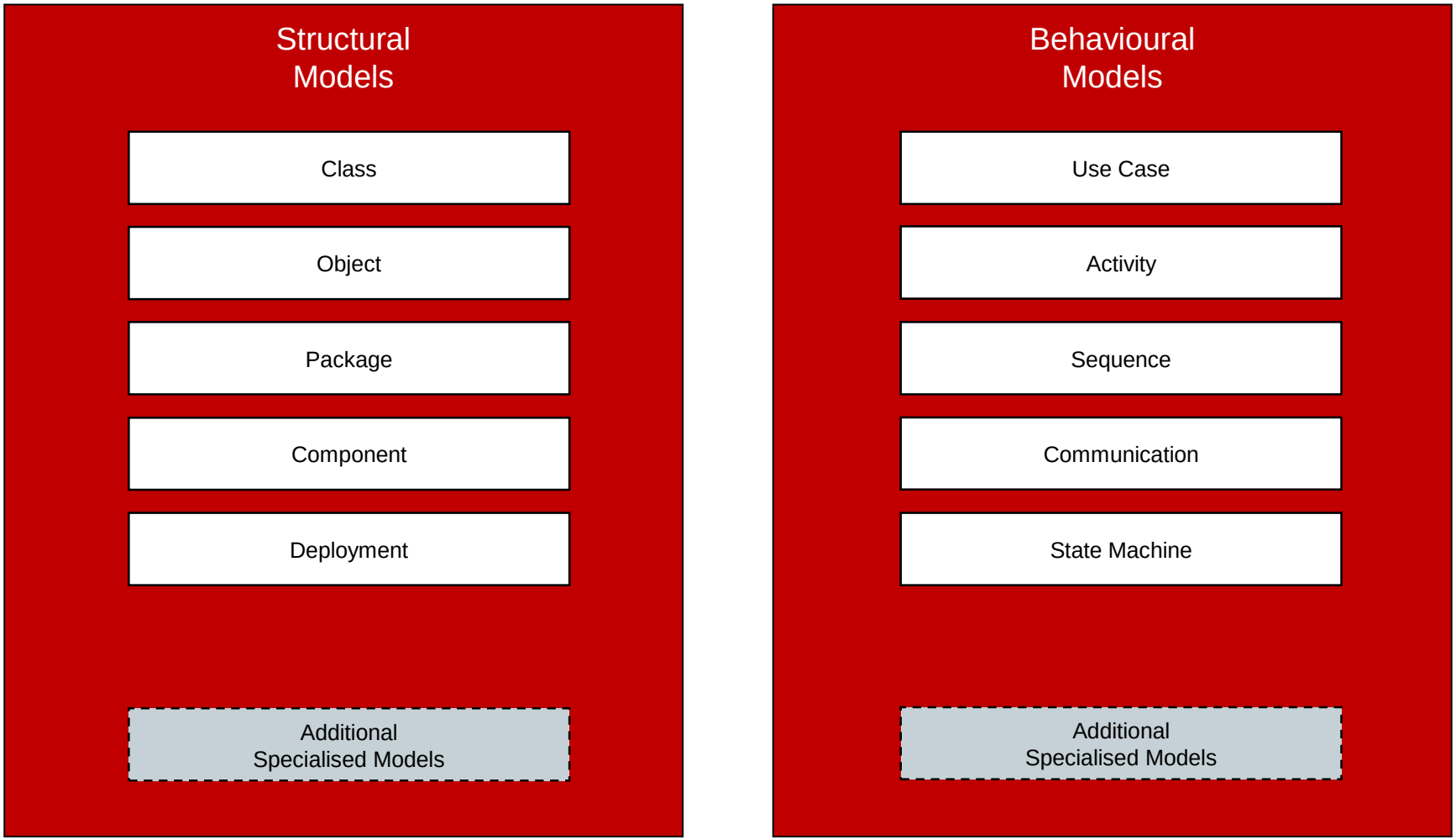
Introduction To The Unified Modelling Language (UML)



Course Context

UML is the preferred notation for modelling software rich business solutions.

This introduction provides a short overview of UML and introduces you to the basic models.

Note that for all of these models there are additional elements that reflect more complex concepts. These have not been included in this course.

The introduction provides:

- An understanding of UML and its model types that enables you to read and contribute to relevant models.
- Provides and certifies a minimum standard of knowledge about modelling software rich systems for all of our architects and designers.

Two specific areas of the UML have not been mentioned as they reflect more detailed aspects.

- Object Constraint Language: OCL supplements UML by providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics. OCL is also a navigation language for graph-based models.
- Model Driven Architecture: MDA is an approach to development that directly connects models with their implementation in software. Essentially the model becomes the specification for software construction, which is then applied in specific deployment concepts by tools specialised for each implementation environment.

If you are going to be undertaking significant amounts of modelling and use our modelling tools (currently Rational Software Architect) extensively then you should also look for more complete training, offered through classroom and interactive on-line courses.

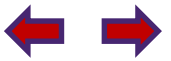
For more details on UML in general please follow this link

For more details on OCL please follow this link

For more details on MDA please follow this link

1

Why use UML



Why modelling?

The Unified Modelling Language (UML) is the preferred notation for BT for modelling software rich business solutions. As most of our business processes are supported by ICT systems it has an application for most areas of our business.

Modelling for business and software systems was initially developed through the 1970s-1990s as businesses began to use more systematic methods of improving operation and increasing automation of processes.

Modelling enabled a better understanding of existing processes and systems and was increasingly aligned to standard taxonomies creating shared concepts and terms.

When we look to create new business solutions we are faced with the tasks of understanding:

- The current state of our business environment
- The specific requirements for change
- How we translate those requirements into changes in our business environment

To help us in this we develop models. Models are abstractions of something for the purpose of understanding and generally represent specific points of view and simplifications of the real world.

They enable us to focus on the essential elements and properties of a desired solution by:

- Highlighting different viewpoints and views
- Identifying important structures and patterns
- Improving communication with various stakeholders
- Enabling early testing before full scale creation
- Reducing complexity for analysis and overall design
- Providing pre-defined concepts for optimising complex relationships when needed

Where does UML fit in?

Creating improved business solutions requires we understand elements such as:

- Processes, people and roles
- Information & data
- Software systems and computing/network devices

UML was initially developed to support object oriented analysis and design. This focused on improving the process of building object oriented software. As the OOA&D approach developed it has grown to take in process modelling, data modelling and physical systems / network modelling, and to combine these with an overall container framework of components, sub-systems, systems and packages.

UML is now used as a notation for the modelling of software rich business solutions.

Notation	Modelling Type	Target Artefact
ARIS / BPMN / **UML** Notation	Business Process Modelling	Process & Procedures
UML (Partial)	Software & Physical Service Modelling	Software & Physical Services
ER / DATA Modelling **UML**	Software Systems & Information Modelling	Software & Data
UML (Partial)	Hardware Device Modelling	Computing & Network Devices
UML (Partial)	Network Modelling	Network Connections

Specific uses of UML:

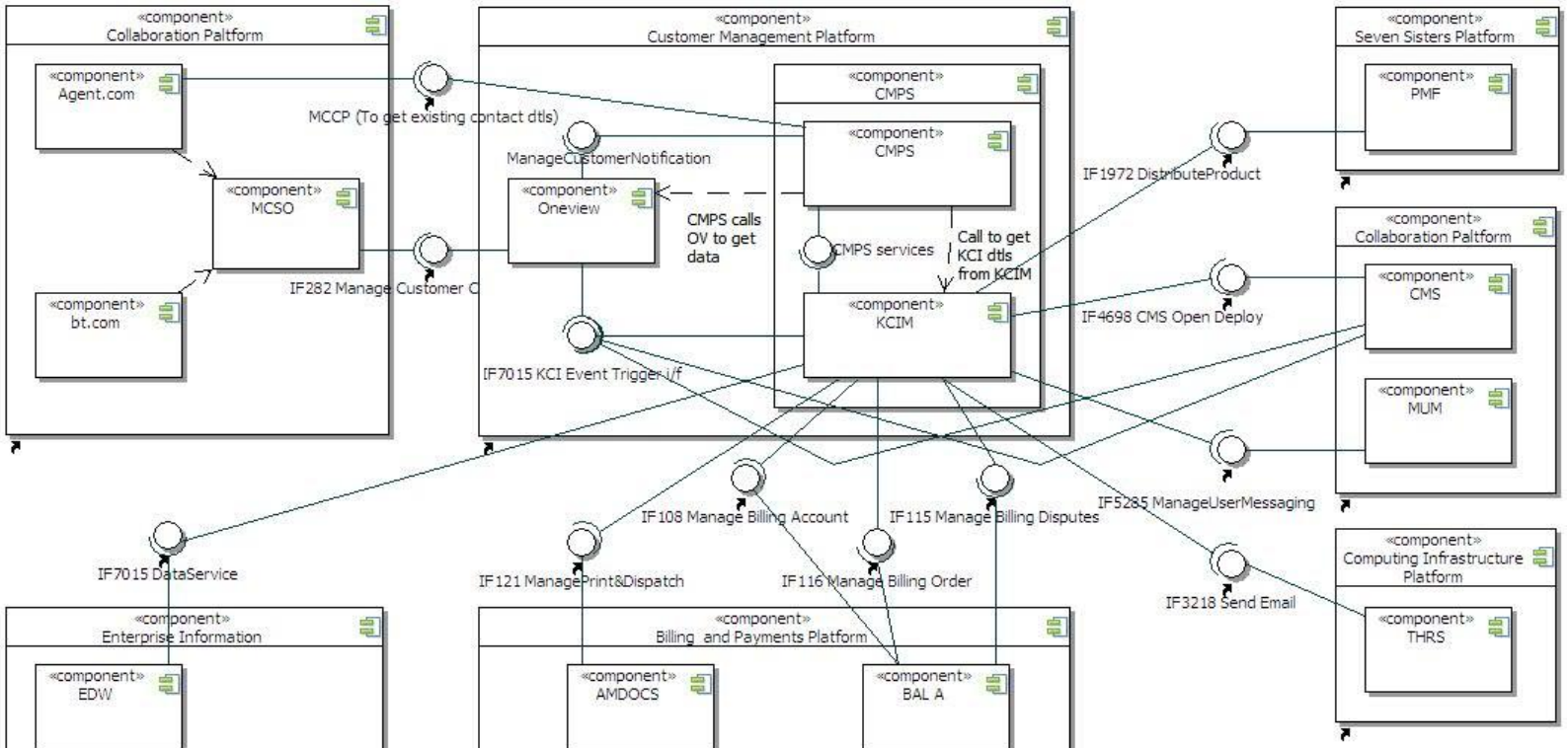
- To model the services offered across the boundaries of our platforms we can create the platform service / interface diagrams (UML package based component models)
- When developing models to support our software service development (all of the UML models)
- To reflect the deployment of software services on computing and network devices in networks (UML deployment models)
- As one of the notations for creating process models (UML activity diagrams) and information or data models (UML class and object models)

