

# COMET - Platform specific model

<!-- -->

## Table of contents

1 Introduction.....	2
2 Platform profile model.....	2
2.1 Goals.....	3
2.2 Related model artefacts.....	3
2.3 Methods and techniques.....	3
2.4 Deliverables and notation.....	4
2.5 Examples.....	4
3 Component Implementation Model.....	5
3.1 Goals.....	5
3.2 Related model artefacts.....	6
3.3 Methods and techniques.....	6
3.4 Deliverables and notation.....	6
4 UMT Configuration Model.....	7
4.1 The platform-independent architecture model.....	7
4.2 Implicit PSM .....	8
4.3 Implicit PSM in a UML tool (PIM annotation).....	8
4.4 Implicit PSM in UMT.....	9
4.5 Considerations.....	10
5 Deployment Model.....	11
5.1 Goal.....	11
5.2 Methods and techniques.....	11
5.3 Client and server.....	12
5.4 Deliverables and notation.....	13

## 1. Introduction

As the name indicates this model defines the result of mapping the *component model* to an implementation on a particular infrastructure. Figure 1 illustrates the work products in the platform specific model.

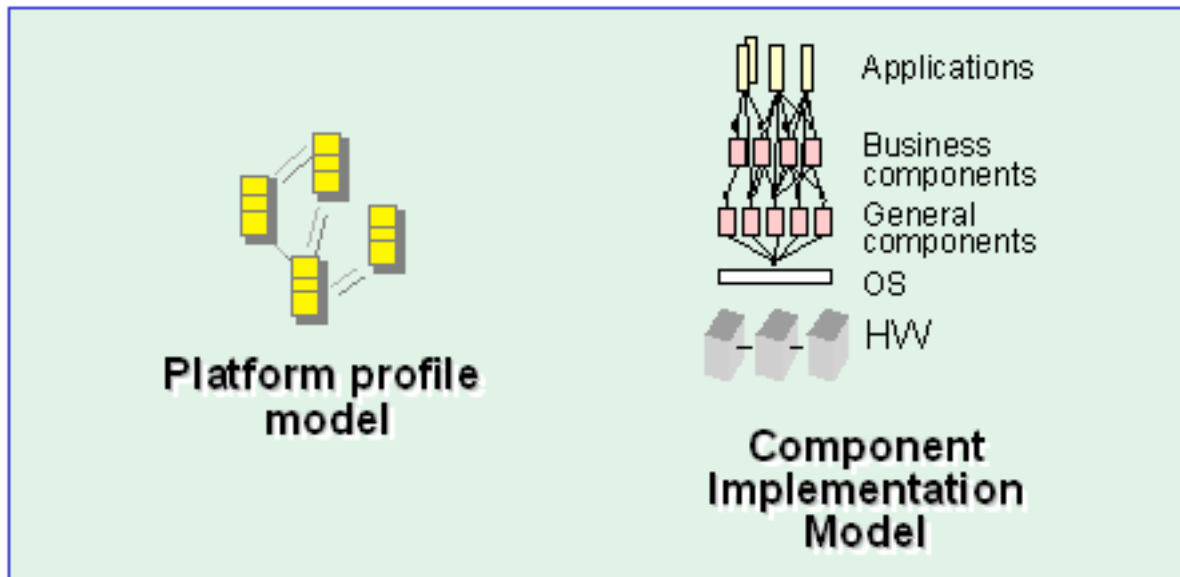


Figure 1: Platform specific work products

The Platform-specific Model contains the following work products:

- The Platform Profile Model (explicit PSM), which specifies the system in alignment to the actual technology profile for the specific platform.
- The Component Implementation Model, which describes the implementation of the component specifications in a given programming language, and the deployment properties/configurations for the target computing platform (hardware, operating system, etc.) in which the system is to run.

In addition two other work products may be developed:

- UMT Configuration Model (Implicit PSM in a code generator tool)
- The Deployment Model should describe the deployment properties/configurations for the target computing platform (hardware, operating system, etc.) in which the Product is to run.

## 2. Platform profile model

## 2.1. Goals

---

The purpose of this model is to specify the system in alignment to the actual technology profile for the specific platform (e.g. based on the UML profile for EJB).

