

# HOPEX Application Design

## User Guide

HOPEX V3.2



**M E G A**  
SEE THE BIGGER PICTURE

Information in this document is subject to change and does not represent a commitment on the part of MEGA International.

No part of this document is to be reproduced, transmitted, stored in a retrieval system, or translated into any language in any form by any means, without the prior written permission of MEGA International.

© MEGA International, Paris, 1996 - 2020

All rights reserved.

HOPEX Application Design and HOPEX are registered trademarks of MEGA International.

Windows is a registered trademark of Microsoft Corporation.

The other trademarks mentioned in this document belong to their respective owners.

# INTRODUCTION



**HOPEX Application Design** is based on the tools offered by the **HOPEX** platform to assist architects in specifying updates to their IT system. The advanced product functions described here are illustrated by the example of a purchase request processing application.

The following points are covered in **HOPEX Application Design**:

- 6 ["Creating an Application Design Project", page 31.](#)
- 6 ["Describing an AD Project and its Application Environment", page 29.](#)
- 6 ["Defining the Objectives and Requirements of a Project \(AD\)", page 41.](#)
- 6 ["Describing the Functional Specifications of a Project", page 51.](#)
- 6 ["Describing the Data of a Project", page 65.](#)
- 6 ["Describing the Technical Specifications of a Project", page 77.](#)

For more details on the interface and functions of **HOPEX** in general, see:

- 6 ["Presentation of HOPEX Application Design", page 2.](#)
- 6 ["The HOPEX Application Design Method", page 5.](#)
- 6 ["Connection to the HOPEX Application Design Desktop", page 20.](#)

To start with **HOPEX Application Design**, see:

- 6 ["Creating an Application Design Project", page 30.](#)

# PRESENTATION OF HOPEX APPLICATION DESIGN

Combined with the products of the **HOPEX** suite, **HOPEX Application Design** offers a methodology and the tools to describe and plan your business transformation.

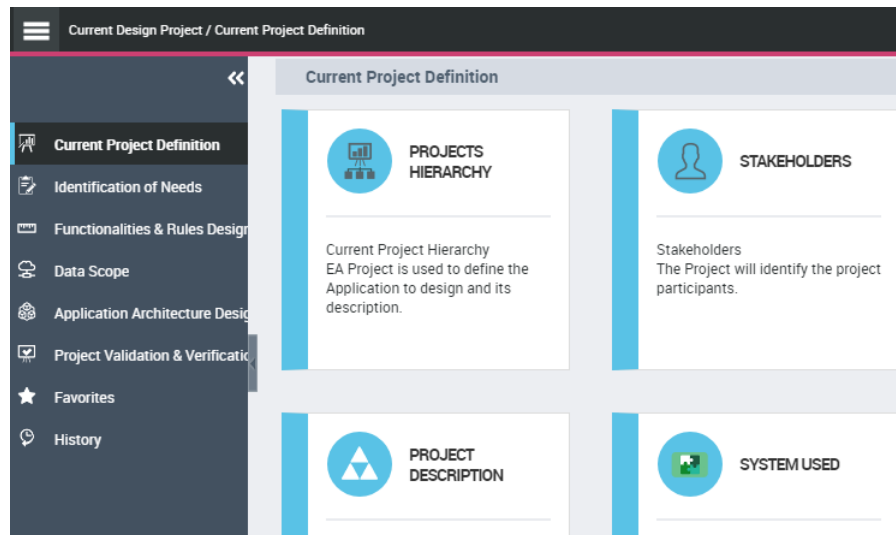
## The scope covered by HOPEX Application Design

To assist architects in describing software specifications, the **HOPEX Application Design** solution is based on the following methodology:

- Defining the project and its application scope
- Capturing business needs and project objectives
- A functional analysis and description of use contexts
- Modeling application data flows (data scope)
- Designing the technical architecture

A validation workflow offers the end client the possibility to validate the specifications by ensuring that its needs are met.

The solution interface follows this methodology. For each step, the solution supplies users with specific tools for each step.



The phases offered in standard mode are in keeping with an Agile development approach.

Analysis reports are available to simplify analysis of the subject and contribute to the development of complete and relevant specifications.

Upgrades and the integration of tests are performed outside of the project cycle phases managed with **HOPEX Application Design**.

---

## HOPEX Application Design Profiles

In **HOPEX Application Design**, there are a number of default user profiles with which specific rights and accesses are associated. These profiles are:

- the **Functional administrator for application design**,
- the **Application designer**,
- the **Application owner**,
- the **UML designer**,
- the **Application design contributor**.

### The functional administrator for application design

The functional administrator for application design is responsible for managing all the functional aspects of the application design. The functional administrator for application design has extended rights on all the objects managed. The functional administrator for application design is also responsible for organizing the work of the application designers.

- The functional administrator manages the creation of users and their assignment to profiles;
- The functional administrator creates projects and identifies the **HOPEX** repository objects that are part of the scope;
- The functional administrator creates the enterprise plans and identifies the **HOPEX** repository objects that are part of the scope;
- The functional administrator specifies the participants in the enterprise plan as well as the role of each;
- The functional administrator allocates users to one or another of the projects that constitute the entry points of the **HOPEX Application Design** desktop.

– For more details on the functional administrator desktop, see *"Presenting the desktop of the functional administrator for application design"*, page 23.

### The application designer

The application designer is the user profile of the **HOPEX Application Design** solution, responsible for driving application design projects.

He/she defines the application environments, application structures, interaction scenarios, application processes and interfaces.

If your license allows, and to enable the users connected to this profile to work, the application designer can also access the objects and main functionalities of the **HOPEX Business Process Analysis**, **HOPEX IT Architecture**, **HOPEX IT Portfolio Management** and **HOPEX Risk Mapper** solutions via the **HOPEX Business Architecture** desktop.

– For more details on the application designer, see *"Presenting the application designer desktop"*, page 21.

## The application owner

The Application Owner is responsible for specifying the characteristics of applications for which he/she is responsible, and to update them.

- For more details on the application designer, see "[Presentation of the application owner desktop](#)", page 23.

## The UML designer

The UML designer is a technical user profile of the **HOPEX Application Design** solution.

The application designer is responsible for the detailed specifications of the system and for building UML diagrams.

- For more details on the application designer, see "[Presentation of the application designer desktop](#)", page 24.

## The application design contributor

The application design contributor is the business function end user profile of the **HOPEX Application Design** solution. The application design contributor accesses the application in read-only to build reports.

- For more details on the application designer, see "[Presentation of the desktop of the application design contributor](#)", page 23.

---

## Business Roles of HOPEX Application Design

In **HOPEX Application Design**, there are, by default, *business roles* that can be assigned to certain users.

- ) A business role is used to assign a task to a person (example: describe an application) and where appropriate, for a specific location (example: Paris agency). Business function roles are assigned to persons and a person can have more than one business function role.

These business roles are:

- **Data Designer**, who is responsible for managing data.
- **Local Application Owner**, whose role is to provide the characteristics of applications for which he/she is responsible, and to regularly update these.
- **Software Designer**, whose role is to provide the functional characteristics of software for which he/she is responsible, and to regularly update these.

- For more information on the use of business roles in **HOPEX Application Design**, see "[Connecting a person to an organization](#)", page 35.

# THE HOPEX APPLICATION DESIGN METHOD

The method used in **HOPEX Application Design** is based on the concept of a project.

) *An Enterprise Architecture project is a part of a system whose study is entrusted to a single team, which transforms a system or part of a system to achieve a given objective.*

---

## Presenting the Method and Example

The specification of IS upgrade projects with **HOPEX Application Design** contains the following steps:

- > **Defining the scope of the project:** this first step defines the application scope of the project, its deliverables, how to break down your project if appropriate into sub-projects and the organization of participants in the project.
- > **Describing the objectives and requirements of the project**
- > **Building the functional specifications of the project:** this step consists of defining the functionalities expected and the use context of the project applications.
- > **Describing the project data:** this involves describing how an organization's data is used by the processes and the applications.
- > **Building the technical functional specifications:** the purpose of the project's technical specifications is to detail the technical architecture of the project components.
- > **Describing data exchanges:** this involves describing the exchange contracts between the components of a business or IT architecture.

## Example Overview

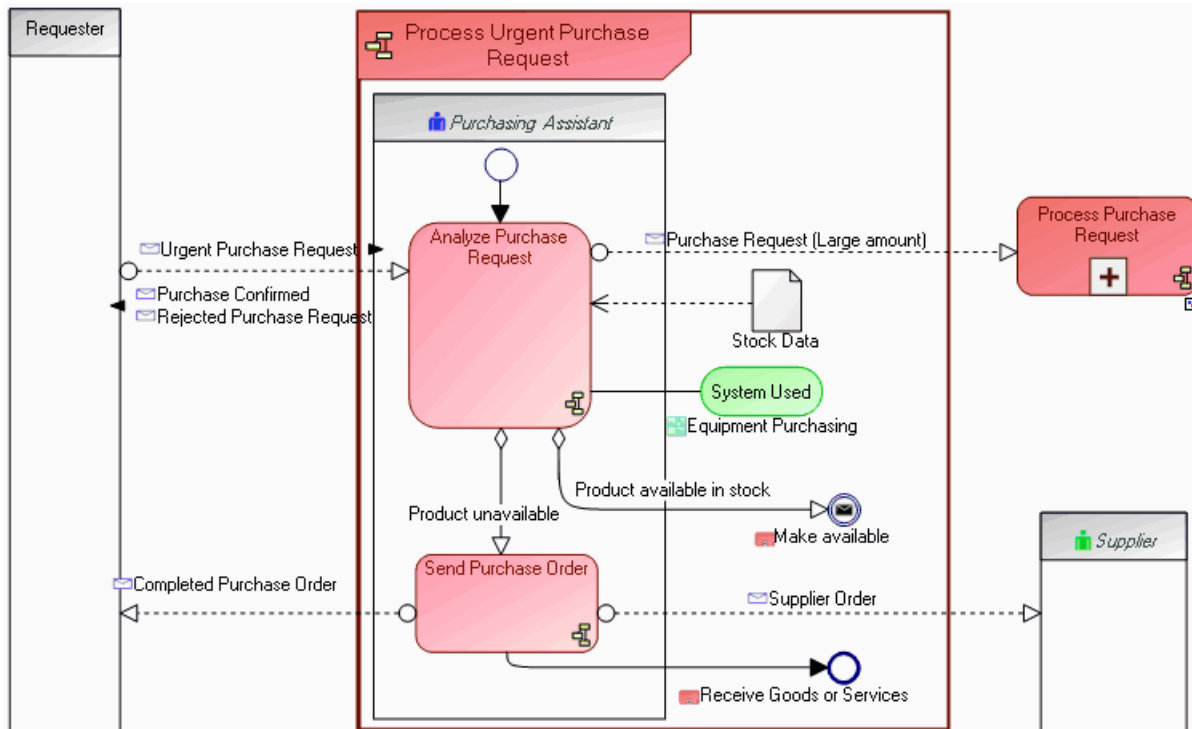
These different points of view are illustrated by the example of a boat rental enterprise that has decided to improve its purchasing process.

Availability of boats being a vital element of company activity, a specific process must be defined to process urgent purchases for boat maintenance and repair.

Company purchasing processes are split between operational purchase processes and investment purchase processes. Operational purchasing processes include purchase of spare parts and raw materials required for boat maintenance and repair.

## Organizational Context

The purpose of the new organizational process for processing urgent purchase requests is to increase responsibilities of the purchasing assistant, so as to be able to send purchase orders at very short notice.



### "Process Urgent Purchase Request" Organizational Process

The purchasing assistant begins by analyzing the purchase request. If the amount is large, the normal process purchase request process is implemented.

If the product is available, the assistant sends the availability request.

If not, the assistant sends a purchase order to the supplier. The rest of the processing is carried out within the framework of the normal "Process Purchase Request" organizational process.

To allow the purchasing assistant to rapidly analyze the purchase request and send the order, he/she is provided with a new application enabling access to data concerning stock and suppliers.

## Describing a Project and its Environment

The *project* is the basic element used by **HOPEX Application Design**.

An Enterprise Architecture project is a part of a system whose study is entrusted to a single team, which transforms a system or part of a system to achieve a given objective.

- For more details on creating a project, see "[Creating an Application Design Project](#)", page 30.

To describe the scope of the *project*, you must:

- describe the project applications and their exchanges
- describe the project deliverables

With **HOPEX Application Design**, you can also define the different organizations concerned by the project.

- For more information on managing organizations concerned by the project, see "[Describing the Organization of Project Participants](#)", page 34.

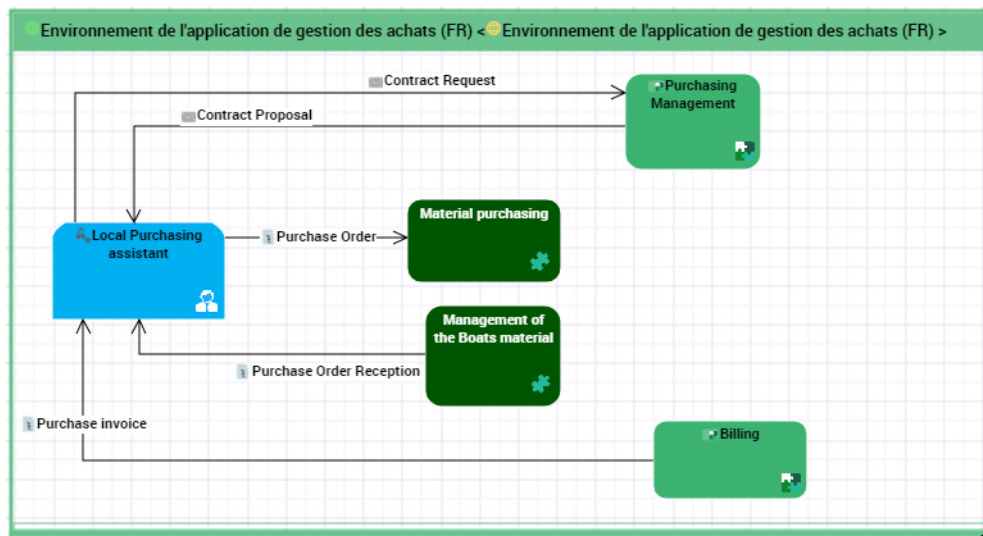
## Describing the project applications and their exchanges

An application environment is used to represent a use context of an application.

With **HOPEX Application Design**, you can use flow scenario diagrams to describe the flows exchanged by an application, an application service or a micro service in a particular context.

- For more details on flow scenario diagrams, see "[Describing a scenario of flows](#)", page 174.

The diagram below describes the flows exchanged in the context of the "Purchasing Management System" application.



Equipment purchases for boat maintenance are made directly by the Representation Office, using an "Equipment

Purchasing" application service. This calls a "Purchase Management" application located centrally at Headquarters.