An example of UML for software platform interactions

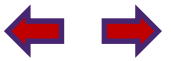
Below is a an example UML component model showing the platform component interfaces and flows involved in a change to the way we manage product templates and billing information.

This diagram is produced as part of our high level design process to identify the platforms and services involved in a design.

Later in the course you will see the standard notation for a UML component diagram that matches this.



2 What is UML?



The basic properties of UML

UML is a standardised visual modelling language intended to be used for modelling business process and their underlying elements and the analysis, design, and implementation of software based systems.

UML can be applied to diverse business domains (e.g., banking, finance, internet, aerospace, healthcare, telecommunications, etc.) and can be used with all major object and component software development methods and for various implementation platforms (e.g., J2EE, .NET).

UML was developed in the early 1990s to provide a consolidated standard industry modelling language (from the pre-existing object, function, process and data oriented modelling languages) and has since become the worldwide standard managed by the Object Management Group (OMG).

- UML version 1.1 adopted by OMG and updated to 1.3, 1.4, 1.5 with minor revisions and additions.
- UML version 2.0 was introduced in Oct 2004 to support Model-Driven Architecture/Model-Driven Development 2.4 is the version released in 2011.
- Version UML 2.x refers to any of the version 2 minor versions.

It is intentionally development process independent and can be applied in the context of different processes. However it has been developed in line with use case driven, iterative, agile and incremental development processes. An example of such a process is Unified Process (UP) and there are proprietary implementations of the UP such as the Rational Unified Process (RUP).

UML is developed by the Object Management, please follow this link for more details on the OMG and UML.

RUP is now owned and developed by IBM, please follow this link for more details on RUP.

The standard UML models

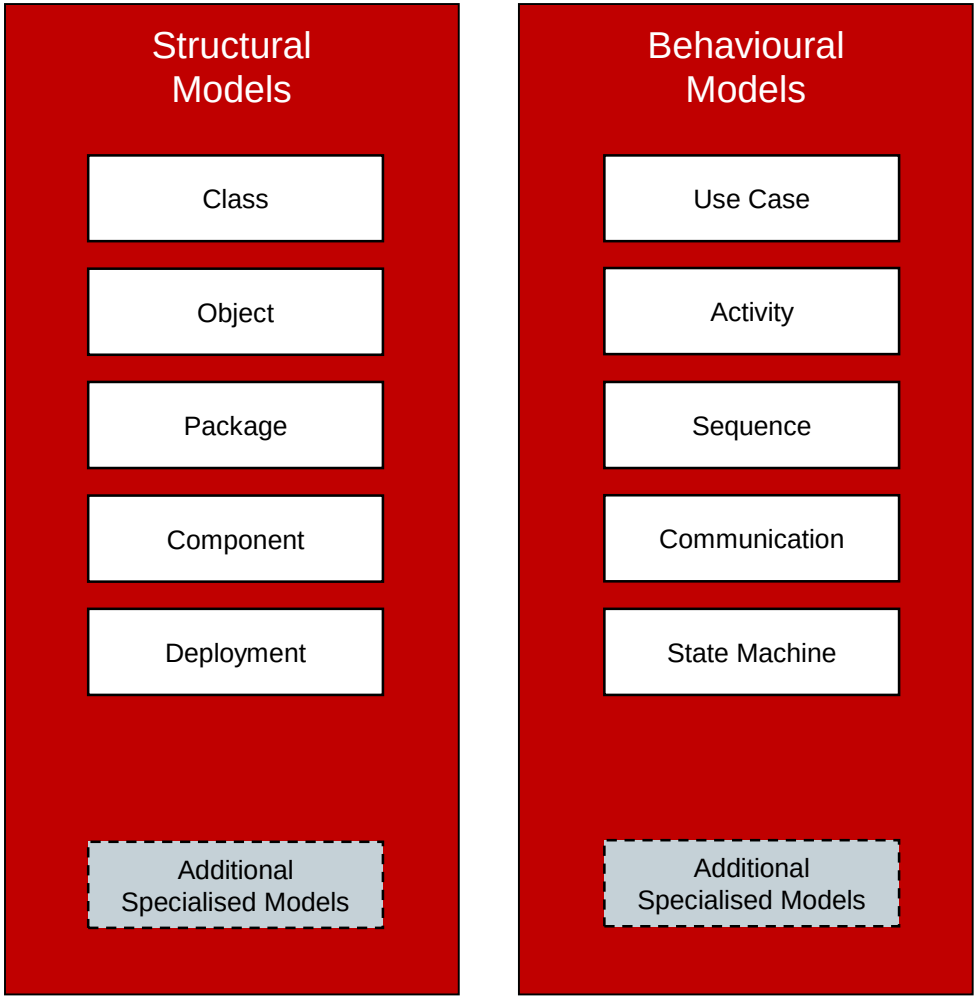
UML comprises a set of standard structural and behavioural models (shown on this slide) supplemented by some more specialised ones such as:

- Composite structure
- Interaction overview
- Network architecture
- Timing
- Locality

The structural models represent the ”things” in the problem space, how they are grouped, and their relationships.

The behavioural models represent actions/activities and flows in he problem space, how they are connected and how they are sequenced.

This course provides examples of the main models (but does not cover the more specialised ones).



When you use the models

The models can be used to capture three different levels of information:

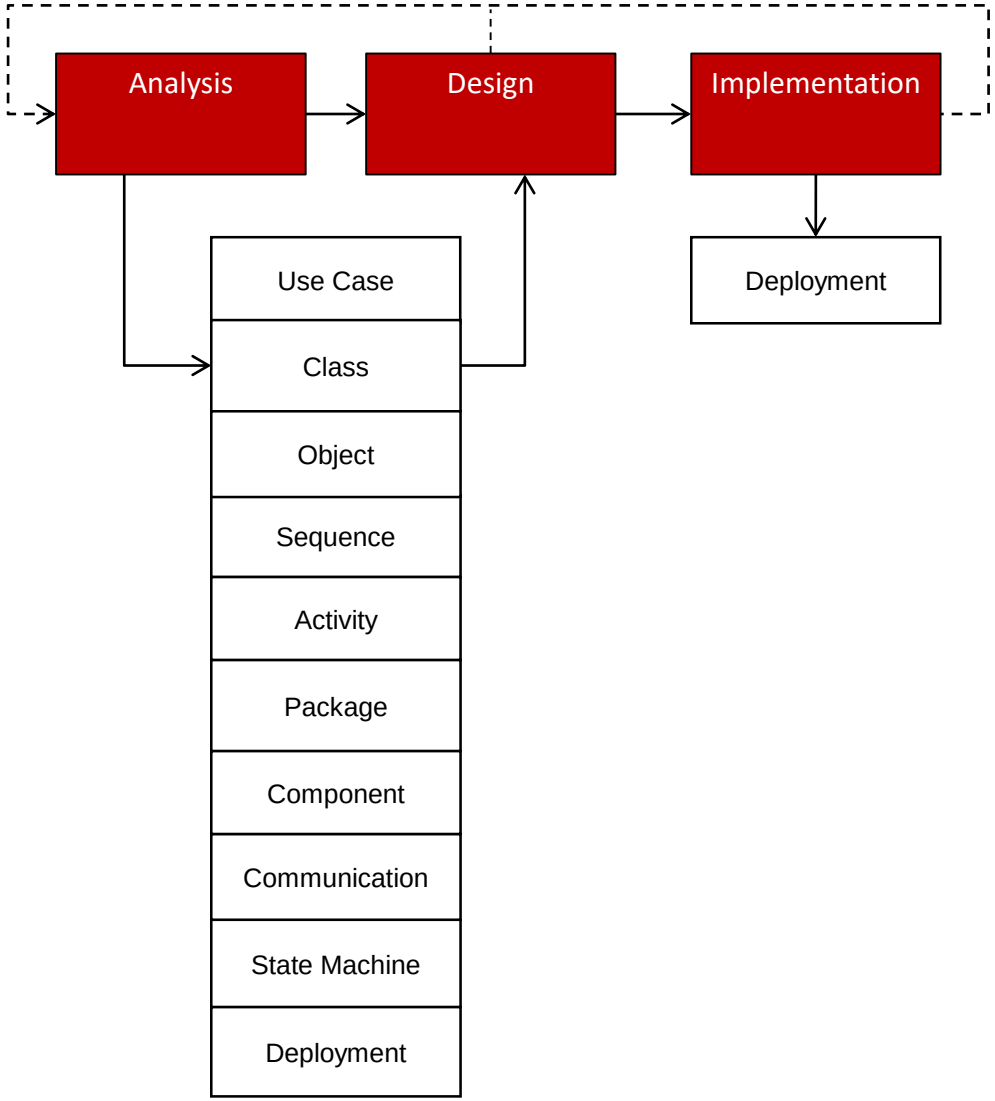
- Analysis (As Is and To Be)
- Design
- Implementation

All of the models can be used for analysing the current or proposed situation and capturing aspects of the design.