## 2.2. Related model artefacts

---

The Business Model and the Requirements Model, but especially the Architecture Model serves as input.

## 2.3. Methods and techniques

---

A component environment might be any technology environment capable of realising a set of components, following the rules of conformity within that specific environment. This can be a distributed object environment (CORBA, COM, EJB), a messaging environment (JMS, MQSeries), etc. It includes a set of available infrastructure services (transaction support, security, naming, concurrency, and so on) on which the application component can depend in that environment.

Some main issues when mapping to these kinds of target technology are (many of these aspects are controlled by the technology profile chosen):

- **Technology type:** A classification of a range of technologies based on a set of technology criteria. Examples are object-oriented programming languages (e.g. Java, Smalltalk, Delphi), function-oriented programming languages (e.g. Lisp), database types (relational, object-oriented, etc.), database access mechanisms (e.g. JDBC, ODBC, SQL), interaction types (see below), and more.
- **Interaction type:** A specialisation of technology type, covering aspects of interaction mechanisms, i.e. how one component interacts with another, i.e. a set of message types describing how a component interacts with other components.
- **Message:** Usually a short communication transmitted by words, signals, or other means from one person, station, or group to another. Here used for a message sent from one (software) component to a set of others.
- **Message type:** Type of message, a part of interaction type. Classifier for the types of messages that can be sent from a component to another. An example is a synchronous message.
- **Communication mechanism:** The mechanism by which a component sends a message to other components, e.g. Java RMI, socket, or RPC.
- **Operation parameter type, kind (in/out/in-out/return) and reference restrictions**
- **Type system**
- **Error and exception handling mechanisms**

- Interface inheritance and support restrictions
- Operation sequence
- Interface properties
- Object creation mechanisms
- Event handling
- Transaction handling
- Security and general QoS

These aspects can be part of a platform specific profile like the EJB-profile. This profile defines how a *platform specific model* should be structured for an EJB environment. Ideally, the platform specific model should be fully generated by the modelling tool. In practise it will most often be partially generated, and possibly refined by the user.

## 2.4. Deliverables and notation

---

The *platform profile model* consists of a set of UML models according to the chosen technology profile.

## 2.5. Examples

---

Figure 2 depicts an example of a Platform Profile Model according to the EJB profile.