If the purchase involves a new type of equipment or a new supplier, a contract request is sent to the supplier using the application at Headquarters. The supplier returns a contract proposal which is submitted to the requester.

In the other cases, a representative from the Representation Office sends a purchase order to the supplier using the "Equipment Purchasing" application.

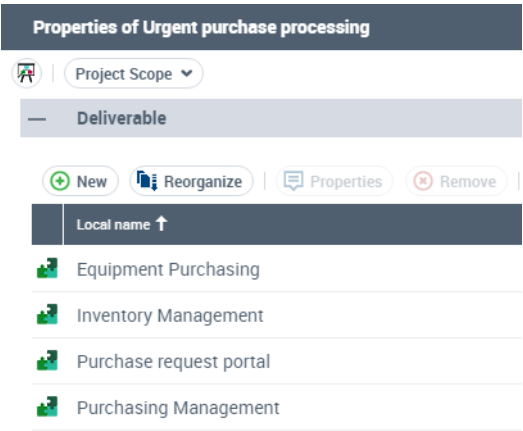
Reception of purchases is handled by the "Boat Inventory Management" application at the Representation Office.

The invoice is managed in the "Billing" application at Headquarters.

## Describing the project deliverables

After analysis, the application elements that will be allocated by the project are described in the project deliverables.

- For more details on updating project deliverables, see ["Describing a scenario of flows", page 174](#).



The "Equipment Purchasing" application is the main application impacted by the project, in particular for improvement of its access portal.

The "Stock Management" application is used by the "Inventory Management" application to view and update information on stock and in particular, spare parts.

The "Purchasing Management" application uses the "Catalog" database to send a purchase order.

The "Maintenance and Repair Management" and the "Spare Parts Management" applications also use the "Equipment Purchasing" application portal.

For more details on how to describe a project environment, see "[Describing an AD Project and its Application Environment](#)", page 29.

---

## Describing the Objectives and Requirements for a Project

To refine the description of the technical scope of a *project*, this step consists of defining the *goals* and the *requirements* of the different categories of users.

- ) *An objective is a goal that a company/organization wants to achieve, or is the target set by a process or an operation. An objective allows you to highlight the features in a process or operation that require improvement.*
- ) *A requirement is a need or expectation explicitly expressed, imposed as a constraint to be respected within the context of a project. This project can be a certification project or an organizational project or an information system project.*

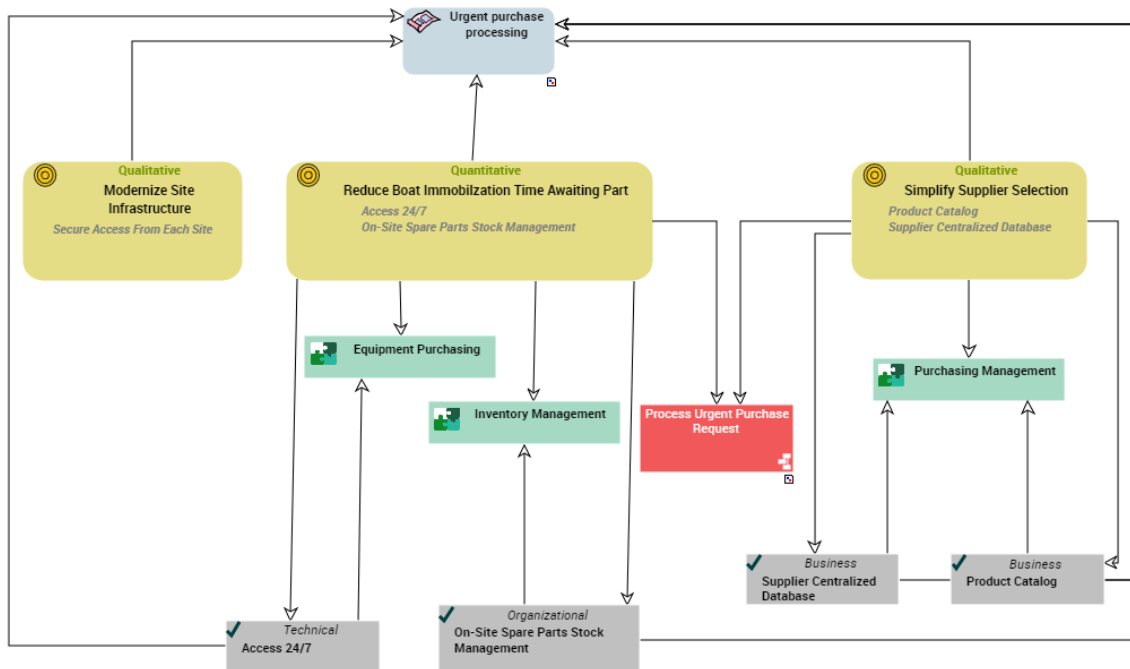
In our example, the development of an application supporting the processing of urgent purchase requests has several objectives:

From a functional viewpoint, this application must ensure reduction in boat immobilization time due to a spare part requirement. Purchase request processing must therefore be accessible 24 hours a day, since company representation offices are spread over several continents, and 7 days a week since the company activity is leisure and tourism.

To reduce delays in the search for suppliers of spare parts already referenced or not, the purchasing department has decided to set up a product catalog and a centralized supplier database.

The project also meets a technical objective, which is to modernize and improve security of communications between company headquarters and the representation offices.

The objective and requirement diagram of a project is used to represent and classify the elements listed.



For more details on managing objectives and requirements, see ["Defining the Objectives and Requirements of a Project \(AD\)", page 41.](#)

## Building Functional Specifications

After defining the project scope, objectives and requirements, this step consists of describing the functionalities expected of the future system by the different categories of users.

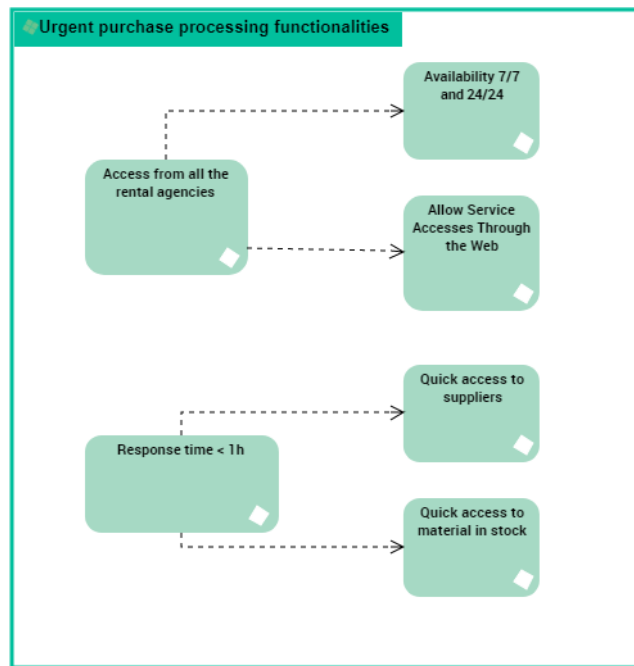
### Describing the map for the expected functionalities of the project

A functionality map is a set of functionalities with their dependencies that, jointly, define the scope of a hardware or software architecture.

A description of the *functionality map* is used to draw up the list of expected functionalities of the project and associate each of them with an objective or requirement, on the one hand, and an application element, on the other hand.

- For more information on managing the expected functionalities of a project, see ["Describing the Functionalities of a Project", page 52.](#)

In our example, a functionalities map is presented in the form of the following diagram.



The expected functionalities of the project for processing urgent purchases are relative to the response times and to the availability of the application from the rental agencies. The main functionalities are divided into sub-functionalities.

## Describing the application environments of the project

An application environment presents the use context of the applications of a project. This describes the interactions between the org-units involved and the internal and external applications of the projects that ensure the project functionalities.

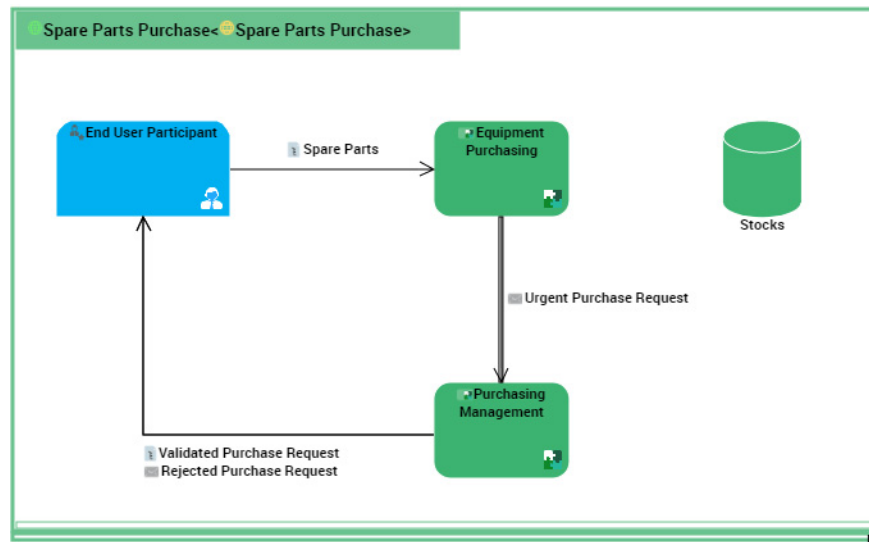
A number of diagram types are offered to describe an *application environment*:

- The flow scenario diagram for application environments is used to describe the flows exchanged between an application and its environment in a given context.
- The application environment diagram is used to describe the interactions between an application and its described environment: its users, the applications and the external users.
- The use case diagram for application environments is used to describe the use cases of the application environment in UML format.
  - A single application can be associated with more than one *application environment* to represent a number of use contexts of the application.

## Flow scenario diagram example

A flow scenario diagram can be built for an application environment, an application, an application service or a micro-service.

- For more details on building a flow scenario diagram, see ["Describing a scenario of flows", page 174.](#)



This flow scenario diagram describes the "Spare Parts Purchasing" application.

The "spare part" request is sent by the local Purchasing assistant to the "Equipment Purchasing" application.

The "Equipment Purchasing" application checks the stock. If the part is not in stock, an "urgent purchasing request is sent" to the "Purchasing Management" application.

The "Purchasing Management" application validates or refuses the request.

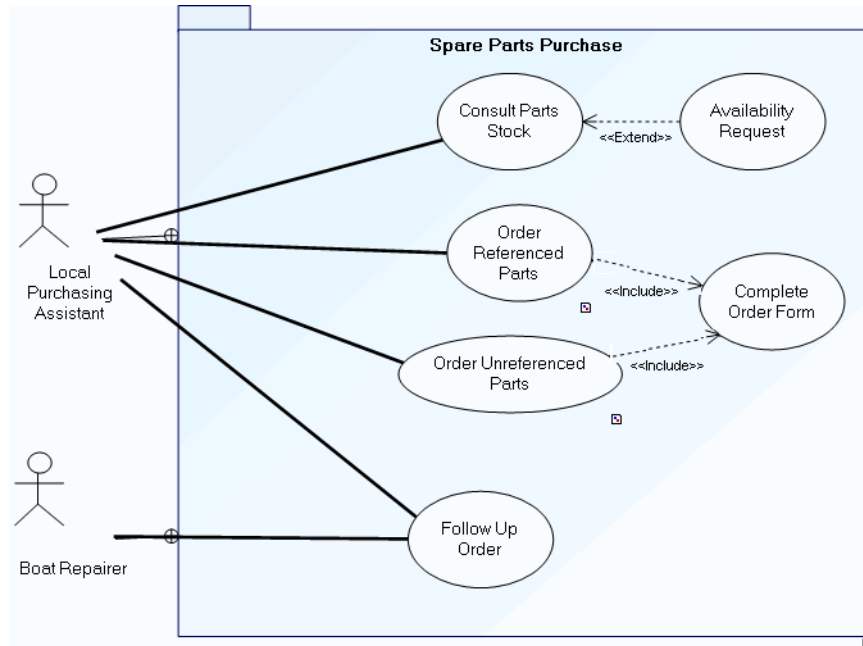
## Example of a Use Case Diagram

You can also use a *use case* diagram to describe the interactions between an application element and the org-units in the specific context.

- ) A use case is a series of actions leading to an observable result for a given actor. Scenarios illustrate use cases for example.

The use case diagrams can be built for the application environments, application services and micro-services.

- For more details on creating a use case, see *"Describing a Use Case of a Project"*, page 61.



The system is used to consult parts in stock and to order new spare parts.

Consultation of parts in stock is carried out by the local on-site purchasing assistant. Following consultation, the assistant can make an availability request.

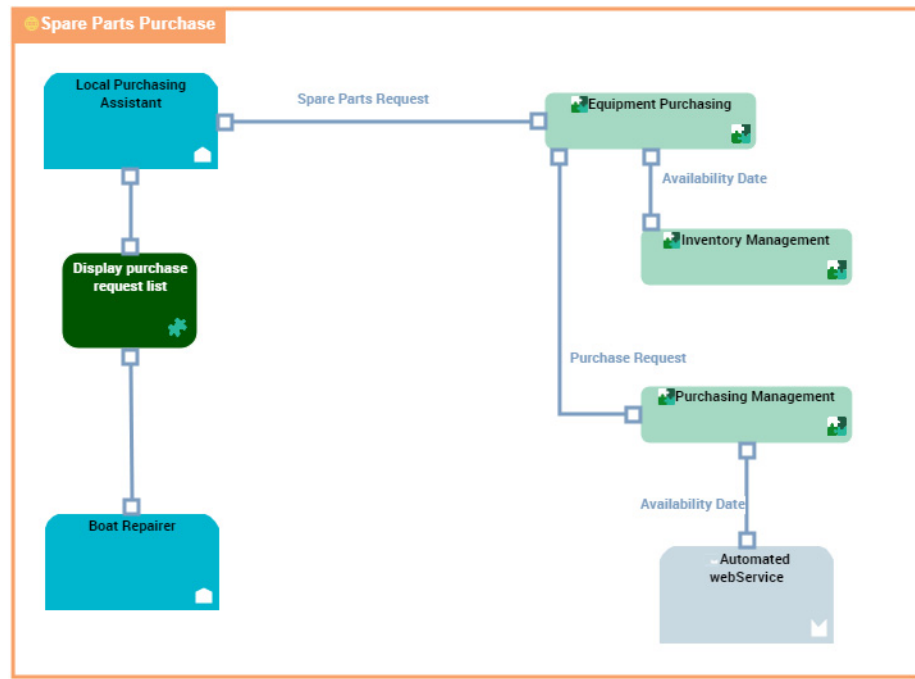
Two order types are possible, one for parts already referenced, the other for parts as yet unreferenced. In both cases, an order form should be completed.