You should choose the models relevant to the specific solution being reviewed, changed or developed.

In a number of cases you will want to maintain both analysis and design models as on-going documentation.

You may also layer the analysis and design models to reflect different levels of detail such as an overall architecture view or a detailed design view.



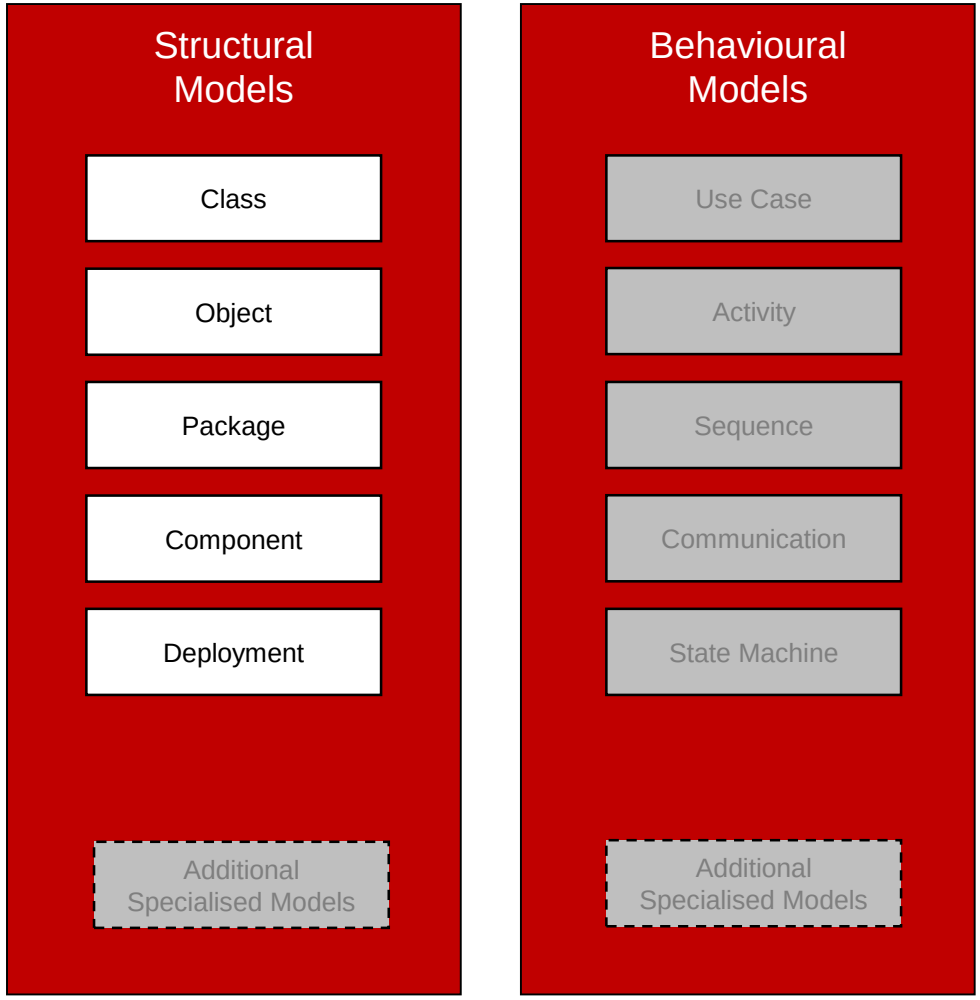
3 The structural models

The structural models

The structural models represent the “things” of concern in our business. They may be physical, software or various collections of these.

The structural models enable us define these structures and then organise then in the most effective ways.

The following slides provide examples of each of the models shown.

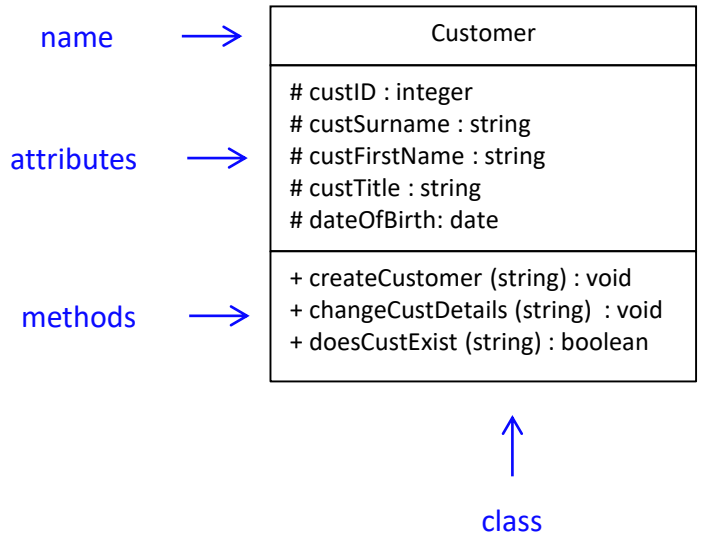


Class diagrams - the class

A class diagram enables us to identify and describe all of the entities involved in a particular system of concern.

- A class diagram shows a number of classes and the relationships between them.
- A class has attributes representing its state and methods representing its behaviour.

The single class shown on this page provides an example of the format for describing attributes and methods. An attribute has a name and a data type; a method has name, an optional supplied value, and an optional returned value.

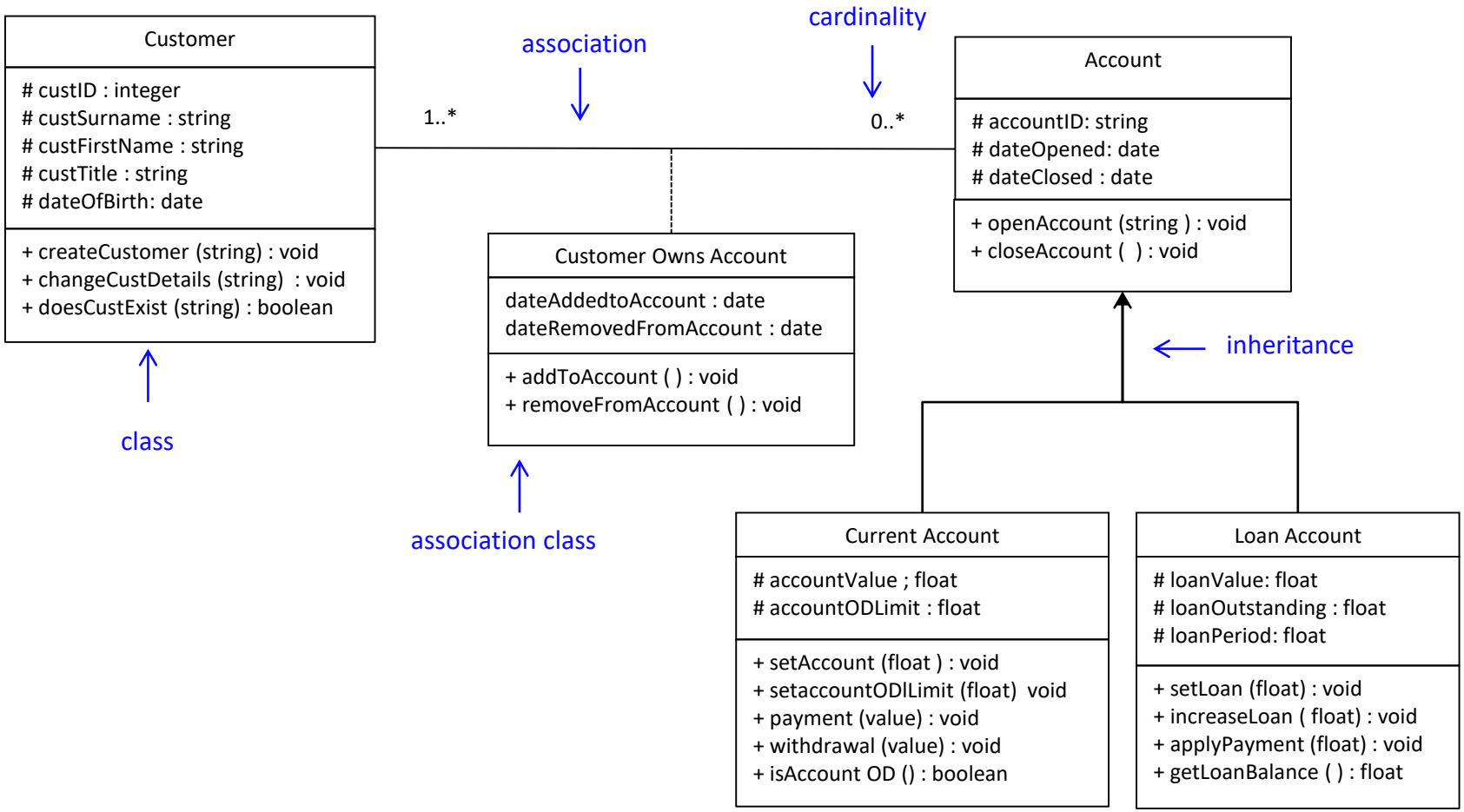


The basic notation for a complete class diagram is shown on the next page . It describes a number of classes and their association to each other.