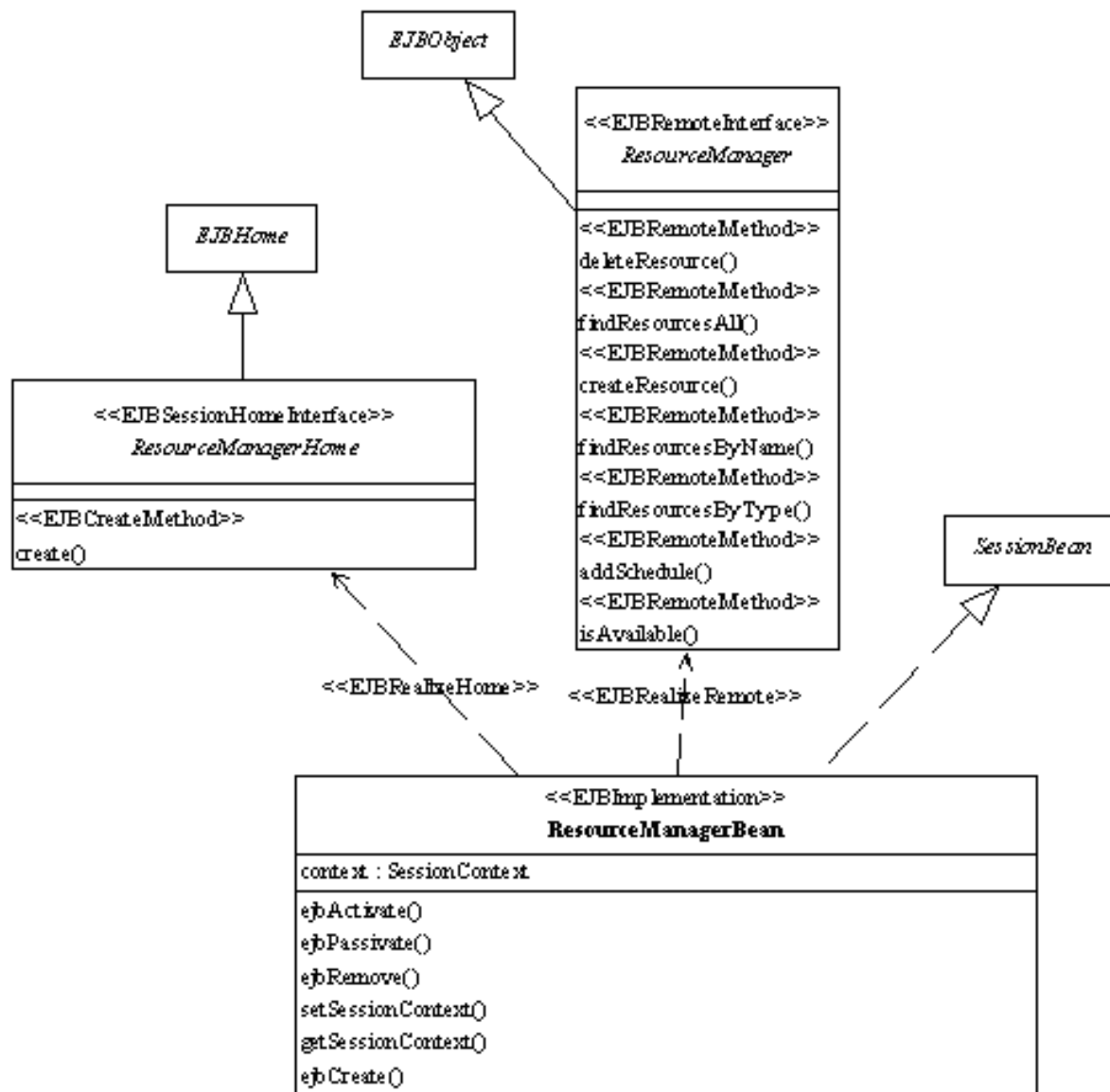


Figure 2: Example of a platform specific model for EJB

### 3. Component Implementation Model

#### 3.1. Goals

The purpose of this model is to deliver an implemented, compiled, and running Product according to the end users' requirements. The Component Implementation Model describes

the implementation of the component specifications in a given programming language, and the deployment properties/configurations for the target computing platform (hardware, operating system, etc.) in which the product is to run.

### 3.2. Related model artefacts

---

This model takes the Platform Profile Model as input to the Component Implementation Model.

### 3.3. Methods and techniques

---

This model takes the Platform Profile Model as input and refines it to an actual implementation in a programming language of choice. For certain component environments such as EJB, which is a language extension of Java, the programming language is given in advance. Other environments such as CORBA component model is language portable and can be implemented in many supported programming languages.

In refining the implementation, modelling tools should be used on the *platform specific model* to automatically generate a mapping to the *component implementation model*. The generated model may be used as the actual implementation, but in many cases the model needs to be refined further by experienced programming language developers.

It is important that further refinements of the Component Implementation Model should be developed using a model-driven approach, meaning that the UML model should be regarded as the source, and must always reflect the latest version of the source code. Some existing tools on the market, e.g. TogetherSoft's "Together Control Center" (see [www.togethersoft.com](http://www.togethersoft.com)), already supports this approach. Ideally, new functionality/features should be introduced in the models and included by the implementation, either by code generation (if the tool supports it) or manually. This is called *model driven development*. This should be introduced in the *platform independent model* and propagated to other models as necessary. Of course, bottom up inclusion of features should also be allowed. New features must then be reflected in the models.

### 3.4. Deliverables and notation

---

The Component Implementation Model consists of a set of UML diagrams and implementation code that describes:

Programming language implementation: UML class diagrams that show the structure of the implementation in the chosen programming language, and implementation code that implements those models.

Deployment model: UML deployment diagrams that show the structure of the deployment with accompanying property/configuration descriptions.

## 4. UMT Configuration Model

### 4.1. The platform-independent architecture model

The baseline for the PSM mapping is the architecture model, which is an artefact produced during architecture modelling. It describes the product components, and their provided and required interfaces. The architecture model is not concerned with platform-specific details, such as communication protocols, realisation mechanisms etc.