Order follow-up is assured by the local purchasing assistant and the boat repairer.

### Application environment diagram example

- For more details on building an application environment diagram, see *"Describing an Application Environment"*, page 57.

The following diagram describes the application environment corresponding to the processing of spare parts purchasing.



*"Spare Parts Purchase" application environment diagram*

The requests for spare parts are formulated by the persons concerned with boat repairs and these requests are processed by the local purchasing assistants.

Consultation of parts in stock is carried out by the local on-site purchasing assistant. Following consultation, the assistant can make an availability request.

Two order types are possible, one for parts already referenced, the other for parts as yet unreferenced. In both cases, a request for availability is put forth.

Order follow-up is ensured by the local purchasing assistant and the boat repairer.

For more details on functional specifications, see ["Describing the Functional Specifications of a Project"](#), page 51.

---

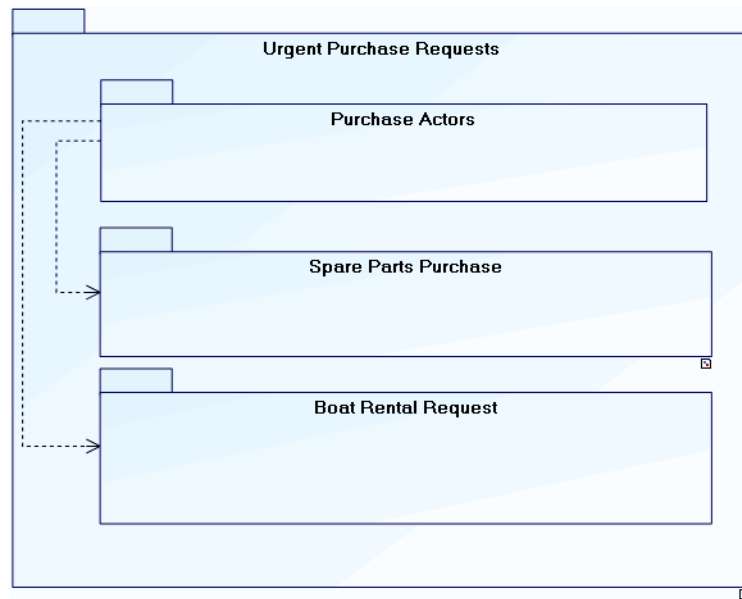
## Describing the Project Data

The purpose of this step is to describe the *data domains* to deduce the database structures that will be implemented by the project.

To successfully complete this step, perform the following tasks:

## Specifying packages

The package allows you to classify elements referenced in a project. You can create sub-packages in a package to classify objects in finer detail, for example actors of a project.



Urgent purchase requests are provided to process purchase of spare parts and boat rental requests. In both of these cases, users are actors of the purchasing domain.

## Specifying logical data areas

A logical data area is used to define a logical data structure made up of classes and data views.

A logical data domain is owned by a package and can reference objects held in other packages.

During integration, a corresponding physical structure can be defined via a physical data area. It is made up of tables and table views.

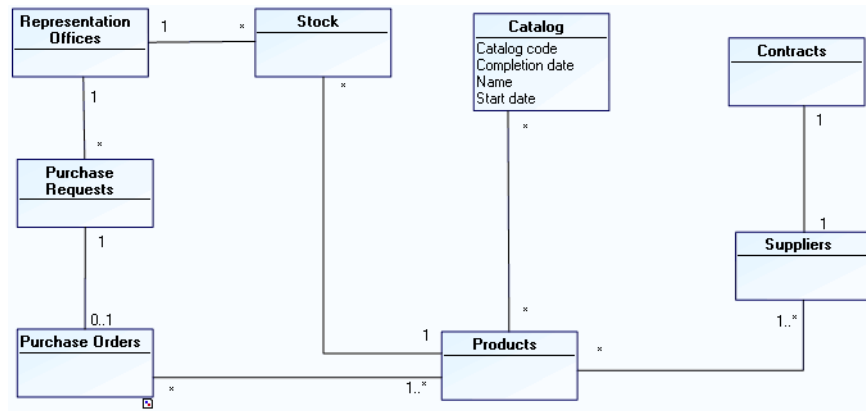
The following logical data area diagram represents a data structure relative to the Orders.

## Specifying relational data areas

A relational data area is used to position the databases in the database management logical model.

The relational data model of the "Purchase Request Automation" project is presented below.

The application manages purchase requests, orders and



product stock levels in each of the representation offices.

A centralized catalog of products and suppliers is installed.

Contracts with referenced suppliers are also accessible using the application.

For more details on describing data, see ["Describing the Data of a Project", page 65](#).

## Building the System Specifications

After the functional specifications and the description of data used, the technical specifications of the project aim to detail the target architecture of the project.

### Describing the target architecture elements

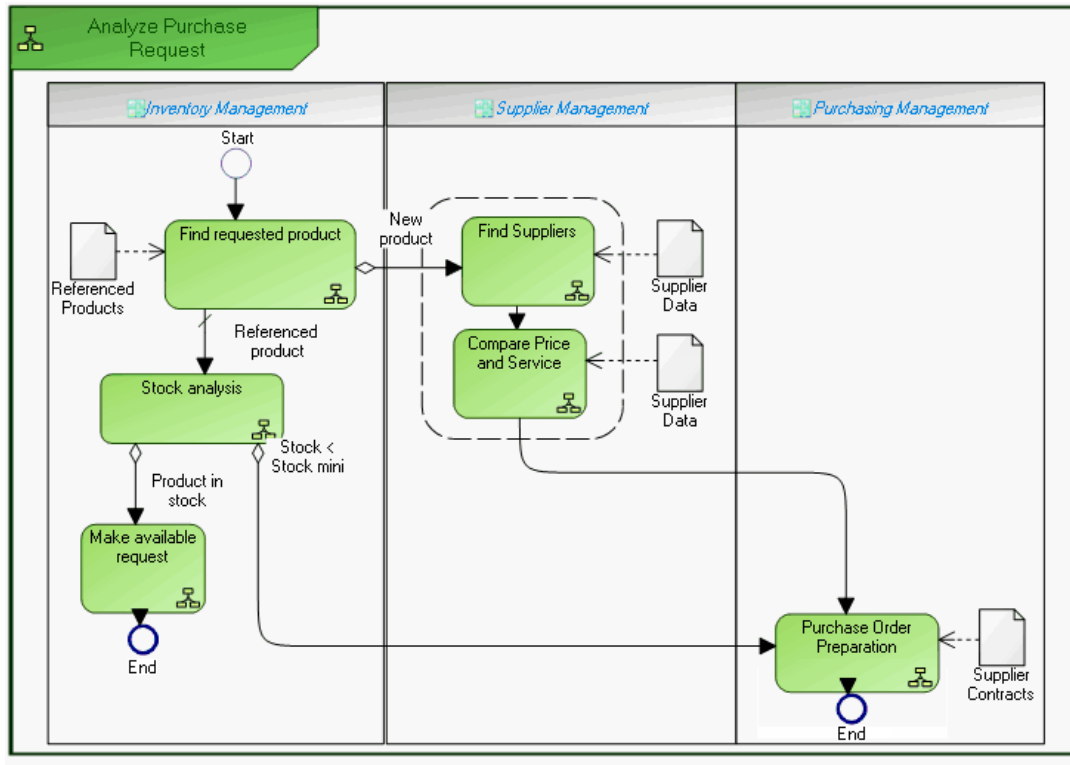
After the functional specifications and the description of data used, the technical specifications of the project aim to describe the target architecture of the project components: *applications, application services, micro-services, databases*.

- ) *An application is a software component that can be deployed and provides users with a set of functionalities.*
- ) *An IT service is a component of an application made available to the end user of the application in the context of his/her work.*
- ) *A micro-service is a software component that can be deployed autonomously, but which does not directly provide an end user service. It can interact with other application services, applications or application systems. This is a deployable software component that uses software technologies. For example: an authentication service, a PDF file printing service.*
- *For more details on describing the target architecture, see the ["Describing IT Services and Micro-services", page 80](#).*

## Application process presentation

An application process is used to describe the detailed sequence of tasks completed on execution of the application.

The following system process diagram proposes a first draft of operation expected of the new application.



A product search is carried out from the referenced products repository.

If the product is referenced, stock is analyzed.

If stock is sufficient, a "Make available" request is activated and the process ends.

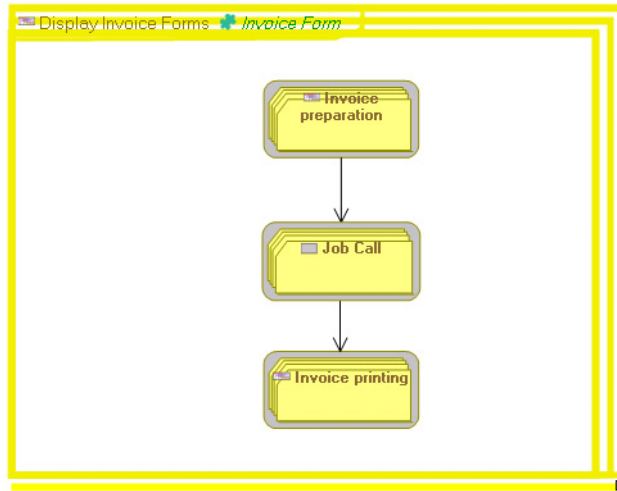
If stock is less than minimum stock, a purchase order is sent to the supplier.

If the product is new, search for a supplier and comparative study of prices is carried out. An order is then sent and the process ends.

- For more details on describing application processes, see ["Describing Application Processes", page 82.](#)

## Describing batch processing

The sequencing of automated processes can be described in a **batch planning structure diagram**.

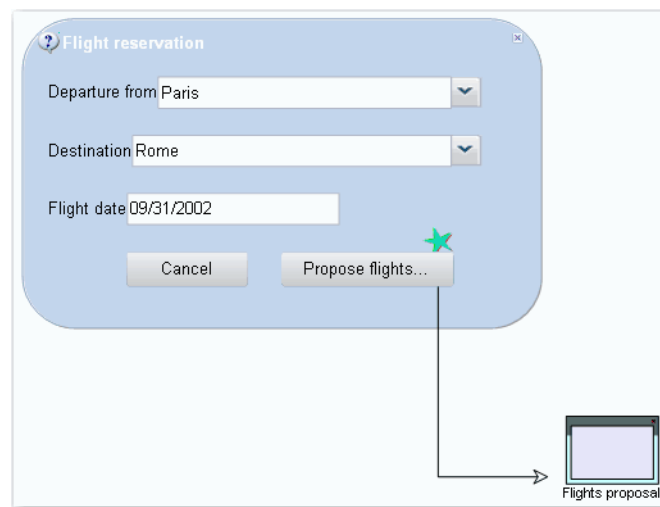


- For more details on describing batch processing, see ["Describing Batch Processing", page 89.](#)

## Describing the list of services and interfaces

It is possible to describe interfaces connecting services or operations with the exterior. This description is carried out in a user interface diagram.

An interface diagram is used to describe the planned interfaces.



- For more details on describing interfaces, see ["Defining User Interfaces", page 94.](#)

For more details on technical specifications, see ["Describing the Technical Specifications of a Project", page 77](#).

# CONNECTION TO THE HOPEX APPLICATION DESIGN DESKTOP

- **HOPEX Application Design** is mainly intended for web users. Desktops described in this guide are accessible only to Web desktop users.

---

## Prerequisite: Importing the Primitive Types

To access primitive types in **HOPEX Application Design**, the administrator must import the "Information Architecture" solution pack.

### ***Preparing the import***

Files to be imported are delivered in compressed files that you must decompress before importing them into a repository.

1. In the folder in which **HOPEX** is installed, open the **Utilities** folder, then the **Solutions Pack** folder.
2. Double-click the **Information Architecture.exe** file.
3. Extract the Contents of the file.

### ***Importing the solution pack***

To import the solution pack:

1. Launch "Administration.exe" and connect as a user with data administration rights.
  - The "System" identifier enables connection with the "Administrator" user. This user is created by default, with repository administration rights. It has no profile (it has all rights) and no password is assigned at installation.
2. Select the environment then the repository on which you want to work.
3. Right-click the repository and select **Object Management > Import a Solution Pack**.  
A dialog box with a list of solution packs appears.
4. Select **Information Architecture - DataType** and **Information Architecture - XsdType** and click **OK**.
5. Exit the Administration application.

---

## Connecting to the solution

To connect to **HOPEX Application Design**, see **HOPEX Common Features**, "HOPEX Web Front-End Desktop".

---

## The HOPEX Application Design desktop

The menus and commands available in **HOPEX Application Design** depend on the profile with which you are connected.

- For more details on using the Web platform for HOPEX solutions, see the **HOPEX Common Features** guide.
- 

### Presentation of space common to all roles

All users have access to the **HOPEX Application Design** desktop and access to the following panes:

- **Home, Dashboards** and **List of Tasks** that are common to all users of **HOPEX** solutions.
- **Ideation**: refers to the management process for ideas that become possible project demands.
- **Reports**: accesses all reports, improving understanding of terms and their use.

### Presenting the application designer desktop

In addition to the panes offered in standard mode to all **HOPEX Application Design** desktop users, the application designer has access to the following panes:

#### **The Environment pane**

The **Environment** pane provides access to the following menus:

- **Organization**, to access the list of org-units and repository processes.
- **Standard navigation**, to build the Enterprises, allocate users to the different steps.
- **Projects**, to access the list of projects and create new ones.

- For more details on project creation, see ["Creating an Application Design Project"](#), page 31.

#### **The Current Design Project pane**

By default, on connection, this pane is not available.

Before defining a current project, confirm that you have created an "Application Design" type environment in the properties of the project in question. See ["Assigning a Working Environment to a Project"](#), page 30.

To define a current project:

1. Click **Main Menu**.
2. Select **Change Work Environment** and select the previously created project concerned.

The **Current Design Project** pane integrates the method embedded in the **HOPEX Application Design** solution and it provides access to the following menus:

- **Current Project Definition**, to describe the project scope, its structure and the stakeholders.
  - For more details on project creation, see ["Describing an AD Project and its Application Environment"](#), page 29.
- **Identification of Needs**, to describe the objectives and requirements of the project.
  - For more details on describing the project objectives and requirements, see ["Defining the Objectives and Requirements of a Project \(AD\)"](#), page 41.
- **Functionalities and Rules Design**, to build the functional specifications of the project.
  - For more information on the functional specification tools, see ["Defining the Objectives and Requirements of a Project \(AD\)"](#), page 41.
- **Data Scope**, to describe the packages and the logical and relational data of the project to define the database.
  - For more details on project data, see ["Describing the Data of a Project"](#), page 65.
- **Application Architecture Design**, to describe the technical elements of the project such as the applications, the IT services, the micro-services, the batch processes and the interfaces.
  - For more information on technical specification tools, see ["Describing the Technical Specifications of a Project"](#), page 77.
- **Validation & Verification**, this facility is based on workflows that are used to manage project progress.

