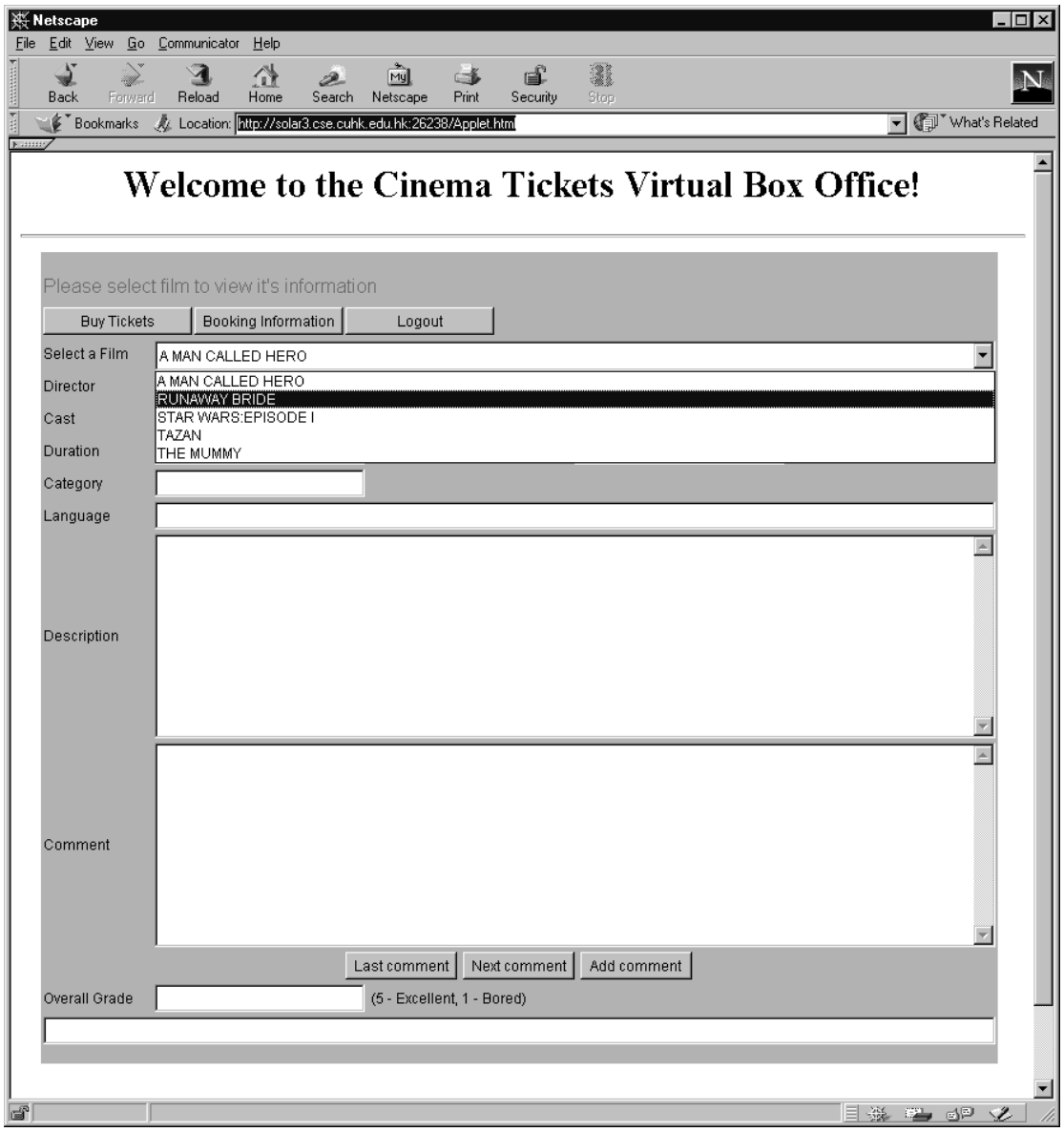


Figure A.8

- Users can view all comments of a film by pressing the last/Next comment button.

  Users also can add comment to a film by pressing the Add comment button.