The figure below shows an example of a high-level architecture model, describing a set of components with provided and required interfaces.

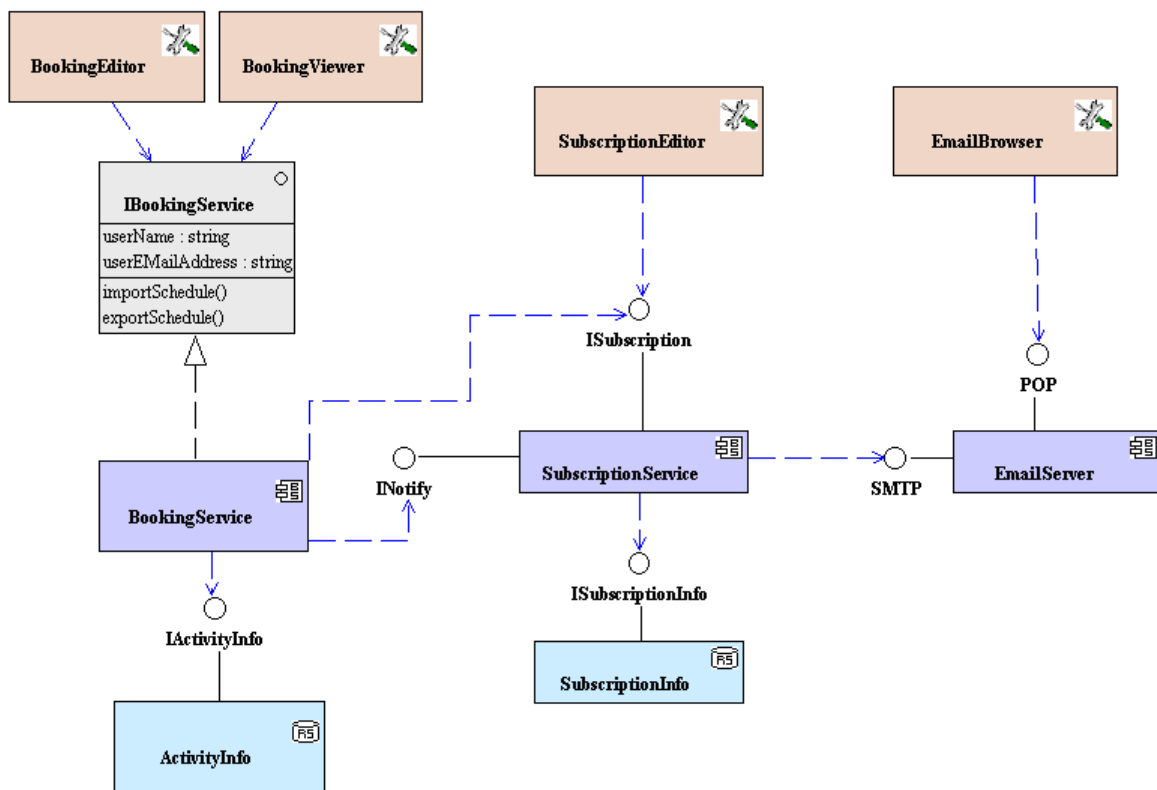


Figure 3: Architecture Model Example

We have identified two ways of specifying the platform-specific details of a product; an implicit and an explicit way. In COMBINE we use the implicit way supported by the UMT code generation tool.

The explicit way defines refinements of the architecture models that are specific to the target

platform, for instance by using a specific platform UML profile for modelling. An example of this is to apply the UML EJB profile and create an EJB architecture model in UML, which in turn is reflected in the implementation.

The implicit way does not refine the architecture model to platform level using model refinements. Instead, we can tag model elements with target platform information, which is used by code generation tools to generate the platform architecture. The tagging can be done either in the modelling environment (the UML tool) using for example tagged values, or in the code generation tool platform properties.

## 4.2. Implicit PSM

---

The implicit PSM is based using simple properties that identify the essential platform properties that enables components to be transformed to the correct target platform, given that a transformation tool (code generator) is familiar with the set of platform patterns used. For example, we might introduce a property for a BusinessService component named 'EJB', which signifies to the code generator that this component shall be mapped to an EJB component. Another example is associating properties with a component's interface. One interface might for example be available as Web Service, EJB and Servlet. The Code Generator must know how to perform the mapping to these platforms.