See also ["Creating an Application Design Project"](#), page 31.

### **The Application Design Resource pane**

The **Application Design Resource** pane provides access to the objects in the repository referenced if applicable in the project:

- **Process**, see the guide **HOPEX Business Process Analysis**.
- **Applications**, see ["Describing an Application with HOPEX Application Design"](#), page 78.
- **Service Contracts & Catalogs**, see ["Describing data exchanges"](#), page 185.
- **Application Processes, Batches and Interface**, see ["Describing Application Processes"](#), page 82.
- **Technologies**, see the **HOPEX IT Architecture** guide.
- **Technological Stacks**, see the **HOPEX IT Architecture** guide.
- **UML Implementation**, see ["About UML implementation"](#).
- **Batch Processing**, see ["Describing Batch Processing"](#), page 89.
- **Technical Architectures**, see ["Describing IT Services and Micro-services"](#), page 80.

### **The Collaboration pane**

The **Collaboration** pane provides access to the workflow management features.

## Presenting the desktop of the functional administrator for application design

The **HOPEX Application Design** functional administrator for application design has the application panes offered to the application designer.

The application design functional administrator also has the **Administration** pane that gives him access to user rights management.

- For more details on administrator tasks, see **HOPEX Power Supervisor** administration guide.

## Presentation of the application owner desktop

In addition to the panes offered in standard mode to all **HOPEX Application Design** desktop users, the application owner has access to the following panes:

The **Application Design Resource** pane which provides access to the objects in the repository referenced if applicable in the project:

- **Process**, see the guide **HOPEX Business Process Analysis**.
- **Applications**, see ["Describing an Application with HOPEX Application Design"](#), page 78.
- **Service Contracts & Catalogs**, see ["Describing data exchanges"](#), page 185.
- **Application Processes, Batches and Interface**, see ["Describing Application Processes"](#), page 82.
- **Technologies**, see the **HOPEX IT Architecture** guide.
- **Technological Stacks**, see the **HOPEX IT Architecture** guide.
- **UML Implementation**, see ["About UML implementation"](#).
- **Batch Processing**, see ["Describing Batch Processing"](#), page 89.
- **Technical Architectures**, see ["Describing IT Services and Micro-services"](#), page 80.

The **Application Design Resource** pane provides access to the objects in the repository referenced if applicable in the project:

- **Process**, see the guide **HOPEX Business Process Analysis**.
- **Applications**, see ["Describing an Application with HOPEX Application Design"](#), page 78.
- **Service Contracts & Catalogs**, see ["Describing data exchanges"](#), page 185.
- **Application Processes, Batches and Interface**, see ["Describing Application Processes"](#), page 82.
- **Technologies**, see the **HOPEX IT Architecture** guide.
- **Technological Stacks**, see the **HOPEX IT Architecture** guide.
- **UML Implementation**, see ["About UML implementation"](#).
- **Batch Processing**, see ["Describing Batch Processing"](#), page 89.
- **Technical Architectures**, see ["Describing IT Services and Micro-services"](#), page 80.

## Presentation of the desktop of the application design contributor

With **HOPEX Application Design**, the application design contributor has the same panes as the application owner, see ["Presentation of the application owner desktop"](#), page 23.

## Presentation of the application designer desktop

In addition to the panes offered in standard mode to all user of the **HOPEX Application Design** desktop, the UML designer has access to the **Application Design Resource** which provides access to the objects of the repository referenced if applicable in the project:

- **Process**, see the guide **HOPEX Business Process Analysis**.
- **Applications**, see ["Describing an Application with HOPEX Application Design"](#), page 78.
- **Service Contracts & Catalogs**, see ["Describing data exchanges"](#), page 185.
- **Application Processes, Batches and Interface**, see ["Describing Application Processes"](#), page 82.
- **Technologies**, see the **HOPEX IT Architecture** guide.
- **Technological Stacks**, see the **HOPEX IT Architecture** guide.
- **UML Implementation**, see ["About UML implementation"](#).
- **Batch Processing**, see ["Describing Batch Processing"](#), page 89.
- **Technical Architectures**, see ["Describing IT Services and Micro-services"](#), page 80.

# ABOUT THIS GUIDE

This guide describes how to use **HOPEX Application Design** to develop the specifications of an IT project.

---

## Guide Structure

The **HOPEX Application Design** guide comprises the following chapters:

- ["Creating an Application Design Project", page 31](#); describes the functionalities offered by **HOPEX Application Design** to a functional administrator to prepare a project.
- ["Describing an AD Project and its Application Environment", page 29](#): explains how to describe the stakeholders of a project, the applications impacted as well as any breaking down of a project into sub-projects.
- ["Defining the Objectives and Requirements of a Project \(AD\)", page 41](#): explains how to specify the objectives and requirements covered by a project and how to connect these to the project deliverables.
- ["Describing the Functional Specifications of a Project", page 51](#), explains how to describe the expected functionalities of a project as well their logical implementation on the applications in the scope.
- ["Describing the Data of a Project", page 65](#), explains how to describe the data used in the project and its use in the applications and services of a project.
- ["Describing the Technical Specifications of a Project", page 77](#) explains how to model IT processes implemented in the different use cases of an application or service. It describes the interactions used to represent, supplement or validate the services architecture. The interfaces connecting the services or operations with the exterior are detailed.

- The reports offered by **HOPEX Application Design** to assist users in each step of the project are presented in the chapters relating to each of the steps.

---

## Additional Resources

This guide is supplemented by:

- the **HOPEX Common Features** guide describes the Web interface and tools specific to **HOPEX** solutions.
  - *It can be useful to consult this guide for a general presentation of the interface.*
- The **HOPEX Information Architecture** guide which describes data models.
- More advanced technical functions are described in the **HOPEX Power Studio** guide.
- the **HOPEX Power Supervisor** administration guide.

---

## Conventions used in the guide

- *Remark on the preceding points.*
- ) *Definition of terms used.*
- M *A tip that may simplify things.*
- . *Compatibility with previous versions.*
- P **Things you must not do.**



**Very important remark to avoid errors during an operation.**

Commands are presented as seen here: **File > Open.**

Names of products and technical modules are presented in bold as seen here: **HOPEX.**

# **Architecture Specification**





# DESCRIBING AN AD PROJECT AND ITS APPLICATION ENVIRONMENT



**HOPEX Application Design** relies on a methodological approach based on the concept of specification project. The different steps of a project are guided by a validation process supported by a workflow.

First of all you should define the application scope of the project, how to break down your project if appropriate into sub-projects and the organization of participants in the project.

The following points are covered here:

- 6 ["Creating an Application Design Project", page 30](#)
- 6 ["Describing the Deliverables of an AD Project", page 32,](#)
- 6 ["Describing the Organization of Project Participants", page 34,](#)
- 6 ["Describing the Sub-Projects of an AD Project", page 37,](#)
- 6 ["Generating the Pre-Closing Analysis Report", page 39](#)

# CREATING AN APPLICATION DESIGN PROJECT

---

## Creating the Project

An AD project is an Enterprise Architecture (EA) project of type "Application Design" dedicated to the definition of software specifications. It is associated with a validation workflow.

) *An Enterprise Architecture project is a part of a system whose study is entrusted to a single team, which transforms a system or part of a system to achieve a given objective.*

To create an **AD project**:

1. In the **Home** navigation pane, click the **New AD Project** tile.  
An enterprise architecture project of **Application Design** type is created. Its properties page opens in the edit area.  

) *In the **Environment > Projects** navigation pane, you can access the project tree.*
2. In the **Characteristics** page, specify the name of the project and the **Owner** if applicable.
  - *For more details on managing libraries, see the **HOPEX Common Features** guide, "Enterprises and Libraries".*
3. Also specify the project participants. See ["Describing the Organization of Project Participants"](#), page 34.

In the **Working Environment Assignment** page you can view the working environment created for the project.

---

## Assigning a Working Environment to a Project

Managing an application design project involves several steps: identification of needs, scope definition, etc.

With **HOPEX Application Design**, these steps are described in an **Application Design** type **working environment**.

Once the working environment is created on an project; you can display it in the interface. The **Current Design Project** navigation pane shows the various stages of the selected project, with the tools required for each one.

## Creating a working environment for a project

To define a working environment for a project:

1. In the home page, click on **All EA Projects**.
2. Select the project concerned to display its properties.
  - *Click the **Properties** button of the edit area if properties are not displayed.*
3. Select the **Working Environment Assignment** page.

4. Click **New**.
5. Rename if needed the new environment and select the "Application Design" type.
6. Click **OK**.

By default, the various steps of the working environment are visible to all users. You can more precisely define the project participants, who will be able to validate the steps of the project through the project validation workflow. See ["Describing the Organization of Project Participants", page 34](#).

## Displaying the working environment of a project

To display the working environment of a project:

1. Click **Main Menu**.
2. Select **Change Work Environment** and select the project concerned.

See also: ["The Current Design Project pane", page 21](#).

# DESCRIBING THE DELIVERABLES OF AN AD PROJECT

Describing the deliverables of an AD project consists of defining the project scope and the list of applications and services that will be added or modified.

) *A project deliverable defines the result of a project and its impact or its contribution to the architectural solution landscape of the enterprise.*

---

## Prerequisite

You must first define a work environment for a project in progress. See ["The Current Design Project pane", page 21.](#)

---

## Defining the Application Scope of the Project

### Application scope example

The "Equipment Purchasing" application is the main application impacted by the project, in particular for improvement of its access portal.

The "Stock Management" application is used by the "Inventory Management" application to view and update information on stock and in particular, spare parts.

The "Purchasing Management" application uses the "Catalog" database to send a purchase order.

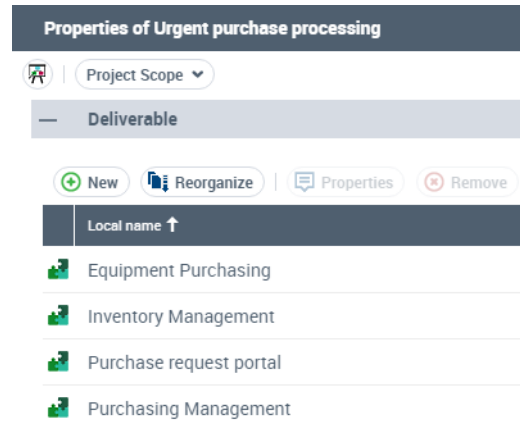
The "Maintenance and Repair Management" and the "Spare Parts Management" applications will also use the "Equipment Purchasing" application portal.

### Describing the project applications

To add an existing application to the list of applications included in the scope of the current project:

1. From the **Current Design Project > Current Project Definition** navigation pane, select **Project Scope**.  
The list of applications impacted by the current project appears in the section **Specified applications**.
2. Click the **New** button.  
The add **Application** project window appears.

3. In the **Name** field, select the application that you want to add and click **OK**.  
The new applications appears in the list of applications impacted by the project.



## Describing the Results Produced by the Project

### Accessing the list of results produced by the current project

To access the list of *results* produced by the current project:

- > From the **Current Design Project > Current Project Definition** navigation pane, select **Hierarchical View of the Project**.  
The list of deliverables expected by the current project appears in the **Deliverables Produced** file.
  - *The applications that make up the project scope are automatically added to the project results.*

### Adding a new project result

To specify that an existing object is a **result** for the current project:

1. From the **Current Design Project > Current Project Definition** navigation pane, select **Project Scope**.  
The list of results expected by the current project appears in the **Other Objects** section.
2. Click the **New** button.  
The add **Project Result** window appears.
3. In the **Object Type** field, select the type of result that you want to add.
4. In the **Name** field, select the object that you want to declare as a result and click **OK**.  
The object appears in the list of project results.

# DESCRIBING THE ORGANIZATION OF PROJECT PARTICIPANTS

The participants of a specification project are *persons (system)* grouped into various *organizations*.

) A person (System) represents a person in the enterprise. This person can be assigned a login and a role (or a profile depending on the connection mode). The login provides access to the HOPEX Application. The role (or the profile) defines the access to product functions and repositories. A system person, if assigned a login, has a specific desktop in each database, and can connect to this desktop from any workstation in a given environment.

) An Enterprise Architecture organization is governing body consolidating different persons and organizations.

A single person can be involved in different organizations with different *business roles* for each.

) A business role is used to assign a task to a person (example: describe an application) and where appropriate, for a specific location (example: Paris agency). Business function roles are assigned to persons and a person can have more than one business function role.

- For more details on the business roles delivered with **HOPEX Application Design**, see "[Business Roles of HOPEX Application Design](#)", page 4.

A **Person Membership** is used to connect a *person (system)* to a *business role* in the context of an organization defined for a project.

In the same way, an **Organization Membership** defines an organizational hierarchy by connecting one organization to another in the context of a project.

---

## Prerequisite

You must first define a work environment for a project in progress. See "[The Current Design Project pane](#)", page 21.

---

## Accessing the Organization of Project Participants

To access the list of *EA organizations* of a specification project:

1. Open the **Characteristics** property page of the **project** that interests you.
2. Expand the **Organization** section.  
The list of *AE organizations* of the highest level appears.

To access the description of the organization of the current project:

1. From the **Current Design Project > Current Project Definition** pane, click the **Stakeholders** tile.  
The list of *AE organizations* of the highest level connected to the project appears.
2. Select the organization that interests you.  
The list of **Person Memberships** as well as the list of **Organization Memberships** that make up the organization described appears.

---

## Creating an Organization

To create an **Organization** for the current project:

1. From the **Current Design Project > Current Project Definition** pane, click the **Stakeholders** tile.  
The list of *AE organizations* of the highest level connected to the project appears.
2. Click the **New** button.  
The window for creating an **Organization** appears.
3. Specify the name of the organization that you want to create and click **OK**.  
The new organization appears in the list of project organizations.

## Connecting a person to an organization

To create a **Person Membership** in an organization connected to the current project:

1. From the **Current Design Project > Current Project Definition** pane, click the **Stakeholders** tile.  
The list of *AE organizations* of the highest level connected to the project appears.
2. Expand the **Organization** section and select the organization in which the person participate.  
The list of **Person Memberships** as well as the list of **Organization Memberships** that make up the membership described appears.
3. In the **Person Membership** section, click **New**.  
A **Person Membership** creation window opens.
4. In the **Member** field, click **Connect Person (system)**.  
A dialog box for connecting system persons appears.
5. Find and select the person associated with the user that interests you and click **Connect**.
6. In the **Role in the Organization**, click **Connect Business Role**.  
A connection window opens.
7. Find and select the business role that interests you and click **Connect**.
8. Click **Connect**.  
The new person membership appears in the list with the name of the user and their role in the described organization.