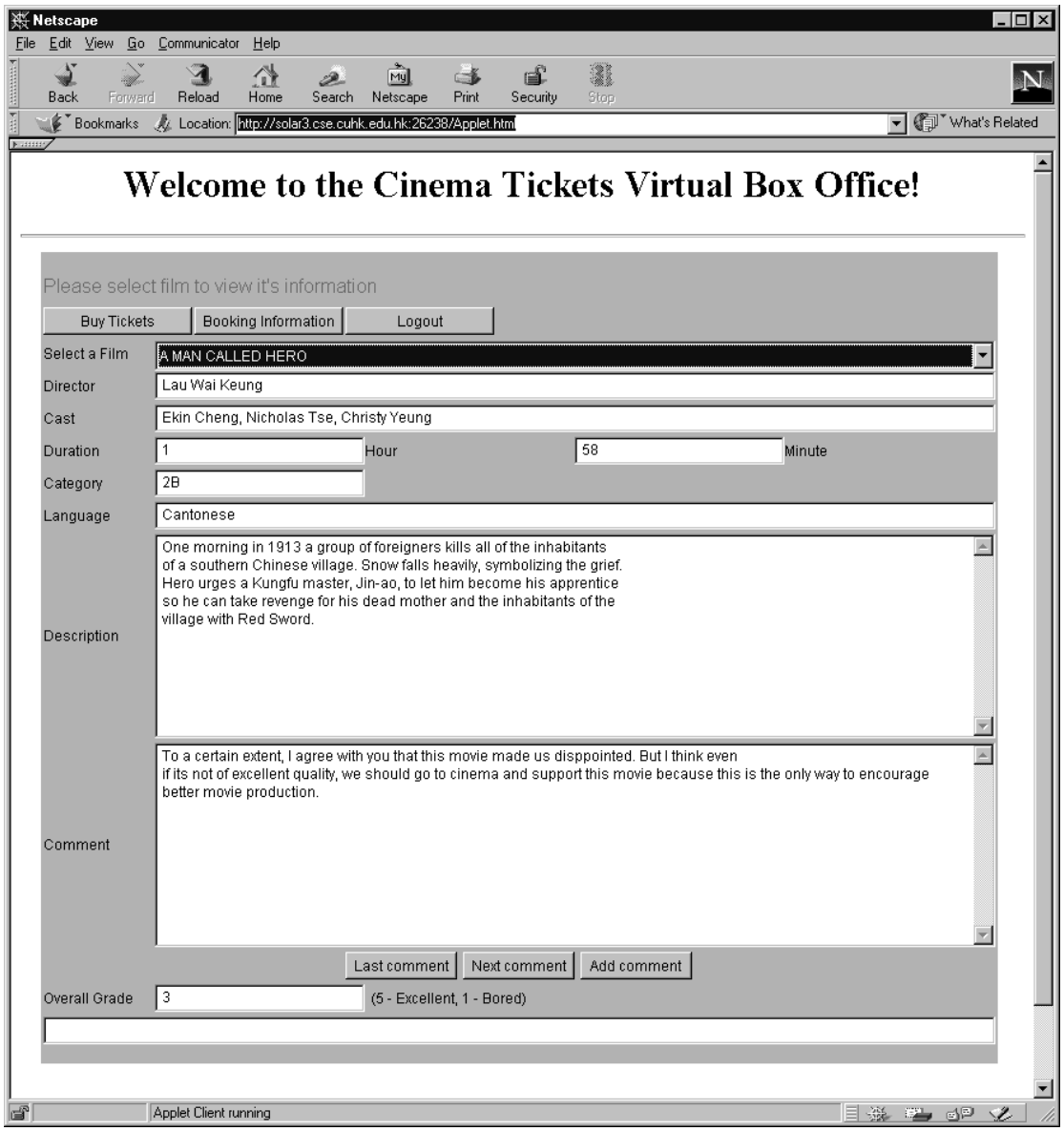


Figure A.9

- To add comments, users just need to fill in the comments, vote a grade for the film and press the OK button. Users can cancel the action by pressing the Cancel button.