## 4.3. Implicit PSM in a UML tool (PIM annotation)

---

In a UML tool, Tagged Values (Extended Properties) can be used to attach platform information to the components and its interfaces, as shown in the figure below (the tagged values are visualised in notes.)

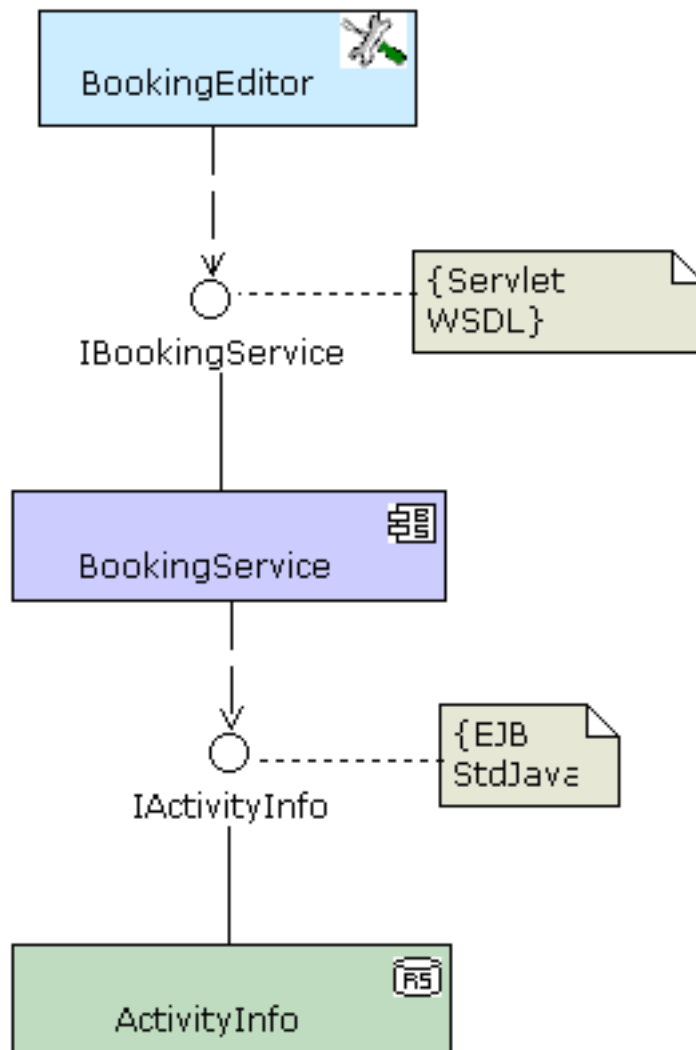


Figure 4: Tagged Architecture Model

#### 4.4. Implicit PSM in UMT

If the UML model is kept strictly independent of technology, without tags that indicate target platform, this information may be added within an external code generation tool, for example UMT.

In UMT, properties may be associated with model elements. These properties can be platform parameters that control the output of a code generation process.

The figure below shows UMT with platform-specific properties associated with the IBookingService interface of the Business Service Component.