## Reusing an existing organization in a project

To reuse an **Organization** from a project:

1. Open the **Characteristics** property pages of the **project** that interests you.
2. Expand the **Organization** section and select the organization in which the person participate.  
The list of **Person Memberships** as well as the list of **Organization Memberships** that make up the membership described appears.
3. In the **Organization Membership** section, click **New**.  
An **Organization Membership** creation window opens.
4. In the **Creation Mode** section, click **Reuse an Organization Membership**.  
A **Sub-organization** section appears with the list of existing organizations.
5. Select the organization that interests you and click **OK**.  
The new organization membership appears in the list.

## DESCRIBING THE SUB-PROJECTS OF AN AD PROJECT

An AD project can be split into a number of sub-projects.

You can also describe the dependency link between the different sub-projects.

---

### Prerequisite

You must first define a work environment for a project in progress. See ["The Current Design Project pane"](#), page 21.

---

### Accessing the Breakdown of a Project into Sub-Projects

To access the list of sub-projects of an AD project:

1. Open the **Sub-projects** properties page of the **project** that interests you.
2. Expand the **Sub-project** section.  
The list of sub-projects of the highest level appears.
  - *The **Sub-projects** properties page of a project is used to access the list of sub-projects of the project described.*

---

### Connecting a Sub-Project to the Current Project

The projects are created by an **Application Design Functional Administrator** or by an **Application Designer**.

- *For more details on these profiles, see ["The functional administrator for application design"](#), page 3 and ["The application designer"](#), page 3.*
- *For more details on project creation with **HOPEX Application Design**, see ["Creating an Application Design Project"](#), page 30.*

Connecting an existing project to the current project:

1. In the **Current Design Project > Current Project Definition** pane, click **Sub-projects**.  
The list of sub-projects of the highest level connected to the project appears.
2. In the **Sub-projects** section, click **Connect**.  
A window for connecting an AE project appears.
3. Find and select the project that interests you and click **Connect**.  
The project appears in the list of sub-projects of the project described.
  - *To access the hierarchy of projects in the **Current Design Project > Current Project Definition** pane, click **Hierarchy of Current Projects** and expand the **Sub-project** folder of the projects that appear in the tree.*

---

## Displaying the Properties of the Current Project

You can specify the list of projects on which the current project depends. For each of these projects, you can also specify which *result* of the current project is impacted by this dependency.

### Creating a project dependency

To specify that the current project depends on another project:

1. In the **Current Design Project > Current Project Definition** pane, click **Sub-projects**.  
The list of sub-projects of the highest level connected to the project appears.
2. In the **Project Dependencies** section, click **New**.  
The creation window for the **Project Dependency** opens.
3. In the **Project Expected** field, find and select the project that interests you and click **Add**.  
The *project dependency* is created.

### Specifying the result of a project dependency

To specify the project deliverables impacted by a project dependency:

1. Open the **Characteristics** property pages of the dependency that interests you.
2. In the **Deliverables Concerned** section, click **Connect**.  
The connection window for the **Project Result** opens.
3. Select the project result that interests you and click **Connect**.  
The result appears in the list of deliverables concerned.

### Specifying the requirements of a project dependency

) *A requirement is a need or expectation explicitly expressed, imposed as a constraint to be respected within the context of a project. This project can be a certification project or an organizational project or an information system project.*

- *For more details on the requirements of a project specification, see "Creating Projects", page 31.*

To specify the project requirements imposed by a project dependency:

1. Open the **Objectives and Requirements** property pages of the dependency that interests you.
2. In the **Requirements Imposed** folder and click **Connect**.  
A **Requirement** connection window opens.
3. Select the requirement that interests you and click **Connection**.  
The requirement appears in the list of requirements imposed.

## GENERATING THE PRE-CLOSING ANALYSIS REPORT

The Pre-Closing Analysis Report offers in a Word document a description of the current project. It includes the following elements:

- The project deliverables
- The stakeholders
- The objectives and requirements of the project, as well as the requirements associated with its deliverables
- The processes impacted by the project
- The project dependencies
- The project workflow steps

To generate a Pre-Closing Analysis Report on the current project:

1. In the **Current Design Project > Current Project Definition** pane, click **Pre-Closing Analysis Report**.
  - You must first define a work environment for the project. See ["The Current Design Project pane", page 21](#).The list of MS Word reports connected to the project in progress appears.
2. In the edit area, click the **New** button.  
The report (MS Word) creation dialog box opens.
3. Enter the name of the report.
4. In the **Report type (MS Word)**, select **Application Design - Preliminary Study**.
5. Click **OK**.  
The report appears in the list.
6. Click the icon of the document and select **Documentation > Open**.



# DEFINING THE OBJECTIVES AND REQUIREMENTS OF A PROJECT



In an application design project, once the scope is described, the second step aims to define the objectives and requirements of the project.

Requirements are then related to the components concerned, whether technical or organizational.

The following points are covered here:

- 6 ["Establishing the Objectives and Requirements for a Project", page 42,](#)
- 6 ["Using an Objective and Requirement Diagram", page 45,](#)
- 6 ["Generating the Requirement Analysis Report", page 50.](#)

## ESTABLISHING THE OBJECTIVES AND REQUIREMENTS FOR A PROJECT

The first task in creating a project, after creating and initializing the *project* is to inventory the *objectives* involved as well as the *requirements* imposed, and to connect these to the main organizational and technical components that make up the project scope.

- ) *An Enterprise Architecture project is a part of a system whose study is entrusted to a single team, which transforms a system or part of a system to achieve a given objective.*
- ) *An objective is a goal that a company/organization wants to achieve, or is the target set by a process or an operation. An objective allows you to highlight the features in a process or operation that require improvement.*
- ) *A requirement is a need or expectation explicitly expressed, imposed as a constraint to be respected within the context of a project. This project can be a certification project or an organizational project or an information system project.*

This paragraph describes:

- ["Managing the List of Objectives for a Project", page 42,](#)
- ["Managing the List of Requirements for a Project", page 43.](#)

---

### Managing the List of Objectives for a Project

#### Accessing the Objectives of a Project

To access the list of *objectives* of a specification project:

1. Open the **Requirements Capture** properties page of the project that interests you.
2. Expand the **Objectives** section.  
Note that two *objective* lists are offered:
  - the **Assigned objectives**, that represent the planned objectives within the framework of the project,
  - the **Achieved objectives**, that represent the objectives already reached.

#### Creating a new objective the current project

To create an *objective* within the context of a project in progress:

1. In the **Current Design Project > Identification of Needs** pane, select **Objectives**.  
A list of *objectives* connected to the project in progress appears.
2. Select the list of **Assigned objectives** and click **New**.  
The window for creating an objective appears.

3. Enter the name of the objective and click **OK**.  
The objective appears in the list.

## Viewing the properties of an objective of the current project

To view the properties of an *objective* of the project in progress:

1. In the **Current Design Project > Identification of Needs** pane, select **Objectives**.  
A list of *objectives* connected to the project in progress appears.
2. Open the properties page of the objective that interests you.

The **Characteristics** property page of the of an *objectives* provides access to:

- its **Name**,
- its **Owner**, by default during creation of the objective, the current library.
- the text of its **Description**.
- the **Objective type**: Qualitative, Quantitative or Other.
- the list of its **sub-objectives**.

With **HOPEX Application Design**, an objective is described in the following pages:

- the **Scope** provides access to the list of components constrained by the objective.
- the **Contributors** provides access to the list of components that contribute to achieving the objective.
  - For more details on how to connect an object to the elements of the scope of a project, see ["Connecting Technical and Organizational Components to Requirements"](#), page 47.
- the **Requirements** page provides access to the list of requirements connected to the objective.
  - ) A requirement is a need or expectation explicitly expressed, imposed as a constraint to be respected within the context of a project. This project can be a certification project or an organizational project or an information system project.
  - For more details on how to connect a requirement objective, see ["Creating a Project Objective and Requirement Diagram"](#), page 46.

---

## Managing the List of Requirements for a Project

### Accessing the Requirements of a Project

To access the list of *requirements* of a specification project:

1. Open the **Requirements Capture** properties page of the project that interests you.
2. Expand the **Requirements** section.  
The list of *requirements* appears.

## Creating a new requirement for the current project

To create a *requirement* within the context of a project in progress:

1. In the **Current Design Project > Identification of Needs** pane, click **Requirements**.  
A list of *requirements* connected to the project in progress appears.
2. Click the **New** button.  
The window for creating a requirement appears.
3. Enter the name of the requirement and click **OK**.  
The requirement appears in the list.

## Viewing the properties of a requirement of the project in progress.

To view the properties of a *requirement* of the project in progress:

1. In the **Current Design Project > Identification of Needs** pane, click **Requirements**.  
A list of *requirements* connected to the project in progress appears.
2. Open the properties page of the requirement that interests you.

The **Characteristics** properties page of a *requirement* provides access to:

- its **Name**,
- its **Owner**, by default during creation of the requirement, the current library.
- the text of its **Description**.
- its **Requirement Type**: Technical, System, Organizational, etc.
- the list of its **sub-requirements**.

With **HOPEX Application Design**, a requirement is described in the following pages:

- the **Scope** and **Contributors** pages used to access the list of components that are, respectively, constrained by the requirement or that contribute to its fulfillment.
  - For more details on how to connect a requirement to the elements of the scope of a project, see ["Connecting an application to a requirement"](#), page 48.
- the **Objectives** page provides access to the list of objectives connected to the described requirement.
  - ) An objective is a goal that a company/organization wants to achieve, or is the target set by a process or an operation. An objective allows you to highlight the features in a process or operation that require improvement.
  - For more details on how to connect a requirement to objectives, see ["Associating an objective to a requirement in an objective and requirement diagram"](#), page 47.

## USING AN OBJECTIVE AND REQUIREMENT DIAGRAM

With **HOPEX Application Design**, you can also create your *objectives* and *requirements* diagram from a specification project.

) An objective is a goal that a company/organization wants to achieve, or is the target set by a process or an operation. An objective allows you to highlight the features in a process or operation that require improvement.

) A requirement is a need or expectation explicitly expressed, imposed as a constraint to be respected within the context of a project. This project can be a certification project or an organizational project or an information system project.

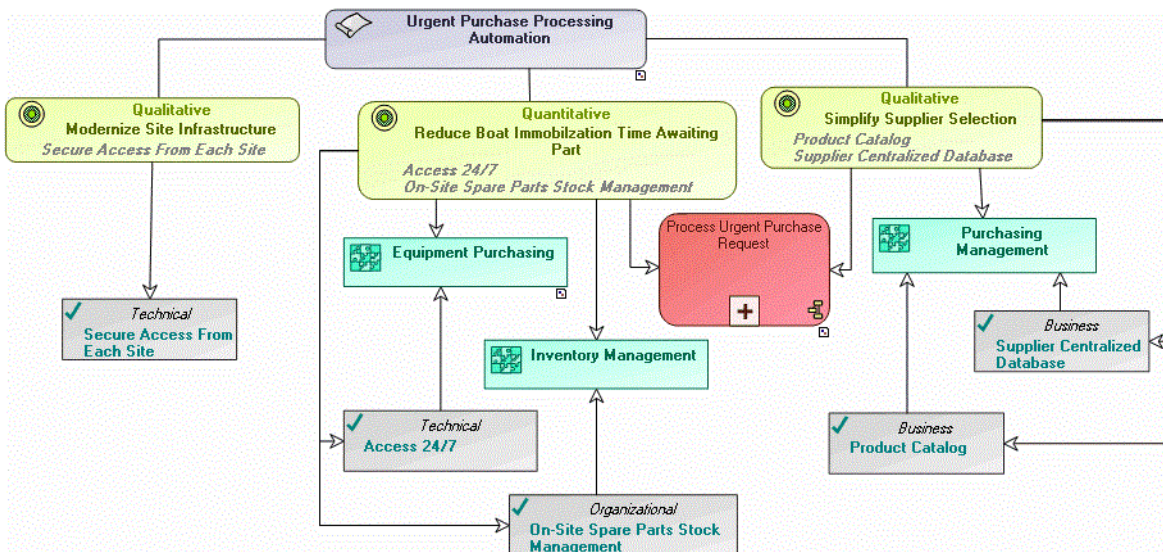
This paragraph describes:

- "Viewing an Objective and Requirement Diagram", page 45,
- "Creating a Project Objective and Requirement Diagram", page 46,
- "Connecting Technical and Organizational Components to Requirements", page 47.

### Viewing an Objective and Requirement Diagram

To view the project objective and requirement diagram of a project:

1. Right-click the project to open its pop-up menu.
2. Double-click **Project Objective and Requirement Diagram**.  
The diagram opens.



In this example, the project meets the quantitative objective of reduction in boat immobilization time, and the

qualitative objectives of improvement in the communication system on the one hand and the management of suppliers on the other.

These objectives are linked to technical or system requirements, such as 24/24 and 7/7 access to the process urgent purchase request application.

The urgent purchase processing process must respect company objectives related to reducing boat immobilization time and simplifying access to suppliers and products.

In addition, the equipment purchasing application must be accessible 24/7. The inventory management application must integrate the requirements for setting up product catalog and supplier database. Finally, the inventory management application must integrate the on-site spare parts management requirement.

- *To show application or organizational components in an objective and requirement diagram, you must have activated the corresponding view from **View > Views and Details**.*
- *Organizational processes are available with the **HOPEX Business Process Analysis** product.*

---

## Creating a Project Objective and Requirement Diagram

### Creating a project objective and requirement diagram

To create a new project objective and requirement diagram:

1. Right-click your project and select **New > Project Objective and Requirement Diagram**.

An empty diagram appears.

- *For more details on objective and requirement diagrams, see the **HOPEX Common Features** guide, chapter "Objectives and Requirements".*

2. In the navigation tree, click on the project described to drag it to the diagram.

### Adding objectives to the project objective and requirement diagram.

To add an *objective*:

1. Click the **Objective** button in the diagram insert toolbar, then click in the diagram.  
The add objective dialog box appears.
2. In the **Name** field, select **List**.
3. Select the objective that interests you and click **OK**.  
The objective appears in the diagram and in the project navigation tree.



- If the described project is positioned in the diagram, a link between the project and the objective automatically appears.

## Adding requirements to the project objective and requirement diagram.

To add a *requirement*:

1. Click the **Requirement** button in the diagram insert toolbar, then click in the diagram.  
The add requirement dialog box appears.
2. In the **Name** field, select **List**.
3. Select the requirement that interests you and click **OK**.  
The requirement appears in the diagram.
  - If the described project is positioned in the diagram, a link between the project and the requirement automatically appears.

## Associating an objective to a requirement in an objective and requirement diagram

To specify the objective associated with a requirement using a project objective and requirement diagram:

1. Open the diagram
2. Click the Link button.
3. Click the objective and, holding the mouse button down, draw a link to the requirement.  
A link from the objective to the requirement appears in the diagram. The name of the requirement appears within the shape of the objective.
  - The link can also be created from the requirement to the objective.