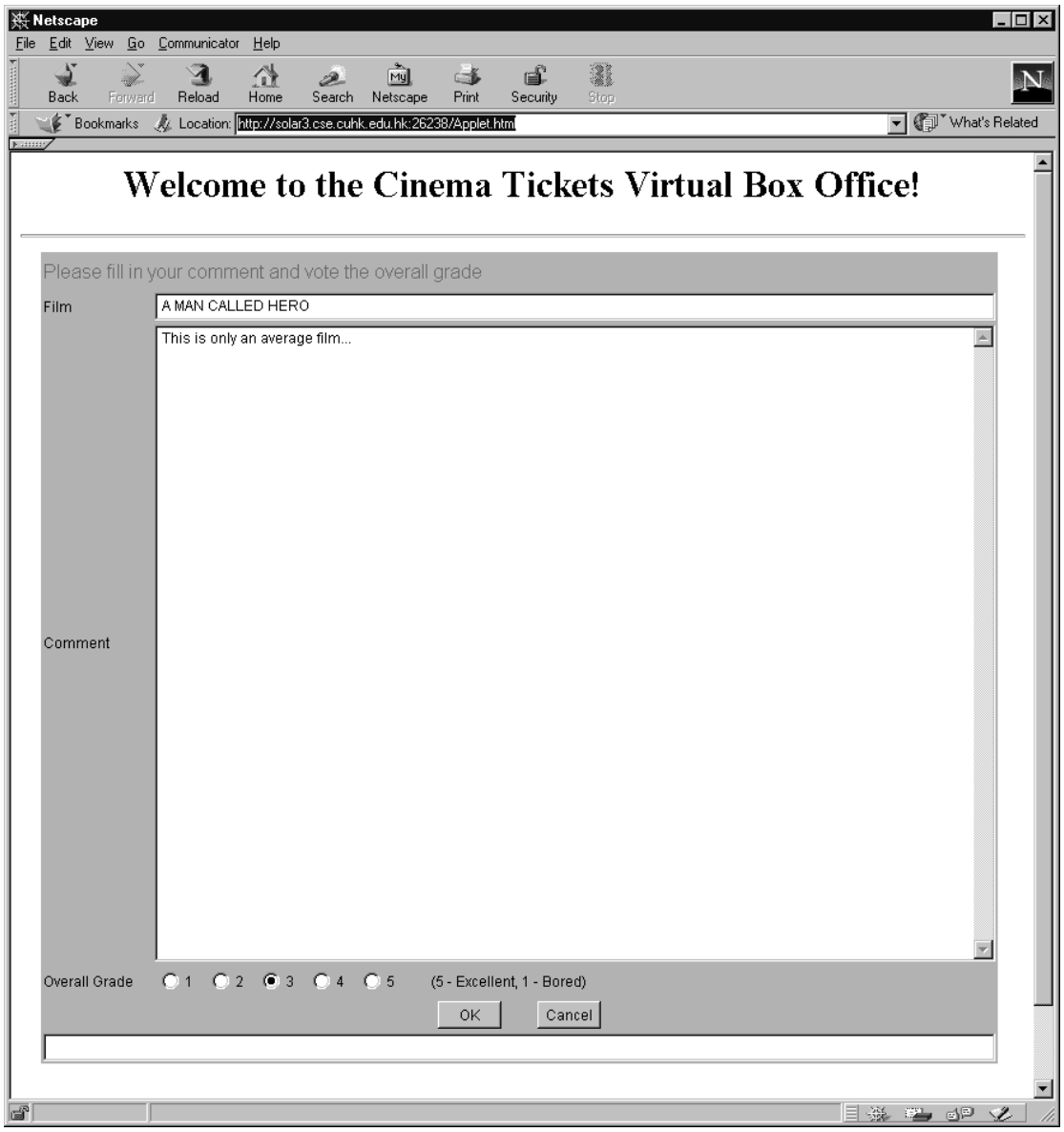


Figure A.10

- If users want to buy tickets, they can click the *Buy Tickets* button. A new screen such as figure A.11 will show up. Users should select the film first, then they can select the cinema and arena. Afterwards, they can select the show date and show time. When all choices are selected, they can press the *Seating Plan* button to go to the seating plan.