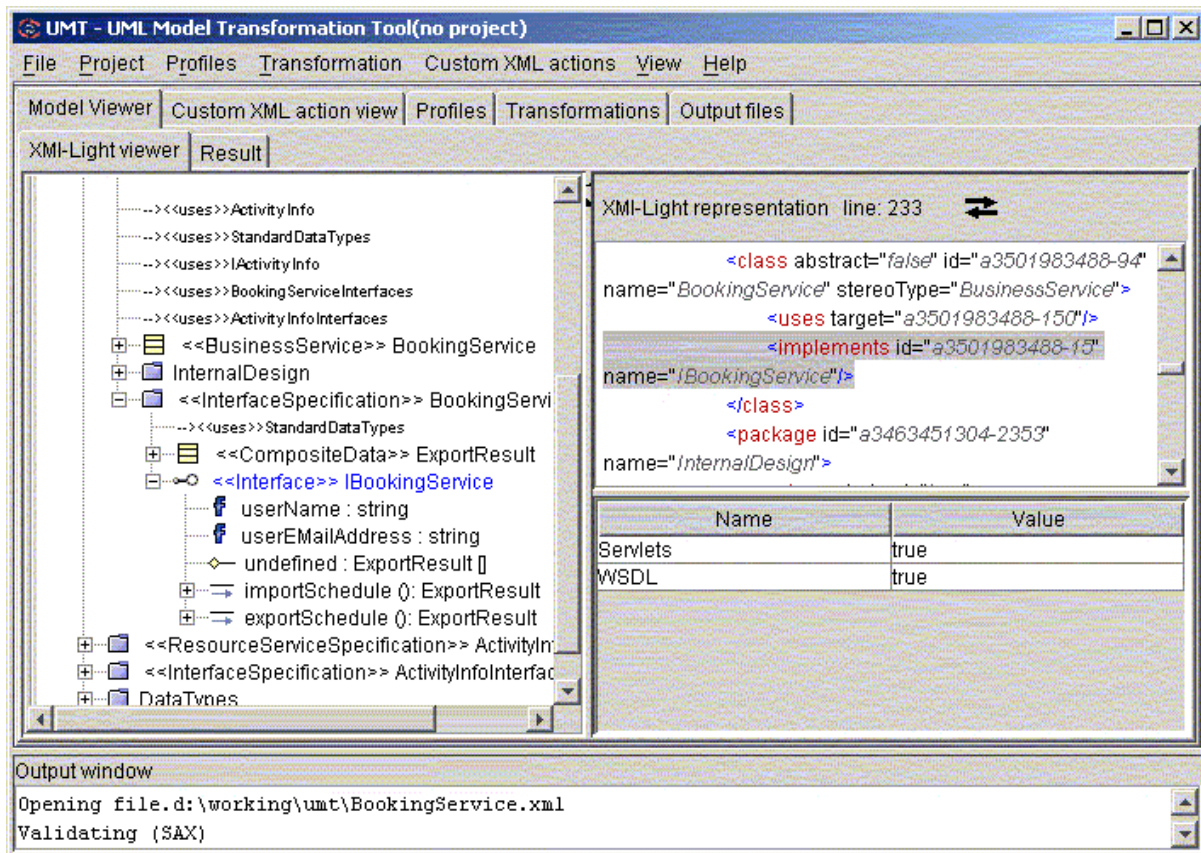


Figure 5: Example UMT with platform properties

The properties defined within UMT can be used by the code generator templates that process the model and generates the platform specific implementation code.

Configuration in the UMT (or another code generation tool) to define the PSM mapping might be used in combination with an annotated UML model, where the annotation is done on the PIM model to attach platform specific information. This PSM information might then be utilized in the code generator tool to aim automatic configuration before executing the code generation.

#### 4.5. Considerations

In the context of the Component Centre a component environment is the most relevant target technology. A component environment might be any technology environment capable of realizing a set of components, following the rules of conformity within that specific environment. This can be a distributed object environment (J2EE, CORBA, .Net, EJB), a messaging environment (JMS, MQSeries), etc. Each technology environment includes a set of available infrastructure services (transaction support, security, naming, concurrency, and

so on) on which the application component can depend in that environment.

Some main issues when mapping to these kinds of target technology are:

- **Technology type:** A classification of a range of technologies based on a set of technology criteria. Examples are object-oriented programming languages (e.g. Java, Smalltalk, Delphi), function-oriented programming languages (e.g. Lisp), database types (relational, object-oriented, etc.), database access mechanisms (e.g. JDBC, ODBC, SQL), interaction types (see below), and more.
- **Interaction type:** A specialisation of technology type, covering aspects of interaction mechanisms, i.e. how one component interacts with another, i.e. a set of message types describing how a component interacts with other components.
- **Message:** A usually short communication transmitted by words, signals, or other means from one person, station, or group to another. Here used for a message sent from one (software) component to a set of others.
- **Message type:** Type of message, a part of interaction type. Classifier for the types of messages that can be sent from a component to another. An example is a synchronous message.
- **Communication mechanism:** The mechanism by which a component sends a message to other components, e.g. Java RMI, socket, or RPC.
- **Operation parameter type, kind (in/out/inout/return) and reference restrictions**
- **Type system**
- **Error and exception handling mechanisms**
- **Interface inheritance and support restrictions**
- **Operation sequence**
- **Interface properties**
- **Object creation mechanisms**
- **Event handling**
- **Transaction handling**
- **Security and general QoS**

## 5. Deployment Model

### 5.1. Goal

The Deployment Model should describe the deployment properties/configurations for the target computing platform (hardware, operating system, etc.) in which the Product is to run.