The objective and requirement properties pages are then updated.

- For more details on the properties of a requirement or an objective, see ["Viewing the properties of an objective of the current project", page 43](#) and ["Viewing the properties of a requirement of the project in progress.", page 44](#).

---

## Connecting Technical and Organizational Components to Requirements

### Reminder of principle

The technical and organizational components to which the objectives and requirements relate can be specified.


For example, the "Purchasing Management" application must respect requirements related to setting up a product catalog and a centralized supplier database.

These links can be created graphically in the project objective and requirement diagram.

## Adding an application in the objective and requirement diagram

To connect an application to a requirement from an objective and requirement diagram, we must first show the applications in the diagram since they are not displayed by default.

To view applications in a project objective and requirement diagram:

1. Click the **Views and Details**  button in the diagram toolbar.
2. Select the **Applications, IT Services, Databases** check box.
3. Click **OK**.  
The corresponding buttons appear in the insert toolbar.

To insert an application in the objective and requirement diagram:


1. In the objects toolbar, click the **Application** button.
2. Click on the diagram.  
The add application dialog box appears.
3. Click the arrow at the right of the **Name** box and select **List** in the drop-down list.  
The list of applications accessible from the current library appears.
4. Select the required application.
5. Click **Add**.  
The application appears in the diagram.

## Connecting an application to a requirement


Connecting an application to a requirement can have two non-exclusive meanings:

- Either the application **contributes** to satisfaction of a requirement,
- Or the application is **constrained** by the requirement.

To graphically specify that an application contributes to satisfaction of a requirement :

1. Click the reel button  in the insert toolbar.
2. Click the application and, holding the mouse button down, draw a link to the requirement.  
The directional link appears as a dotted line from the application to the requirement.

To graphically specify that an application is constrained by satisfaction of a requirement:

1. Click the reel button  in the insert toolbar.
2. Click the requirement and, holding the mouse button down, draw a link to the application.
3. Release the button.  
The directional link appears from the requirement to the application.

The requirement and objective properties pages are then updated.

- For more details on the properties of a requirement or an objective, see ["Viewing the properties of an objective of the current project", page](#)

*43 and "Viewing the properties of a requirement of the project in progress.", page 44.*



# DESCRIBING THE FUNCTIONAL SPECIFICATIONS OF A PROJECT



After describing the project scope, as well as its objectives and requirements, you must specify the functionalities expected of the main components of the project.

At the end of this functional specifications step, company management can decide whether or not to proceed with the project.

- 6 ["Describing the Functionalities of a Project", page 52,](#)
- 6 ["Describing an Application Environment", page 57,](#)
- 6 ["Describing a Use Case of a Project", page 61,](#)
- 6 ["Generating the Functional Analysis Report", page 64.](#)

# DESCRIBING THE FUNCTIONALITIES OF A PROJECT

The **HOPEX Application Design** solution makes a distinction between the *technical functionalities* and the other *functionalities*.

- ) *A functionality is a capability expected from an equipment item (hardware or software) to ensure the operation of a business function or an organization.*
- ) *A technical functionality is a capability expected from an equipment item (hardware or software) to ensure the operation of a technical element or an application.*

---

## Prerequisite

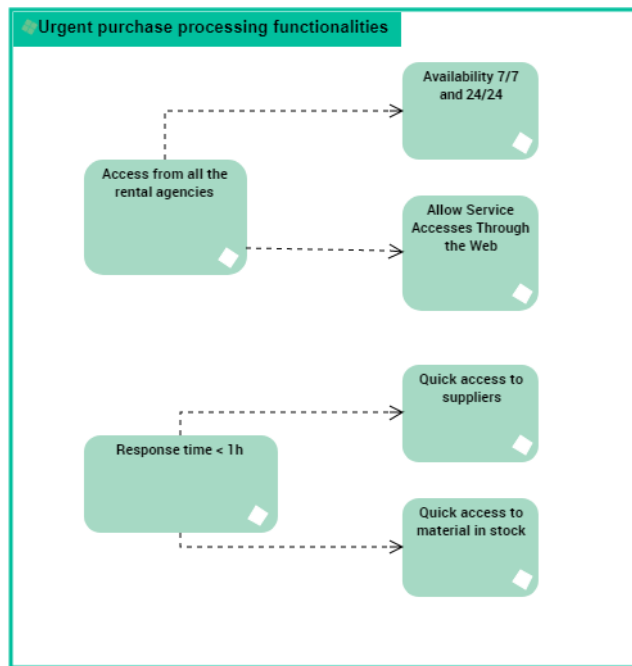
You must first define a work environment for a project in progress. See ["The Current Design Project pane"](#), page 21.

---

## Describing the Functionality Map

A functionality map is a set of functionalities with their dependencies that, jointly, define the scope of a hardware or software architecture.

In our example, a functionalities map is presented in the form of the following diagram.



The expected functionalities of the project for processing urgent purchases are relative to the response times and to the availability of the application from the rental agencies. The main functionalities are divided into sub-functionalities.

## Accessing the list of functionality maps

To access the list of functionality maps:

1. Click the navigation menu, then **Current Design Project**.
2. In the navigation window, select **Functionality and Rules Design**.
3. Click the **List of Functionality Maps** folder and select the **Functionality Maps** tab.  
The list of functionality maps appears in the edit area.

## Creating a functionality map

To create a functionality map diagram:

1. Click **Current Design Projects > Functionalities and Rules Design** in the navigation menu.
2. Click **List of Functionality Maps** and select the **Functionality Maps** tab.
3. Click the **New** button.  
A **Functionality Map** creation window opens.
4. Specify the **Name** of the new functionality map.

5. Click **OK**.  
The new functionality map appears in the list.

## Describing a functionality map

A functionality map is described by a diagram.

To create a functional map diagram:

- > Right-click the functionality map that interests you and select **New > Functionality Map**.

The diagram opens in the edit area. The frame of the functionality map described appears in the diagram.

To add a sub-functionality in a diagram, see ["Creating a functionality component in a functionality map", page 54](#).

To define the dependencies of sub-functionalities, see ["Describing the dependencies between the functionalities using the functionality map", page 54](#)

## Creating a functionality component in a functionality map

The components represented in a functionality map are *functionalities*.

To add a new functionality to a functionality map diagram:

1. In the diagram insertion toolbar, click **Functionality Component**.
2. Click the functionality map frame.  
An add functionality dialog box appears:
3. Enter the name of the functionality you wish to create.
4. Click **OK**.

The new functionality component appears in the diagram.

## Describing the dependencies between the functionalities using the functionality map

A dependency link between one functionality and another is used to specify the elements on which this dependency is based.

For example, the functionality relating to response times depends on the time to access the list of suppliers and on the time to access the availability of parts in stock.

To create dependency links between two functionalities in a functionality map diagram:

1. In the insert toolbar, click **Functionality Dependency**.
2. Click the functionality, and keeping the left mouse button pressed, move the cursor to the functionality used.
3. Release the mouse button.

The link appears in the diagram.

- *A single sub-functionality can have more than one dependency within a single functionality map.*

## The properties of a functionality map

The **Characteristics** properties page of a functionality map provides access to:

- its **Owner**, by default during creation of the functionality map, this is the current project.
- its **Name**,
- the text of its **Description**.

With **HOPEX Application Design**, a functionality map is described by the following pages:

- the **Structure** page, which is used to specify a list of functional components owned and the dependencies between them.
  - For more information on the components of a functionality map, see ["Creating a functionality component in a functionality map"](#), page 54 and ["Describing the dependencies between the functionalities using the functionality map"](#), page 54.

---

## Describing a Functionality

) A functionality is a capability expected from an equipment item (hardware or software) to ensure the operation of a business function or an organization.

### Accessing the list of functionalities

To access the list of functionality maps from the **Current Design Project** navigation pane:

1. Click **Functionalities and Rules Design** in the navigation menu.
2. Click the **List of Functionality Maps** folder and select the **Functionalities** tab.

The list of functionalities appears in the edit area.

### Functionality properties

The **Characteristics** properties page of a functionality provides access to:

- its **Owner**, by default during creation of the functionality map, this is the current project.
- its **Name**,
- the text of its **Description**.

---

## Describing a Technical Functionality

) A technical functionality is a capability expected from an equipment item (hardware or software) to ensure the operation of a technical element or an application.

With **HOPEX Application Design**, the use of technical functionalities and technical functional maps is identical to that of the functionalities and functionality maps.

- For more details on the operation of functionality maps and functionalities, see ["Describing the Functionality Map", page 52](#) et ["Describing a Functionality", page 55](#).

To access the list of *technical functionality maps* from the **technical functionality map** navigation pane:

1. Click **Functionalities and Rules Design** in the navigation menu.
2. Click the **Technical Functionality Maps** folder and select the **Technical Functionality Maps** tab.

The list of technical functionality maps appears in the edit area.

- To access the technical functionality list, select the **Technical Functionality** tab.

## DESCRIBING AN APPLICATION ENVIRONMENT

In **HOPEX Application Design**, the *applications* comprise the link between the business and technical functionalities expected and the architecture elements.

) *An application is a software component that can be deployed and provides users with a set of functionalities.*

In an application environment, you can describe the communications that an application establishes with the other components of the project. Each use context of an application is represented by an *application environment*.

) *An application environment presents the use context of the applications of a project. This describes the interactions between the org-units involved and the internal and external applications of the projects that ensure the project functionalities.*

An application environment can be described by different types of diagrams:

- *flow scenario diagrams* that describe the flows exchanged between the application elements of the environment in different contexts.
  - *For more details on flow scenario diagrams, see "Describing a scenario of flows", page 174.*
- *use case diagrams* that are used to represent a use context of an application element in a specific format.
  - *A flow scenario can also be associated with a use case. For more details on describing a use case, see "Describing a Use Case of a Project", page 61.*
- *application environment diagrams* that describe the interactions between the environment application elements and the exterior in the context represented by the environment.
  - *For more details on describing an application environment diagram, see "Building the Application Environment Diagram", page 58.*

For a description of the internal structure of an application, see ["Describing an Application with HOPEX Application Design"](#).

---

### Prerequisite

You must first define a work environment for a project in progress. See ["The Current Design Project pane"](#), page 21.

---

### Accessing the List of Application Environments

Accessing the list of application environments for a project:

1. Click the navigation menu, then **Current Design Project**:
2. In the navigation window, select **Functionality and Rules Design**.
3. Click the **List of Application Environments** folder.  
The list of application environments appears in the edit area.

## Application environment properties

The **Characteristics** properties page of an application environment provides access to:

- its **Owner**, by default during creation of the application environment, this is the current enterprise plan.
- its **Name**,
- the text of its **Description**.

See also: ["Describing an Application with HOPEX Application Design"](#), page 76.

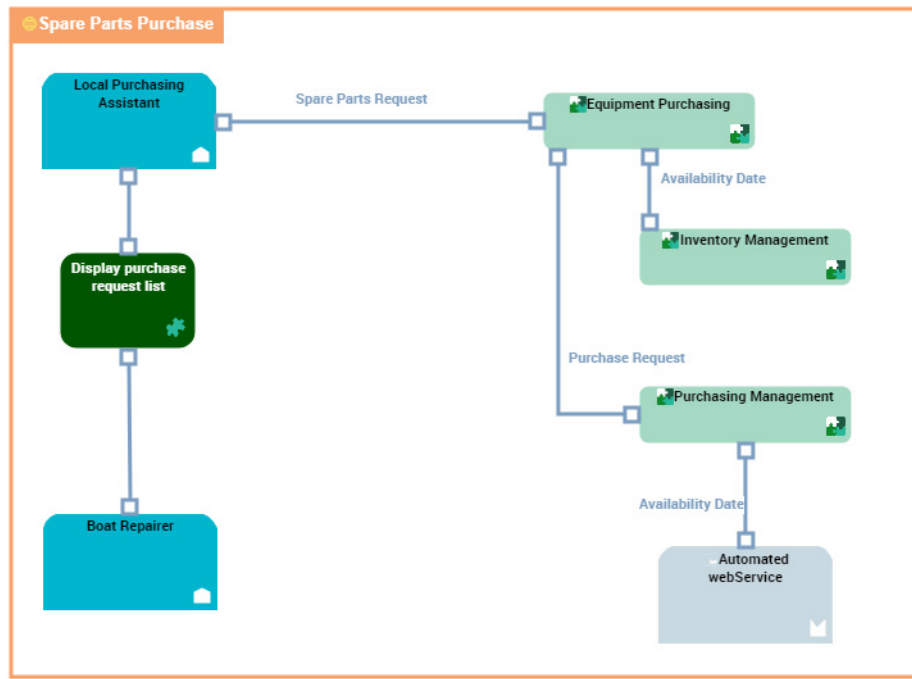
---

## Building the Application Environment Diagram

With **HOPEX Application Design**, an *application environment* is entirely described by a an application environment diagram that is used to describe the interactions between the environment applications described, its users and the external applications.

) *An application environment presents the use context of the applications of a project. This describes the interactions between the org-units involved and the internal and external applications of the projects that ensure the project functionalities.*

The following diagram describes the application environment corresponding to the processing of spare parts purchasing.



*"Spare Parts Purchase" application environment diagram*

The requests for spare parts are formulated by the persons concerned with boat repairs and these requests are processed by the local purchasing assistants.

Consultation of parts in stock is carried out by the local on-site purchasing assistant. Following consultation, the assistant can make an availability request.

Two order types are possible, one for parts already referenced, the other for parts as yet unreferenced. In both cases, a request for availability is put forth.

Order follow-up is assured by the local purchasing assistant and the boat repairer.

An application environment diagram includes:

- **applications** that represent the environment described.

In the example, this concerns the applications used for buying spare parts.

) *An application is a software component that can be deployed and provides users with a set of functionalities.*

- **applications**, **application services** or **partner micro-services** that represent the external elements used in the described environment.

This example concerns automated Web services.

) *An IT service is a component of an application made available to the end user of the application in the context of his/her work.*

- **org-units** or **type positions** that represent the users or the suppliers of the environment described.

This example concerns local participants.

) *An org-unit represents the role played by something or someone within the enterprise environment of the modeled system. It is related to the business activities of the enterprise, and interacts with the system in different use cases. It can be an element in the enterprise structure such as a division, a department, or a workstation.*

- **interactions** between components.

) *An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.*

- access, request and service points

For more details on describing these elements, see ["Managing Interactions"](#).