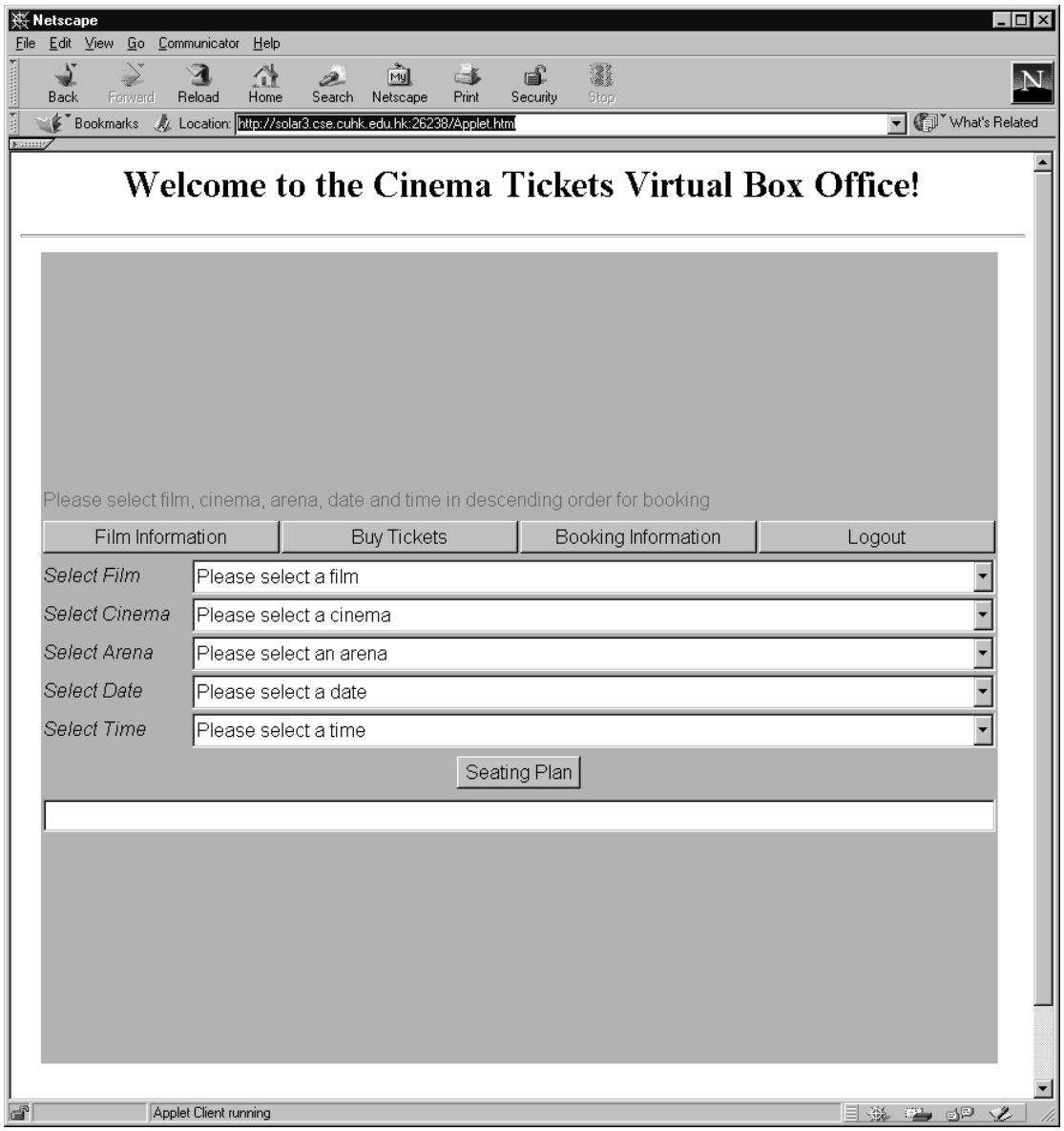


Figure A.11

- In the seating plan, users can click on a seat to select it or click on a selected seat to deselect it. If seats are selected, their color will change to deep blue, their labels will be added to the list named *Selected Seats*, the total amount will be adjusted accordingly. When the desired seats are selected, users can press the Confirm button to complete the tickets buying procedure. Users can cancel the tickets buying procedure by pressing the *Return* button.