An association is notated to show the nature of the relationship. An association can show a number of different details. The diagram shows the cardinality and it can additionally show directionality, constraints and aggregations when needed. If the association has attributes this creates an association class to hold those attributes as shown.

An inheritance association is one that reflects a specialisation/ generalisation relationship. The attributes and behaviour of a super class (the one at the head of the arrow) are inherited by the sub classes (the classes at the other end of the arrow) This means the sub classes take on the attributes and behaviours of the super class and can add on additional ones of their own.

Class diagrams - classes and associations



Object diagrams

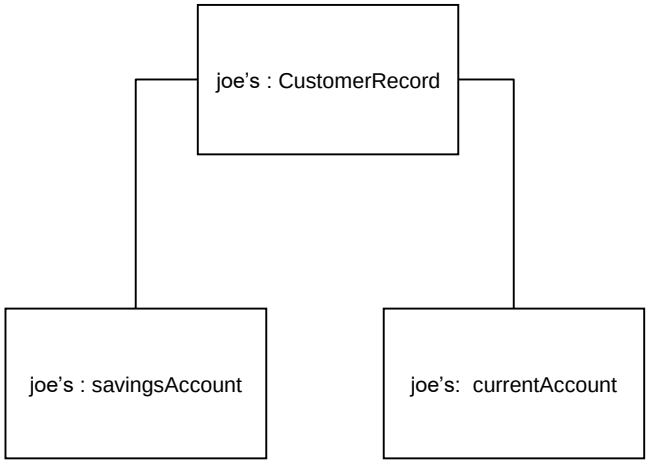
An object diagram is a special case of a class diagram.

It shows a single instance of a class and its specific relationships.

In the example we can see that an instance of a customer record (for Joe) is at this point in time connected to an instance of a savings account for Joe and current account for Joe.

The more general model of a customer record class and its connections to savings and loan accounts would only show that it is possible for Joe to have both accounts not that he actually does.

An object diagram enables you to create specific state scenarios to check that the more general class model with cope with all of the different real world instantiations it has to address.



Package diagrams

A package is used to organise and handle complexity in large models.

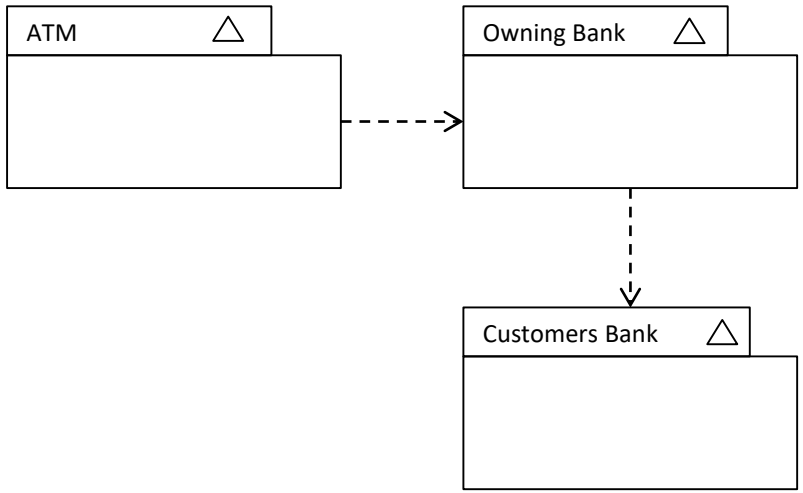
It is generally used to hold other UML models within a partition and provides the namespace for that partition.

Packages are represented as named folders as shown in the diagram and may be empty or optionally show its members.

Packages tend to be used to group use cases or classes.

There are different types of packages. The packages shown represent “model packages” and are denoted by the use of the

Other uses of packages are for more formal namespace management using specific △ associations representing; merge, import and nesting.

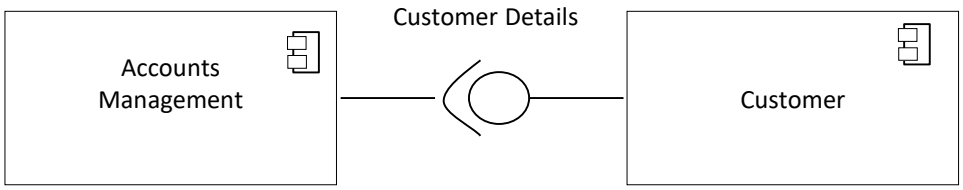


Component diagrams

Component diagrams are similar to package diagrams in that they represent specific namespaces of logical structures.

They display only the interfaces between components and hide the elements within the component.

The lollipop circle indicates that component provides the interface.
The open semi circle indicates that a component requires the interface.

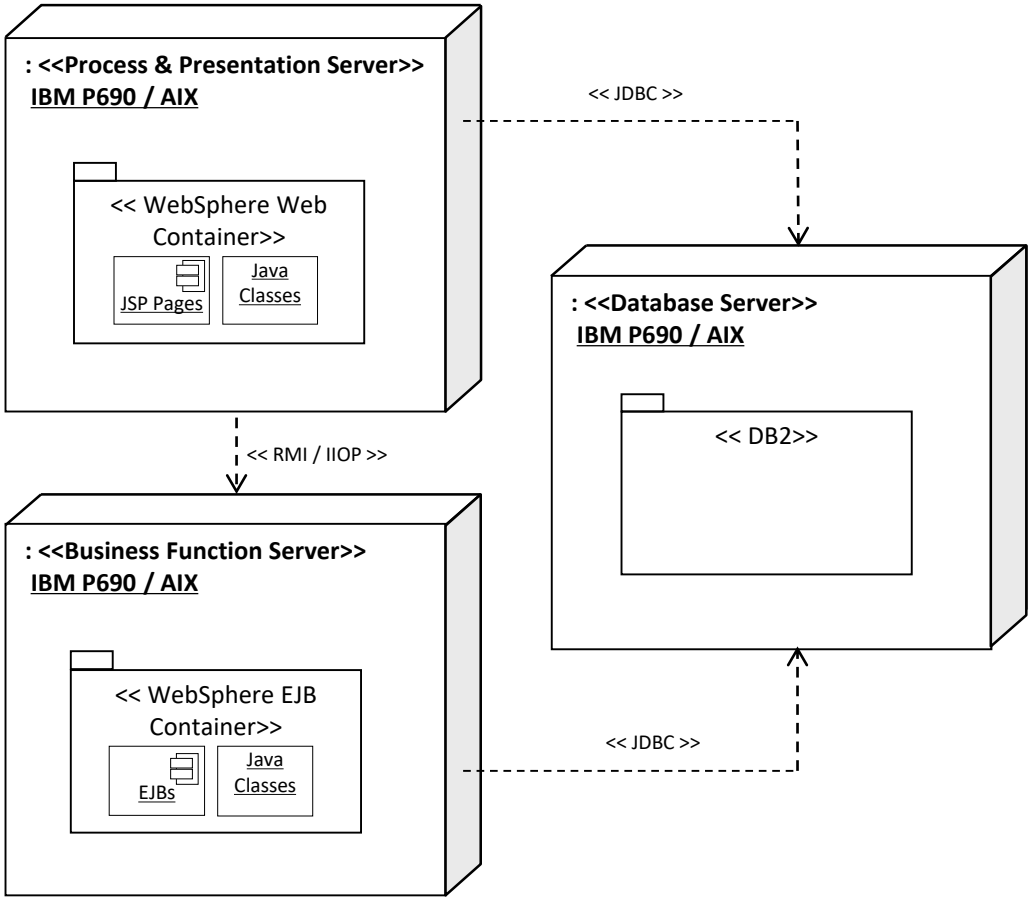


Deployment diagrams

Deployment diagrams show instances of artefacts placed onto specific target environments.

They are generally used to define software placement on specific computing and network devices.