## DESCRIBING A USE CASE OF A PROJECT

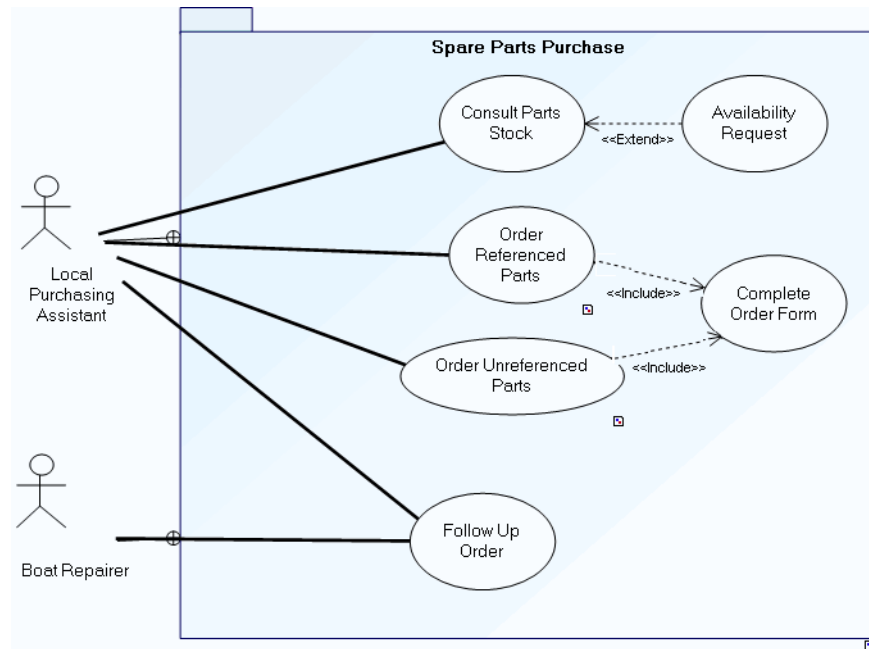
Creating use cases of project components enables representation of the different use contexts of the system. From a use case, you can describe how a functionality or set of functionalities is implemented within an application or service.

By means of use cases, **HOPEX Application Design** enables addition to description of an application architecture of the detailed specifications of its different sub-systems.

## Example of a Use Case

A *use case* diagram enables description of interactions between a system and actors of the organization.

) A use case is a series of actions leading to an observable result for a given actor. Scenarios illustrate use cases for example.



The system is used to consult parts in stock and to order new spare parts.

Consultation of parts in stock is carried out by the local on-site purchasing assistant. Following consultation, the assistant can make an availability request.

Two order types are possible, one for parts already referenced, the other for parts as yet unreferenced. In both cases, an order form should be completed.

Order follow-up is assured by the local purchasing assistant and the boat repairer.

---

## Creating a Use Case Diagram for an Application Environment:

### **Prerequisite**

- You must first define a work environment for a project in progress. See ["The Current Design Project pane", page 21](#).
- An application environment must also be created. See ["Describing an Application Environment", page 57](#).

To create a use case diagram from an application environment:

1. Click the navigation menu, then **Current Design Project**.
2. In the navigation window, select **Functionality and Rules Design**.
3. Click the **Application Environments** tile.  
The list of application environments appears in the edit area.
4. Right-click the application environment concerned and select **New > Application Environment Use Case Diagram**.  
The diagram opens in the edit window.

---

## Use Case Diagram Elements

For a complete description of the use case diagram, see ["Use Case Diagram"](#).

# GENERATING THE FUNCTIONAL ANALYSIS REPORT

In **HOPEX Application Design** a MS Word report template is used to describe the functional specification of IT projects. The report is based in particular on the modeling of use cases, actors and application processes of the project.

To be visible in the report, modeled elements must be within the scope of functional analysis; they must therefore be attached to packages of stereotype "Use case group", "Data model group", "User interface group" or "Actors profile group".

When information exists, the generated report presents:

- actors of the system (system actors or users). These are actors contained in the "*Project name* - Actors" package, or any other package of "Actors profile group" stereotype attached to the project.
- use case diagrams supplemented by description of each use case. These are use cases contained in the "*Project name* - Use cases" package, or any other package of "Use case group" stereotype attached to the project.
- dynamics of user interfaces represented by system process diagrams contained in the "*Project name* - User interfaces" package, or any other package of "User interface group" stereotype attached to the project.
- business data models contained in a package of "Data Model" stereotype, under the package of project use cases.
- state machine diagrams contained in a package of "Data model" stereotype, under the package of project data models.

To create a functional analysis report on an application design project:

1. Click the navigation menu, then **Current Design Project**.
  - *The access to the **Current Design Project** navigation pane requires that you have defined a working environment on the project concerned. See "[The Current Design Project pane](#)", page 21.*
2. In the navigation window, select **Functionality and Rules Design**.
3. In the edit area, click the **Functional Analysis Report** tile.
4. Click **New**.  
The report creation wizard opens.
5. Enter the name of the report.
6. Click **Suivant** then **OK**.

To open the generated document:

- > Click the icon of the document and select **Documentation > Open**.

# DESCRIBING THE DATA OF A PROJECT



In **HOPEX Application Design**, the description of the technical architecture of an application system can be completed by the description of the data used by the system.

The following points explain how to model the data used by processes and applications, without taking their physical implementation into account.

- 6 ["Modeling Project Data", page 66,](#)
- 6 ["Using Data Stores", page 70.](#)

# MODELING PROJECT DATA

**HOPEX Application Design** provides the tools required to model logical data via class diagrams and data models.

Using *data area* and *data views* concepts, you can detail a logical data structure in a particular use context.

- For more details on creating and updating a data model, see the "Data Model" chapter in the **HOPEX Information Architecture** guide.

---

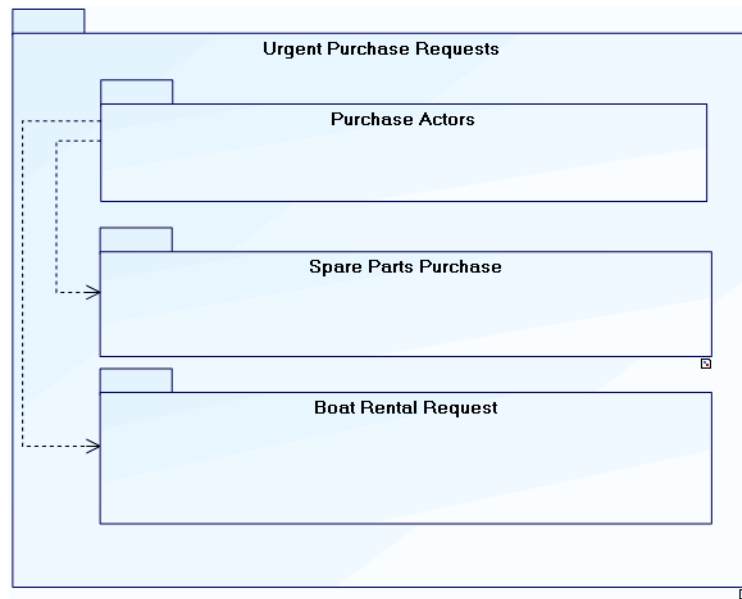
## Logical View Concepts

### UML package

A package is used to represent the static structure of a system, particularly the types of objects handled in the system, their internal structure, and the relationships between them.

The package is an owner element. It provides a namespace for the elements that it consolidates.

The package allows you to classify elements referenced in a project. You can create sub-packages in a package to classify objects in finer detail, for example actors of a project.



Urgent purchase requests are provided to process purchase of spare parts and boat rental requests. In both of these cases, users are actors of the purchasing domain.

Class diagrams are used to graphically represent the elements of a package.

For more details on building a class diagram, see ["The Class Diagram"](#).

See also ["Structure and Deployment Diagrams"](#), page 65.

## Data models

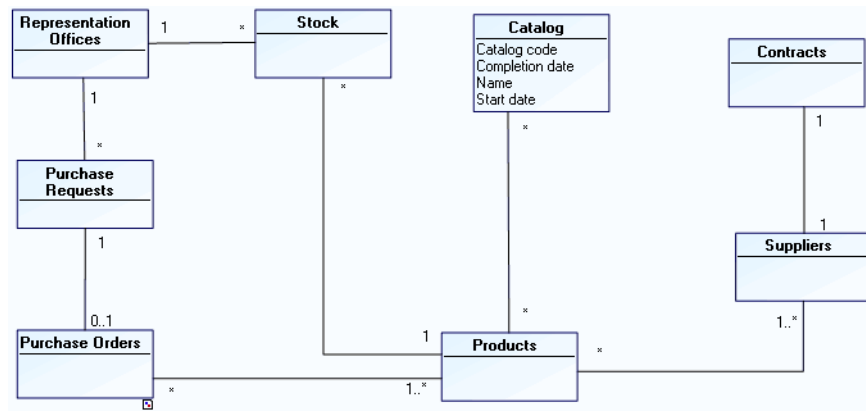
Like a package, a data model is used to represent the static structure of a system, particularly the types of objects handled in the system, their internal structure, and the relationships between them.

Data diagrams are used to graphically represent the elements from a data model.

For more details on creating and updating a data model, see ["The data model"](#), page 25.

## Example

The data model of the "Purchase Request Automation" project is presented below.



The application manages purchase requests, orders and product stock levels in each of the representation offices. A centralized catalog of products and suppliers is installed.

Contracts with referenced suppliers are also accessible from the application.

- For more details on creating and updating a data model, see "Modeling data".

## Data areas

A Data Area represents a restricted data structure dedicated to the description of a software Data Store (see ["Using Data Stores", page 70](#)). It is made of classes and/or data views and can be described in a Data Area Diagram.

A logical data area is used to define a logical data structure made up of classes and data views.

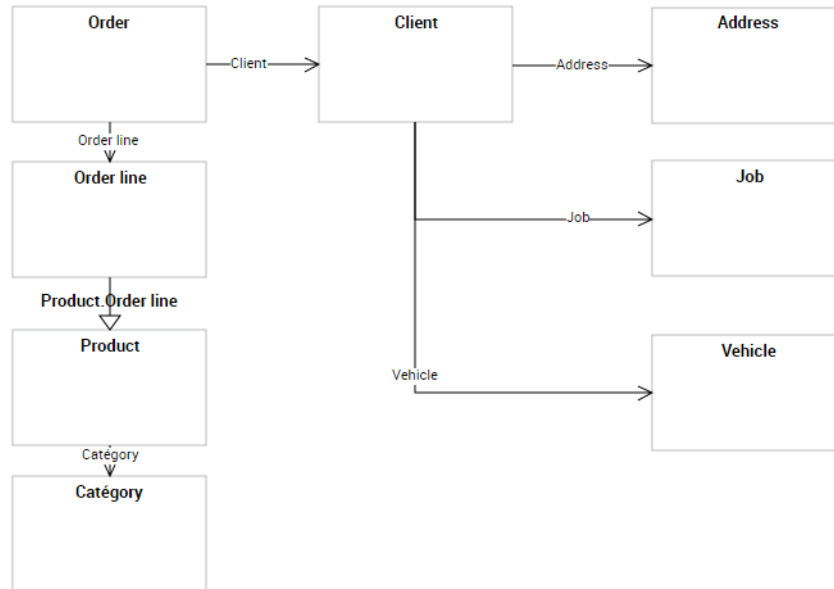
A logical data domain is owned by a package and can reference objects held in other packages.

You can define the access mode (creation, deletion, etc.) to the objects referenced by a data area by integrating them as components of the data area.

During integration with HOPEX Database Builder, a corresponding physical structure can be defined via a physical data area. It is made up of tables and table views.

### Example

The following data area diagram represents a data structure relating to Orders; it describes classes and their relationships in a Whole/Part formalism.



To address these specific use cases, you can create *Data Views* in which you can see and modify the scope covered by the classes.

### Logical data view

A data view represents the scope covered by an element of a data model or a data area. A data view is based on the selection of several classes connected in the specific context of the view.

### Data store

A data store references, in a process or an application system, persistent data defined in a data area.

See ["Using Data Stores", page 70](#).

# USING DATA STORES

) A data store provides a mechanism to update or consult data that will persist beyond the scope of the current process. It enables storage of input message flows, and their retransmission via one or several output message flows.

## Introduction to the data store concept

A data store references an **data domain**.

) A data area represents a restricted data structure dedicated to the description of a software Data Store. It is made of classes and/or data views and can be described in a Data Area Diagram.  
- For more information on data area, see chapter "Logical and application data areas" in the **HOPEX Information Architecture** guide.

If you describe a logical application system, only **logical data stores** can be used.

) A logical data store represents the use of data via application systems without considering how their access will be concretely implemented.

If you describe an application system, only **physical data stores** can be used.

) A physical data store represents the implementation of a logical data store.

If you describe a scenario of sequence or a scenario of flow, only **application data stores** can be used.

) An application data store materializes the usage of data in the context of a software component (for instance an application). An application data store provides a mechanism to retrieve or update information stored outside of the current software component.

) The Scenario of flows diagrams that describe the flows exchanged in different use scenarios of the object described.

) The scenario sequence of flow diagrams that describe the chronology of the flows exchanged in different use scenarios of the object described.

Last but not least, you can also distinguish data stores local to a system from external data stores that are positioned on the border of diagrams.

) A local data store represents a data store used only inside the system described.

) An external data store represents a data store used inside and outside of the system described.

## Usage contexts

The table below presents the list of diagrams that use the different types of data stores.

| Data store type         | Diagrams  |
|-------------------------|---|
| Logical data store      | Logical application system structure diagrams   |
| Physical data store     | Structure diagrams <ul style="list-style-type: none"> <li>- of application,</li> <li>- of application system,</li> <li>- IT service,</li> <li>- of micro-service.</li> </ul>  |
| Application data stores | Scenario sequence diagrams <ul style="list-style-type: none"> <li>- of application,</li> <li>- of application system,</li> <li>- IT service,</li> <li>- of micro-service.</li> </ul> Scenario of flows diagrams <ul style="list-style-type: none"> <li>- of application,</li> <li>- of application system,</li> <li>- IT service,</li> <li>- of micro-service.</li> </ul> |

## Creating a local data store

) A local data store represents a data store used only inside the system described.

To create, for example, a **local physical data store** from an application system structure diagram:

1. Open the diagram that interests you.
2. In the diagram objects toolbar, click **Local physical Data Store**.
3. Click in the frame of the described application system.  
A creation dialog box prompts you to select an existing **Data Area**.
  - The data area is the structure that will concretely support the data store. A physical or an application data store is supported by an **application data area**. A logical data store is supported by a **logical data area**.
  - For more information on data areas, see chapter "Logical and application data areas" in the **HOPEX Information Architecture** guide.
4. Select an existing **Data area**.
5. Click **OK**.  
The local physical data store appears in the diagram with the name of the data area selected.

## Creating an external data store

) An external data store represents a data store used inside and outside of the system described.