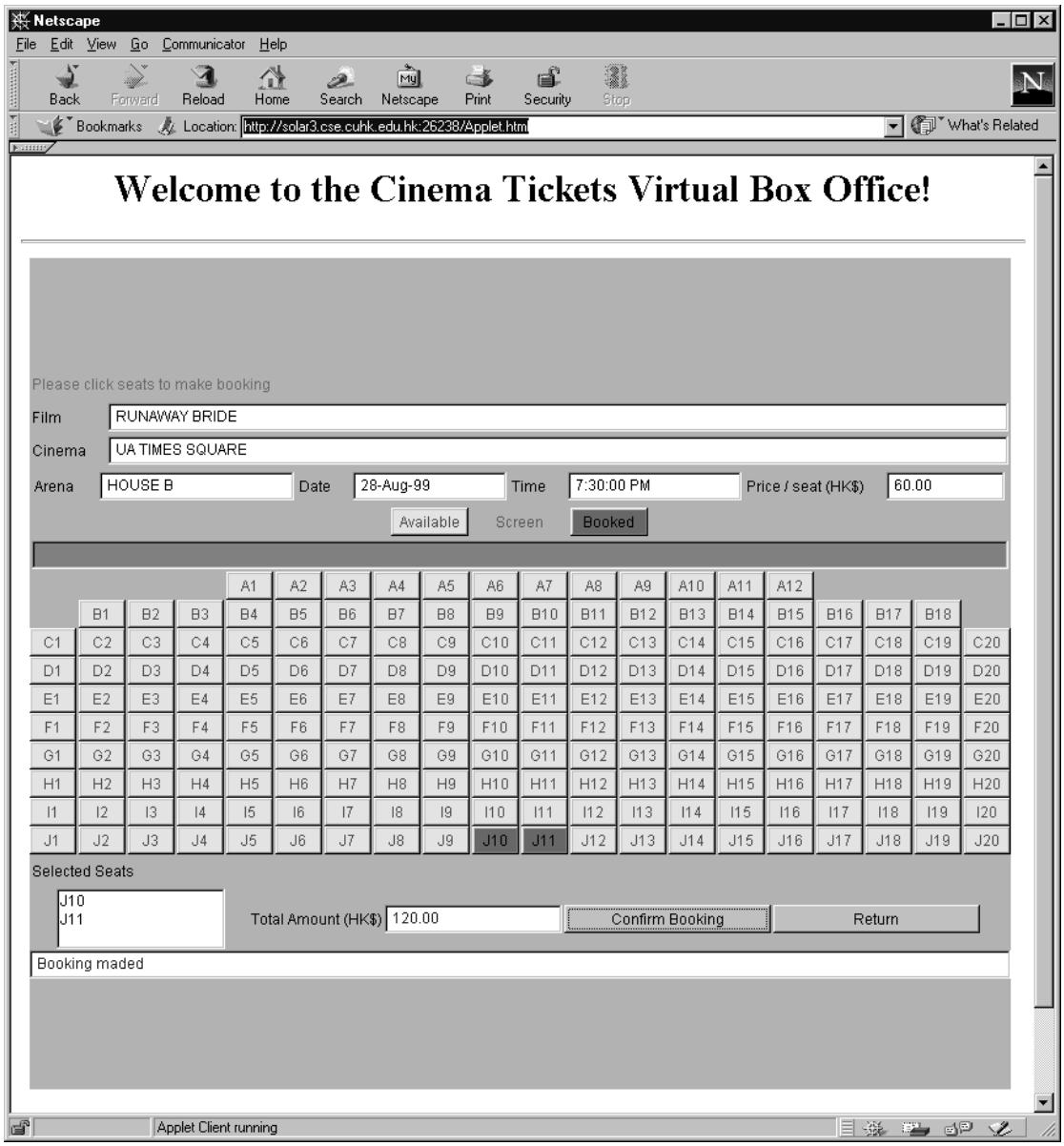


Figure A.12

- After users have bought tickets, they can view the information of bookings made by them by pressing the *Booking Information* button. The detail booking information is shown as in figure A.13.