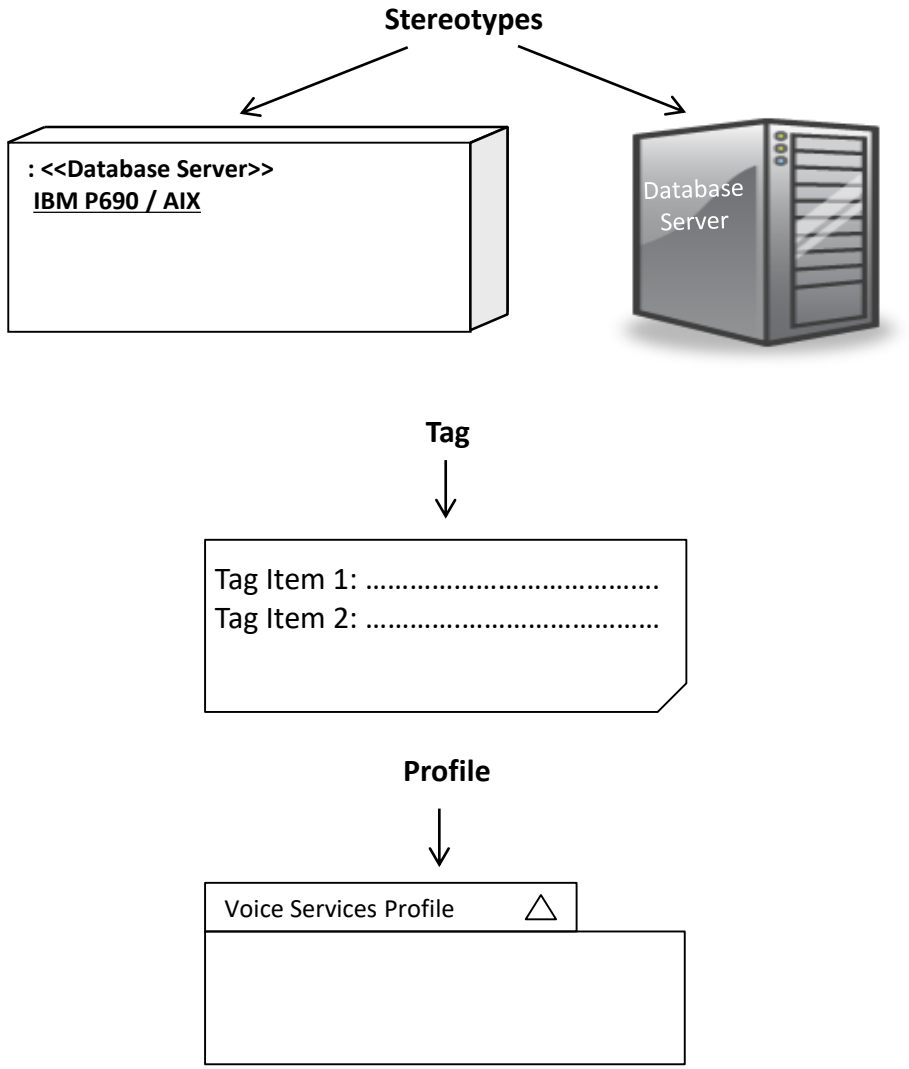
Each node may contain other artefacts and associations can be identified between nodes.



Customisation of UML: stereotypes, profiles and tagged values

An important aspect of UML is that it can be customized for specific purposes. There are three important language features that support this:

- Stereotypes: labels or graphics added to the standard diagram notation element to describe the role that the element is playing in the diagram.
- Tagged Values: allow you to add additional information to UML models. They are represented as notes added to structures within UML such as profiles, components and classes.
- UML Profiles: a UML profile is simply a collection of customizations that have been defined for a particular problem/solution domain. Examples include Service Oriented Architecture and Voice Services. They can be presented as packages.



4

The behavioural models

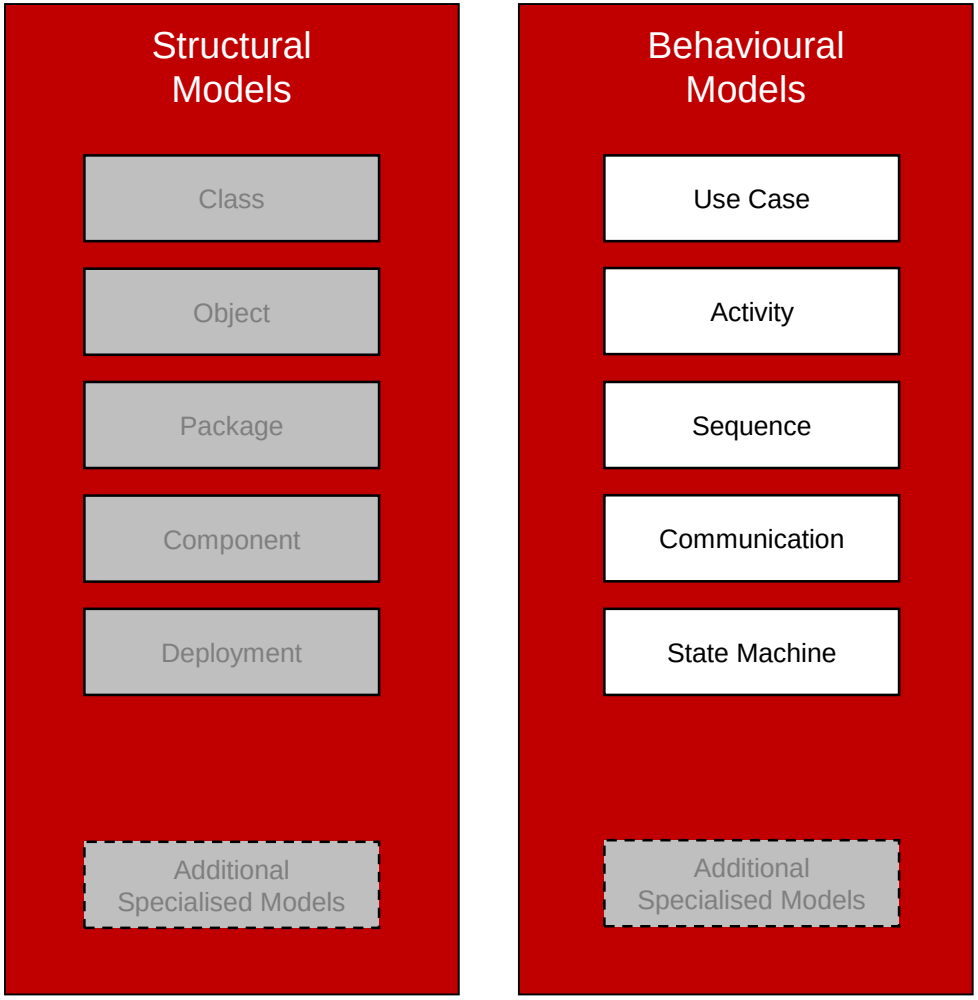
The behavioural models

The behavioural models represent the "activities, events and flows"" of concern in our business.

These may be in the world of business processes or in the execution paths of software or physical systems.

The behavioural models enable us define these behaviours and then organise them in the most effective ways.

The following slides provide examples of each of the models shown.



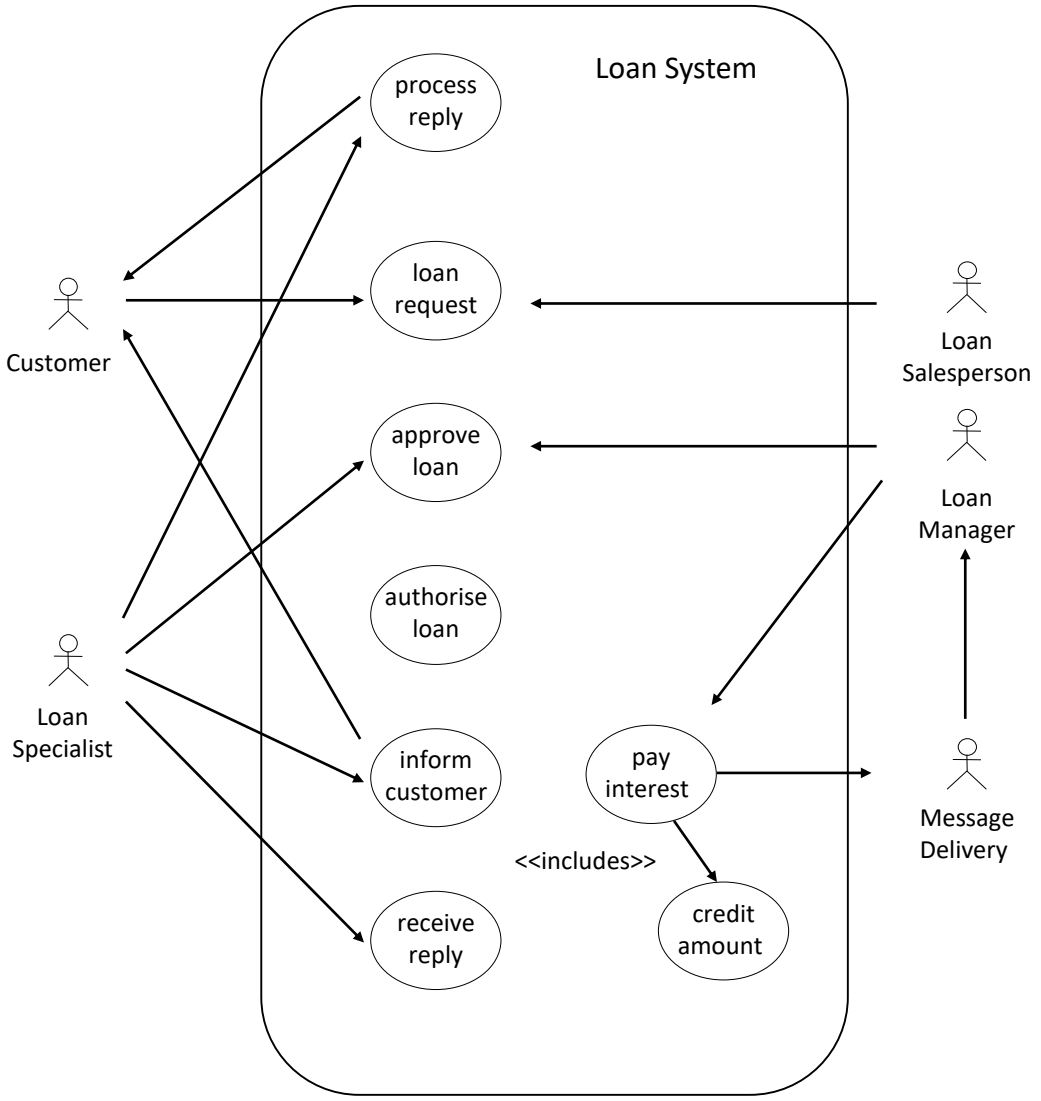
Use case diagrams

A use case model shows the interactions between a system and the external actors that use that system across the system boundary.