### 5.2. Methods and techniques

To deploy a product there are typically some constraints that need to be taken into account.

Typically there is already a hardware infrastructure in place (machinery, networks etc) as well as associated software infrastructure (OS, communication protocols etc). The possibilities to add or change the already existing infrastructure will vary. However, the product will typically also set some requirements to the execution environment, both when it comes to hardware and software. Thus, the infrastructure at hand will set constraints on how to deploy the product and the product will have a set of requirements to the infrastructure in which it will be deployed.

The reference architecture will also constrain the deployment as it indicates the logical distribution of the set of components. Thus, when the infrastructure is in place, deploy the components of the product using the the Component Structure the reference architecture as guidance.

Important aspect that should be described as part of the Deployment Model is:

- Distribution of the components
- Integration with other products and actors

### 5.3. Client and server

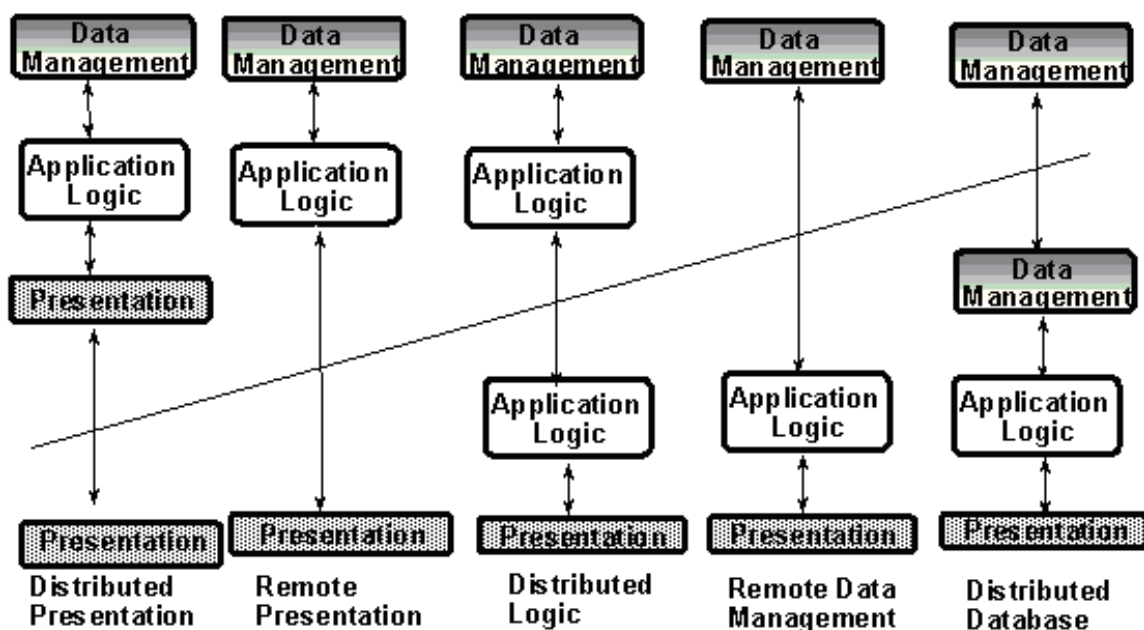


Figure 6: Various splits between client and server.

Figure 6 shows the Gartner Group's reference model for client/server systems which presents 5 different ways to split the client and server part (illustrated by the diagonal line), related to a separation of a system into three logical parts: Presentation, Application Logic and Data

Management. The split depends typically on what kind of machinery and devices that will be used. In some situation this will vary and the system is required to dynamically adapt to the actual environment. E.g., the client device might be both a PC and a Palm or a mobile phone.

#### **5.4. Deliverables and notation**

---

The Deployment Model consists of

- UML deployment diagrams that show the structure of the deployment with accompanying property/configuration descriptions.