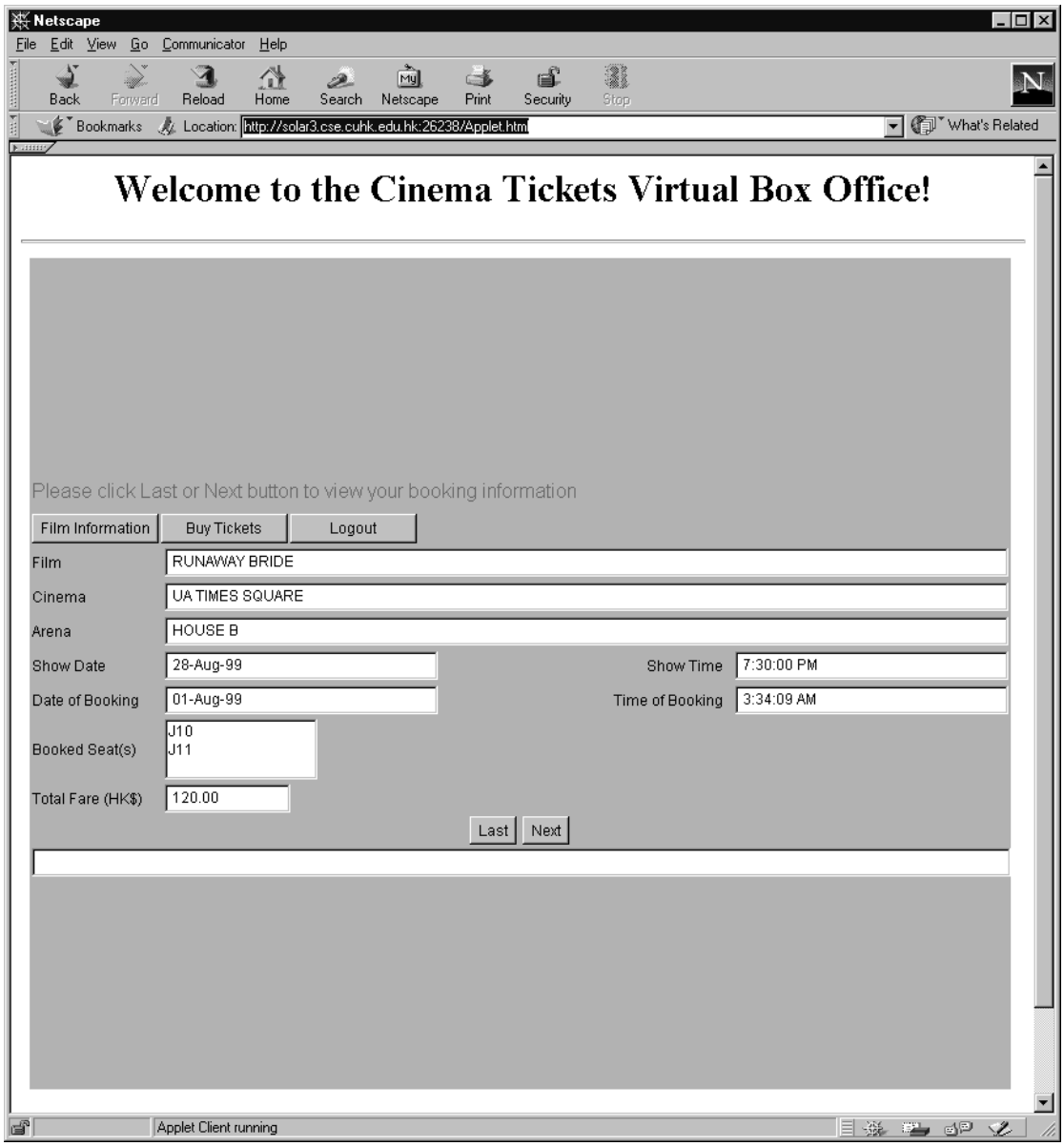


Figure A.13

- Users can logout the system by pressing the *logout* button at any screen.