An actor represents a role performed by an external agent. That external agent could be a person or another system.

Each use case (such as “make loan request” in the diagram) is an atomic sequence of steps that delivers a unit of value to the actor performing the role.

A use case may also “include” the functionality of another case; or it may “extend” the functionality of a use case to which it refers.



Use case description

Each use case has a description in the form of a set of structured clauses.

This format can be varied, the one shown is about average for the format type and level of complexity .

The aim of a use case is to clearly describe all of the exchanges between the actors and the system; and the state that the system will be left in once the use case completes.

A use case should be written so that all stakeholders can understand it minimising "technical" jargon.

High level use cases are similar to the later developed concept of user stories but tend to have a more formal structure.

Use Case: Pay Interest
Description:
The due interest on all deposit accounts for the last month has been calculated and credited to the accounts and a report produced showing the interest paid.

Actors:
Manager
Printer Queue

Assumptions:
None

Steps:

- The Manager starts the pay interest Use Case (on the 28th of each month)
- The pay interest job is submitted to the system
- For each deposit account held within the deposit account file, the interest due is calculated and the credit amount Use Case is executed for each account that has interest due.
  [Exception: - Deposit Account Value <= 0]
- When all of the accounts have been processed a report is created and stored in the system showing the interest paid on each account and the total amount of interest paid; an e-mail is sent to the to the Manager informing them to access and review the report; and the use case terminates

Alternate Course:
None.

Exceptions:
[Deposit Account Value <= 0]
If a deposit account is found with a balance of zero or less then no interest is calculated, the credit amount use case is not executed for that account, and the report entry for the account will clearly highlight the account value.

Use case description – quality and performance attributes

The use case description can be extended by including a statement about the performance and quality attributes.

The set of performance and quality attributes should reflect the latest set requested as part of our specify and plan process this can be as small as 5 and as many as 30-40.

A sample set is shown on the slide.

Use Case Performance Attributes:

- **The number of instances of each Actor:**
  There is one manager for each branch.

- **The geographical distribution of the Actors / System Components:**
  The managers will be located in the branch along with the branch system upon which the request for processing interest payments is input. The calculation and payments will be processed on a central server system.

- **The number of concurrent instances of the Use Case:**
  A branch may have 1 concurrent instance of the Use Case. The central system may have up to 500 concurrent requests from branches for processing interest payments. These will be queued and processing sequentially

- **The response time required for an individual instance of this Use Case:**
  The Use Case should be able to be executed overnight in the month end batch processing window of 17.00 to 08.00. All of the branch runs together should be processed in a maximum of 3 hours.

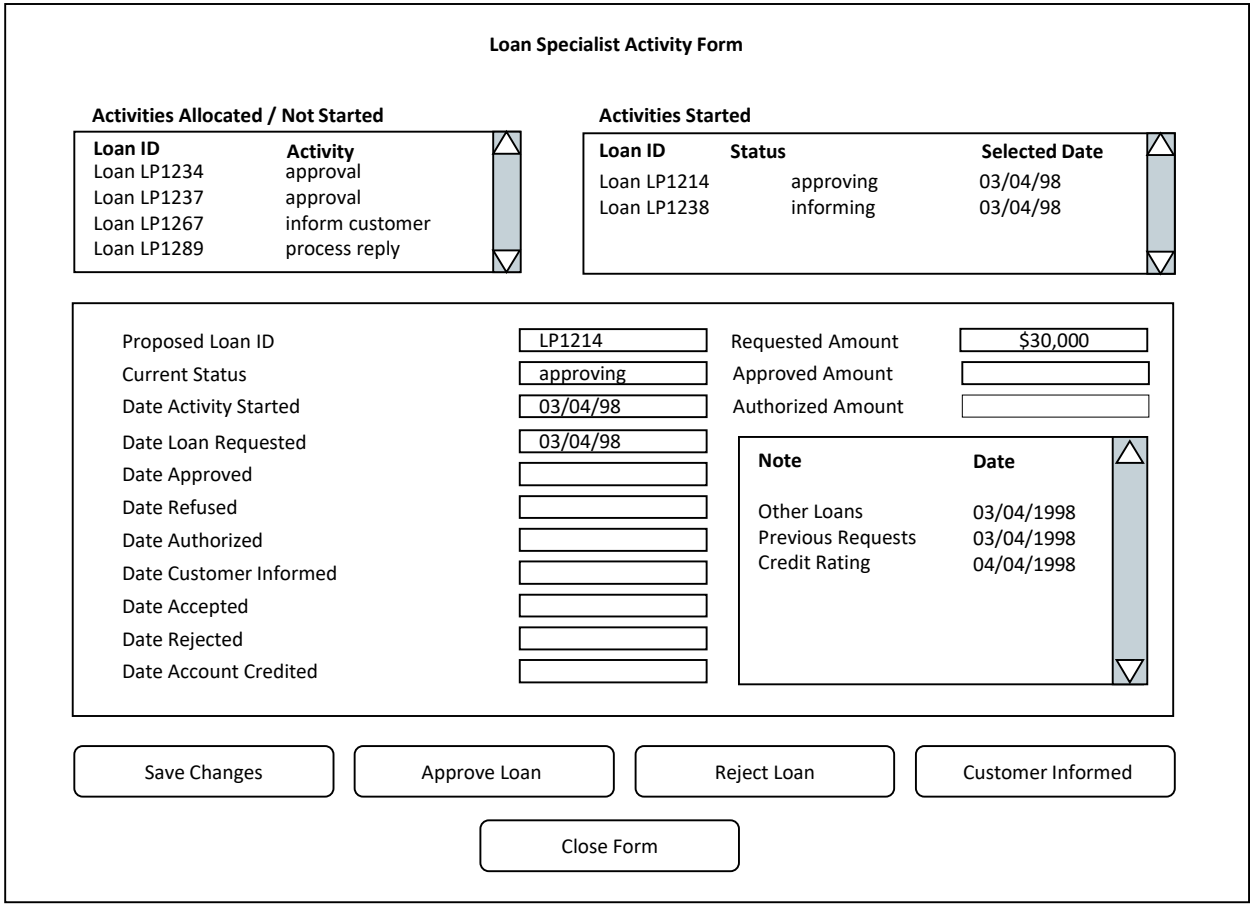
- **The security requirements for this Use Case:**
  The Use Case should only be able to be executed by authorised Bank Employees performing the role of a Manager.

- **The reliability of this Use Case in its execution:**
  The Use Case should always execute reliably unless there are system infrastructure problems from which the Use Case should be able to exit gracefully.

Supplementing the use case with an interface prototype

While not a formal part of UML there is often a need to produce a model of the user interface to improve the discussion about the requirements and to also identify and discuss all of the alternate and failure paths.

In some cases initial discussions about requirements should ignore the physical interface issues and just focus on the business events . In others the user interface makes it much easier to have discussions about the business events (but be careful not to allow the nature of the interface to determine the business events).



Supplementing the use case with scenarios using sequence diagrams

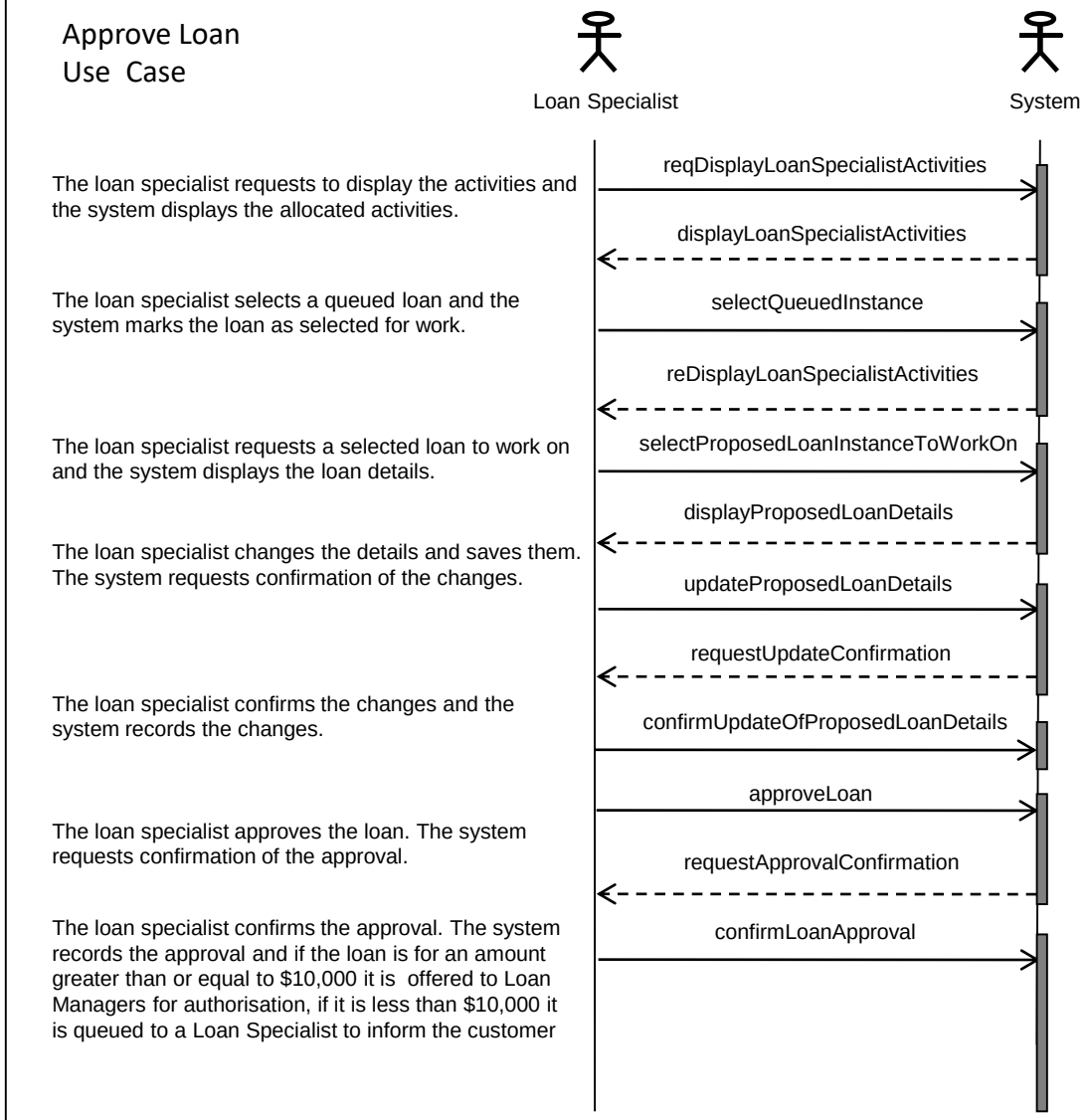
A scenario diagram is a specific type of sequence diagram that describes the interactions between an actor and a system when a use case is run.

Each specific event is identified and the exchange that occurs during the event.

Multiple scenarios may be modelled for the "happy path", "alternate paths" and "failure paths".

The one shown is the "happy path" for the approve loan use case.

The scenario can be described by using the defining sentences and / or the specific operation names depending on the level of detail required, the audience, and the need to connect the logical operation to their specific implementations.



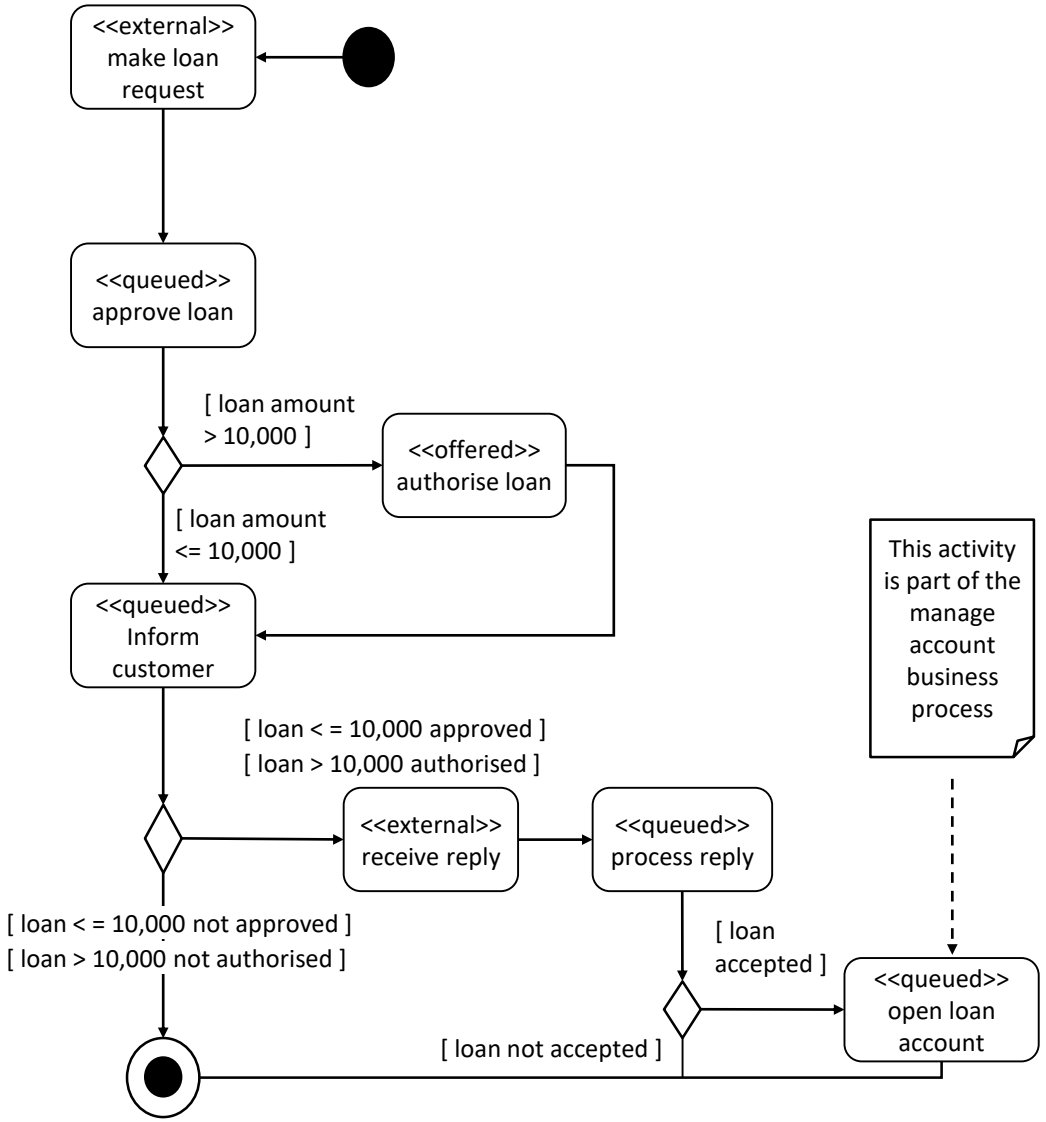
Activity diagrams

Activity diagrams are used to display a sequence of activities from a start point to a finish point.

Activity diagrams can used to model different aspects of sequences including business processes and software execution paths.

The example shown is of a business process model for a loan request.

Activity diagrams can be partitioned and nested and they can include swimlanes grouping activities within role or organisation boundaries.



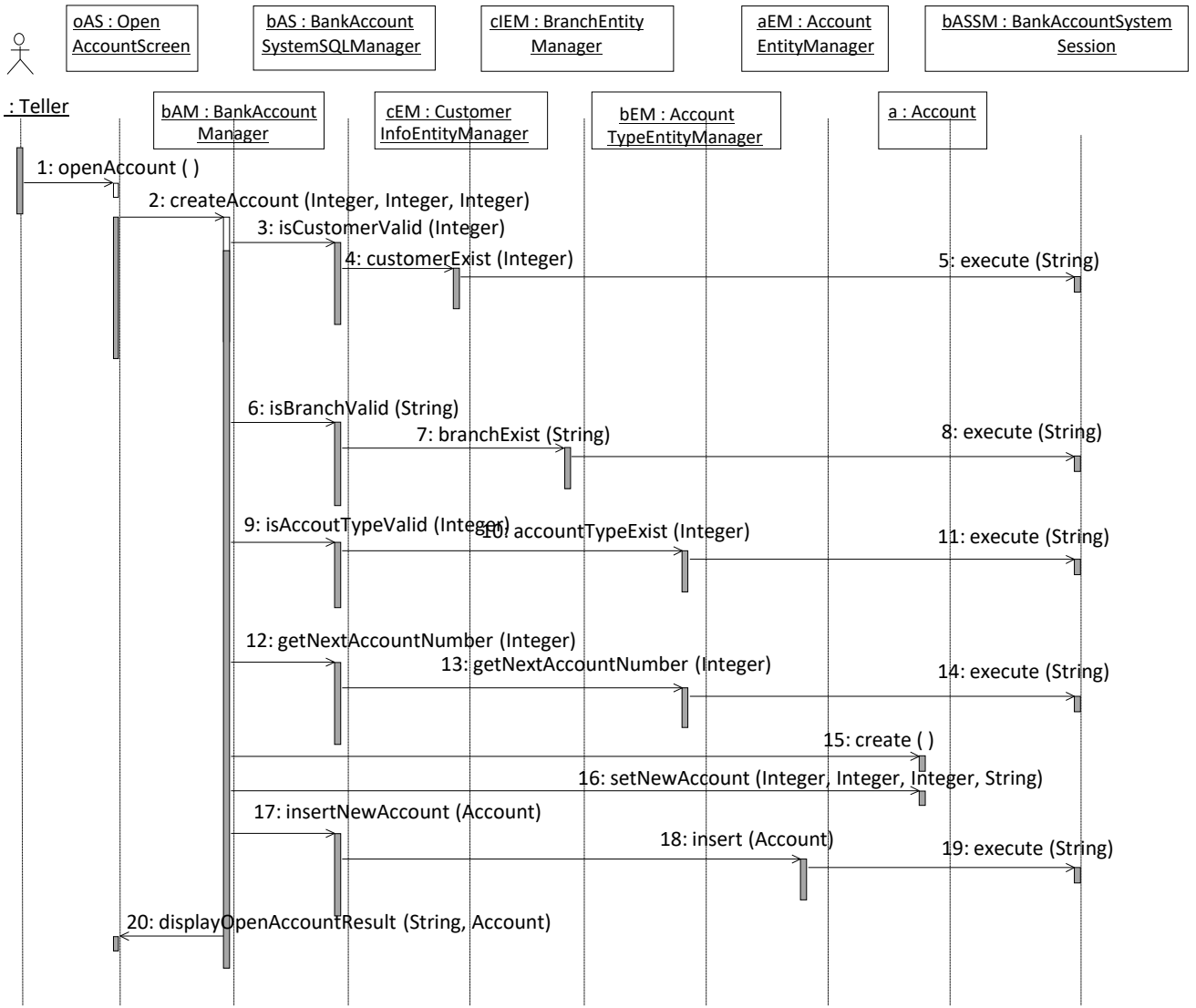
Sequence diagrams

Sequence diagrams describe the interactions between objects triggered by a particular event. In this case the request to open an account.

A sequence diagram typically shows a number of objects at the top of the diagram with lifelines below to show activity over time.

Events are shown invoking methods on the objects.

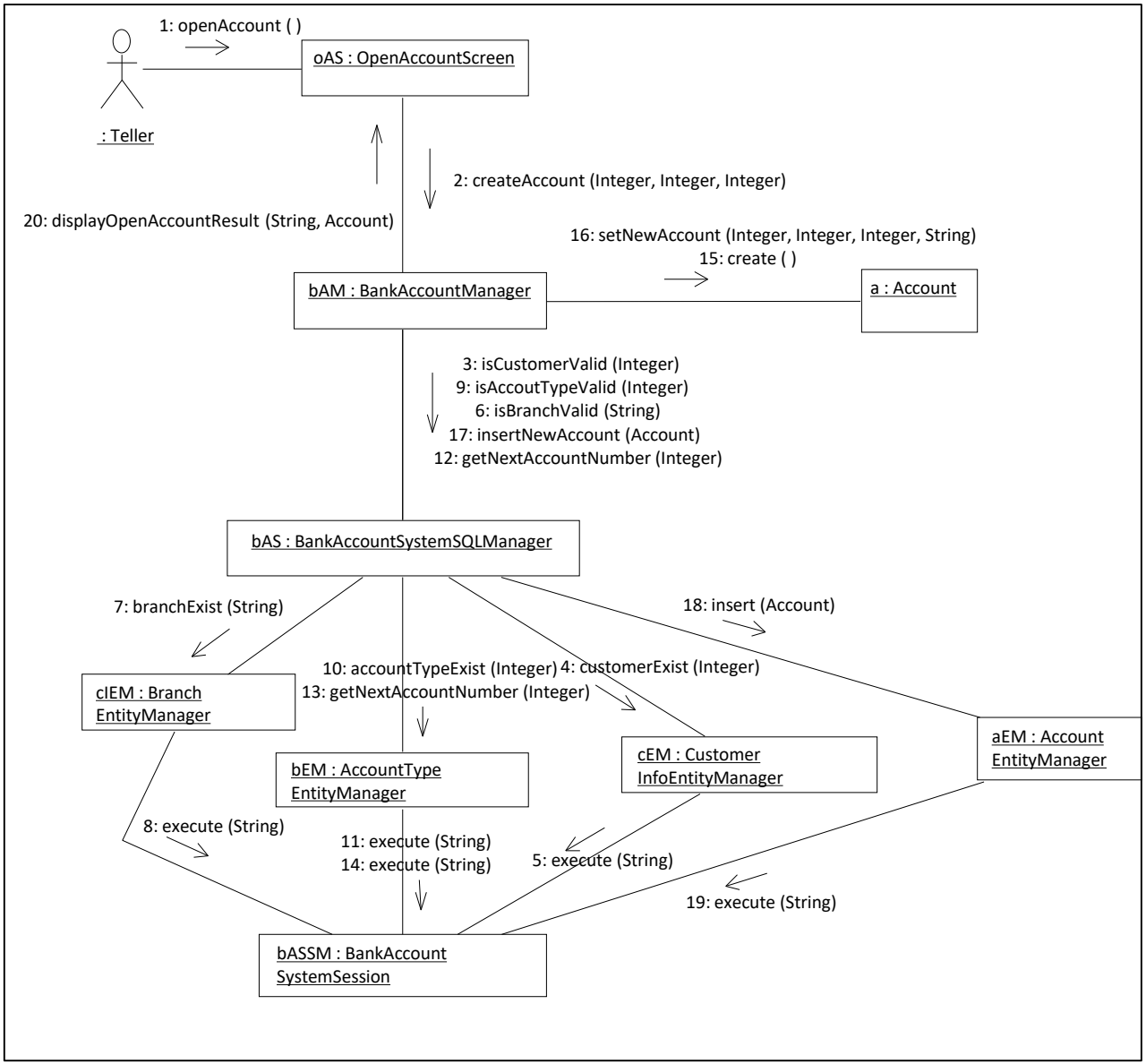
A sequence diagram is very useful for showing the order and flow of execution between objects.



Communication diagrams

Communication diagrams (previously called collaboration diagrams) present the same information as sequence diagrams but in different format.

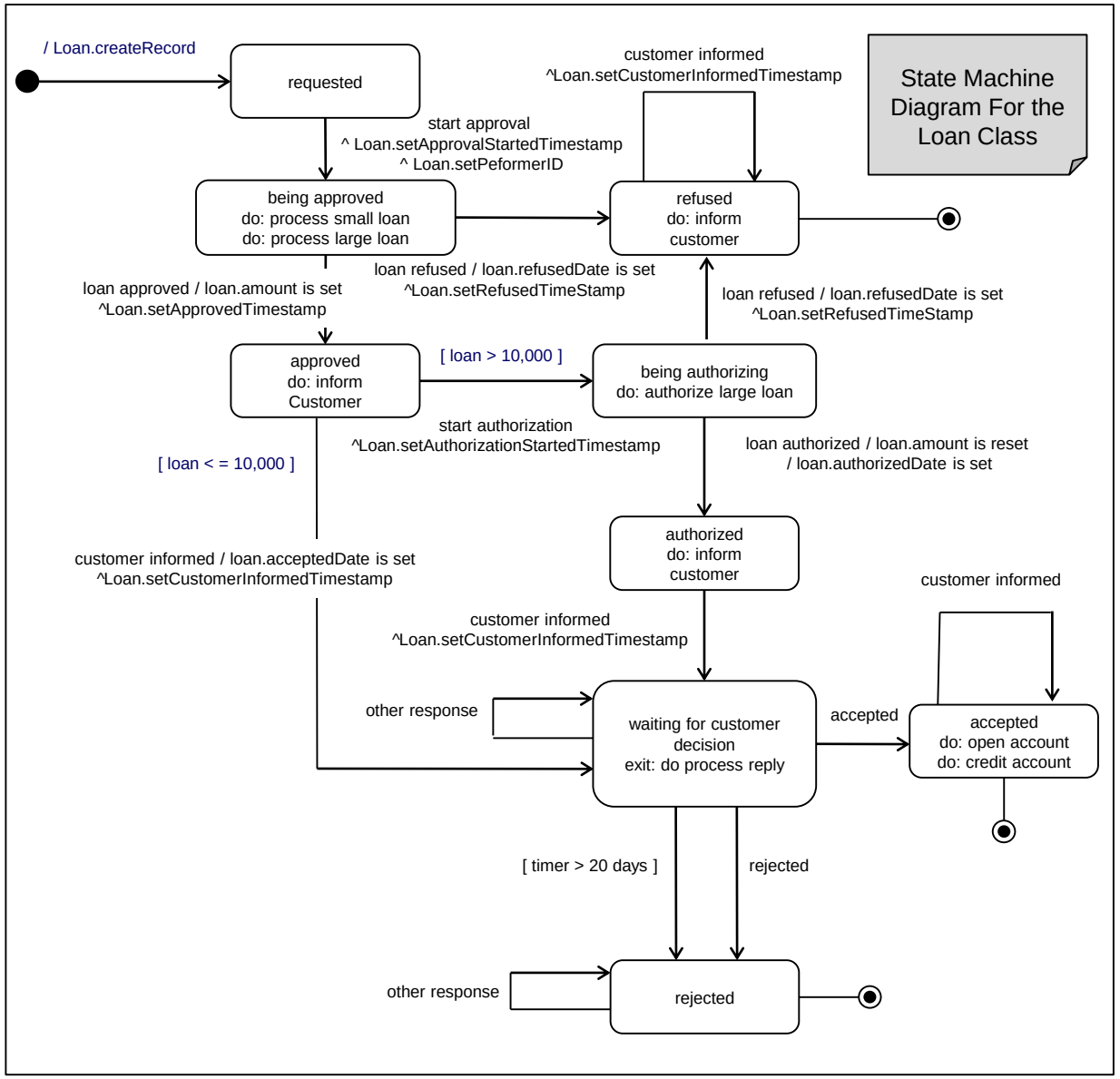
They show the event flows between objects in a more structural view enabling the analyst or designer to understand and shape the overall architecture of collaborating objects.



State machine diagrams

State machine diagrams show the states of an object / class and the actions that drive the transitions between those states.

They are useful when you need to explore the detail of a complex lifecycle for an object / class.



The End