To create, for example, an external physical data store from an application system structure diagram:

1. Open the diagram that interests you.
2. In the diagram objects toolbar, click **External physical Data Store**.

3. Click at the edge of the frame of the described application system.  
A creation dialog box prompts you to select an existing **Data Area**.
  - *The data area is the structure that will concretely support the data store. A physical or an application data store is supported by an **application data area**. A logical data store is supported by a **logical data area**.*
  - *For more information on data areas, see chapter "Logical and application data areas" in the **HOPEX Information Architecture** guide.*
4. Select an existing **Data area**.
5. Click **OK**.  
The local physical data store appears in the diagram with the name of the data area selected.

## Describing access to a data store

To create a read access to the data store:

1. In the diagram insert toolbar, click **Link**.
2. Draw a link between the data store and the entity that reads the data (component or application system use).  
A **Read-only access to data storage** is automatically created with the link from the data store to the entity.
  - *To create a link with write access, you must draw a link between this entity and the data store to which it has write access. A **Write access to data storage** is then automatically created.*

# DESCRIBING THE TECHNICAL SPECIFICATIONS OF A PROJECT



After the functional specifications and the description of data used, the technical specifications of the project aim to describe the target architecture of the project components: Application, services, Databases and entities impacted by the project.

## DESCRIBING AN APPLICATION WITH HOPEX APPLICATION DESIGN

An application is a set of software components constituting a coherent whole regarding deployment, functional coverage and IT techniques used.

The application is the management and deployment unit of a set of software components. An application can be deployed on one or several machines.

A project for describing the functional architecture of an information system is used to inventory the existing applications and their interactions. See ["Décrire les spécifications fonctionnelles d'un projet", page 53.](#)

) *An application is a software component that can be deployed and provides users with a set of functionalities.*

---

### Prerequisite

You must first define a work environment for a project in progress. See ["Le volet Projet en cours", page 22.](#)

---

### Creating an Application with HOPEX Application Design


To create an application:

1. Click the navigation menu, then **Current Design Project**.
2. In the navigation window, click **Application Architecture Design**.
3. Click the **Applications** tile.  
The lists of applications appears in the edit area.
4. Click **New**.  
The **Creation of Application** dialog box appears.
5. Enter the **Name** of your application and click **OK**.  
The new application appears in the list.

### Properties of an application with HOPEX Application Design

To access application properties:

- > In the list of applications, select the required application and click the

**Properties**  button of the edit area.

Different pages give you access to application characteristics and components.

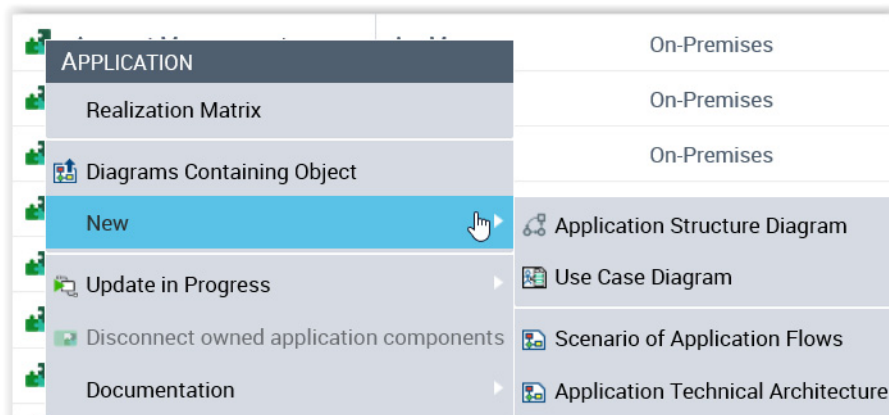
## Modeling an Application with HOPEX Application Design

With **HOPEX Application Design**, an application can be described by several types of diagram:

- an **application structure diagram**: it is used to represent the interactions between the application components in the form of exchange contracts.
  - For more details, see ["Creating an Application Structure Diagram"](#), page 172.
- an **Application Technical Architecture Diagram**.  
An application technical architecture describes one of the configurations possible for application deployment. It describes how the different technical areas of the application are connected to each other and the technologies and the communication protocols that they use. An application can have a number of possible technical architectures (E.g.: autonomous installation, horizontal or vertical deployment, etc.)
  - You can access the list of technical application architectures of the current project in the **Application Architecture Design** pane.
  - For more details on adding applications, see ["Creating an Application Technical Architecture Diagram"](#), page 180.
- A **scenario of application flows**: it describes the flows exchanged between the IT services or the micro-services used by this application. A scenarios can represent a particular application use case or more globally all the flows exchanged within this application.
  - For more details, see ["Using a Scenario of Application Flows Diagram"](#), page 174.

To create an application diagram:

1. Display the applications list.
2. Click the icon of the application and select **New** followed by the diagram type you want to create.



## DESCRIBING IT SERVICES AND MICRO-SERVICES

- ) *An IT service is a component of an application made available to the end user of the application in the context of his/her work.*
- ) *A micro-service is a software component that can be deployed autonomously, but which does not directly provide an end user service. It can interact with other application services, applications or application systems. This is a deployable software component that uses software technologies. For example: an authentication service, a PDF file printing service.*

---

### Prerequisite

You must first define a work environment for a project in progress. See "[Le volet Projet en cours](#)", page 22.

---

### Defining an IT Service or a Micro-Service

#### Accessing the list of IT services

To access the list of IT services using the **Current Design Project** navigation pane:

- > Select **Application Architecture Design > IT Services** .  
The list of IT services appears in the edit area.

#### Accessing the list of micro-services

To access the list of micro-services using the **Current Design Project** navigation pane:

- > Select **Application Architecture Design > Micro-Services** .  
The list of micro-services appears in the edit area.

#### Properties of an IT Service or a Micro-Service

The complete description of an IT service or a micro-service is accessed from its properties pages.

The **Characteristics** properties page for a micro-service provides access to:

- its **Owner**, by default, during creation of the micro service, the current library.
- its **Name**,
- the text of its **Description**.
- the **Technologies** section provides access to the list of *software technologies* used by the micro-services.
  - *For more details on software technologies, see **HOPEX IT Architecture**.*

---

## Using an IT or Micro-Service Flow Scenario

- *For more details on scenarios of flows diagrams, see "Describing a scenario of flows", page 174.*

A micro-service flow scenario represents the flows exchanged between certain elements of the micro-service in a given context. The elements represented are:

- IT services,
- micro services,
- stores of local or external application data,
- input or output application ports.

The interactions offered between these elements:

- application flows that carry a content,
- application flow channels that group a number of application flows on a single link,
- application data channels that represent the interactions between the application data stores.

---

## Using the IT Service or Micro-Service Structure Diagram

- *For more details on structure diagrams, see "Creating a structure diagram", page 172.*

With **HOPEX Application Design**, the components of a micro-service can be described by a micro-service structure diagram.

This type of diagram includes:

- IT services,
- micro services,
- RE or RDB external data stores; see "Utiliser les dépôts de données", page 72.
- access, request and service points; "Describing Service and Request Points", page 187.
- *interactions* between components.
  - ) *An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.*

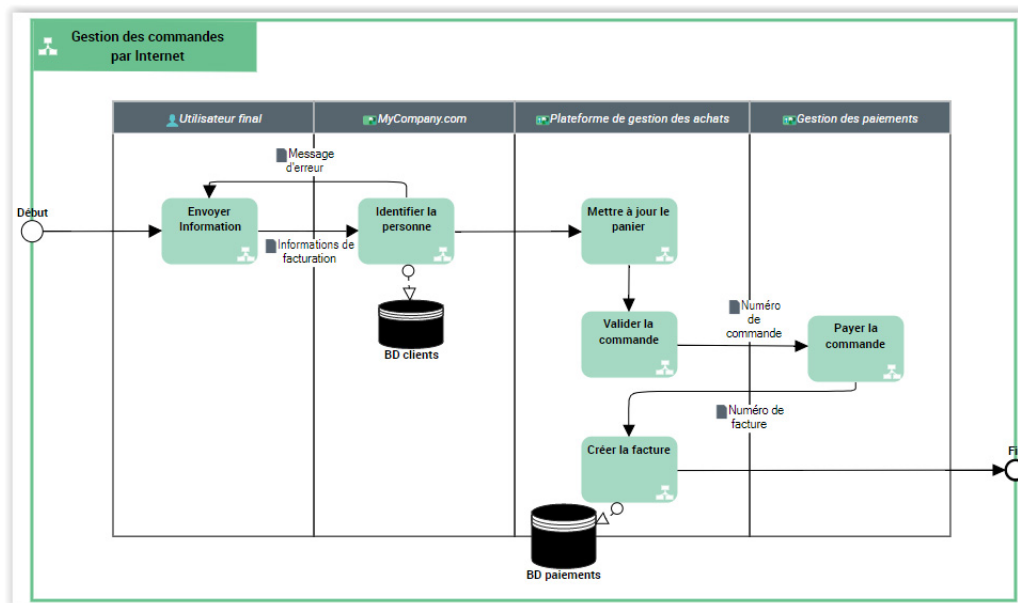
## DESCRIBING APPLICATION PROCESSES

In detailed specification phase, the progress of tasks implemented in an IT service can also be modeled by a system process. More generally, operation of an architecture element can be described by a system process modeling, for example, sequence flow of screens presented to the user.

### System Process example

The diagram below represents purchasing requests processing.

- The final user send information about him/her via "MyCompagny.com" application.
- A customer search is carried out by the application from the Clients data base.
- This error message is returned if the customer is not found.
- If the customer is found, the "Purchasing Management Platform" updates the shopping cart and the order validation is activated.
- The "Validate the order" task activates "Payment management" application for "Order pavements". The invoice number is sent back to the "Purchasing Management Platform" for billing
- The "Create invoice" task updates the payments database and the process ends.



Application process diagram

- For more details on the use of application processes, see the HOPEX Business Process Analysis guide, chapter "[System Processes](#)", page 21.

## Prerequisite

You must first define a work environment for a project in progress. See "[Le volet Projet en cours](#)", page 22.

---

## Managing System Processes with HOPEX Application Design

) A system process is the executable representation of a process. the events of the workflow, the tasks to be carried out during the processing, the algorithmic elements used to specify the way in which the tasks follow each other, the information flows exchanged with the participants.

- For more details on creating system process diagrams, see the **HOPEX Business Process Analysis** guide, paragraph "Managing System Processes".

-

### Accessing system processes

To access the list of system processes using the **Current Design Project** navigation pane:

1. Select **Application Architecture Design**.
2. Select the **System Processes** tile.

The list of system processes appears.

### Creating a system process diagram

To create a system process diagram:

- > Right-click the system process that interests you and select **New > Application Process Diagram**.

The diagram opens in the edit area.

- Diagram initialization automatically positions the frame of the described process and the main events.

---

## Modeling Tasks of a System Process

The functional analysis phase of the project is based on use cases and on the data model, obtained in preliminary study phase, to represent the functional architecture of the future system.

The functional analysis phase describes the system processes implemented in the different use cases of an application or service.

A system process diagram specifies the sequence flow of tasks to be executed so that the user can check that the application satisfies its requirement.

### Describing a Use Case by a System Process

To create a system process from a use case:

1. Right-click the use case to open its pop-up menu.

2. Select **New > System Process**.

The diagram of a new system process opens. The system process associated with the use case is already positioned in the diagram.

- For more details on creating system process diagrams, see the **HOPEX Business Process Analysis** guide, paragraph "Managing System Processes".

## Functional Modeling Example

The *system processes* used for a project functional analysis are stored in a package.

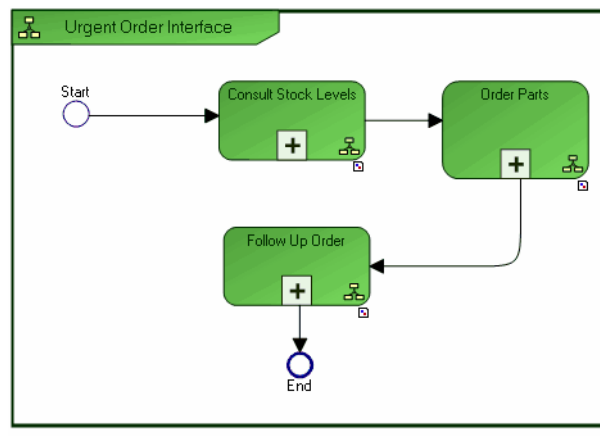
In the example of the purchase request processing automation project, system processes are stored in the "Urgent Purchase Requests" package .

- ) A system process is the executable representation of a process. the events of the workflow, the tasks to be carried out during the processing, the algorithmic elements used to specify the way in which the tasks follow each other, the information flows exchanged with the participants.

## Access the system process diagram from a package

To access the system process diagram from a package:

1. In the **Home** navigation window, expand the tree associated with the package, for example the "Urgent Purchase Requests" package. The list of system processes stored in the package appears.
2. Right-click the system process, for example "Urgent Order Interface" to open its pop-up menu, and select **System Process Diagram**. The diagram opens.



The main steps of this system process are:

consult local stock levels so that a spare part can be made available

order parts when stock level has reached critical threshold or if a part is unreferenced

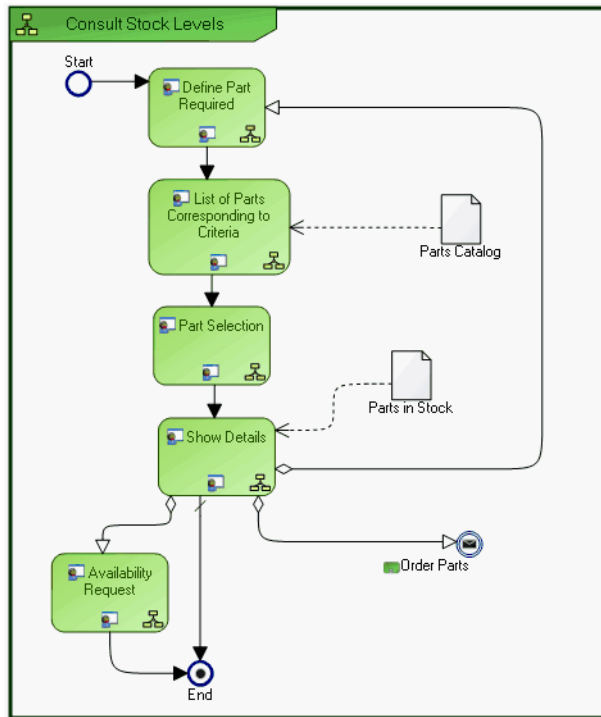
follow-up an order

- You can also access a system process from a use case. For more details, see ["Describing a Use Case by a System Process", page 84.](#)

## Display the diagram describing a step in the system process in detail:

To open the diagram describing a step in the system process in detail:

1. Right-click the system process, for example "Consult Stock Levels" to open its pop-up menu.
2. Select **System Process Diagram**.  
The diagram associated with the process opens.



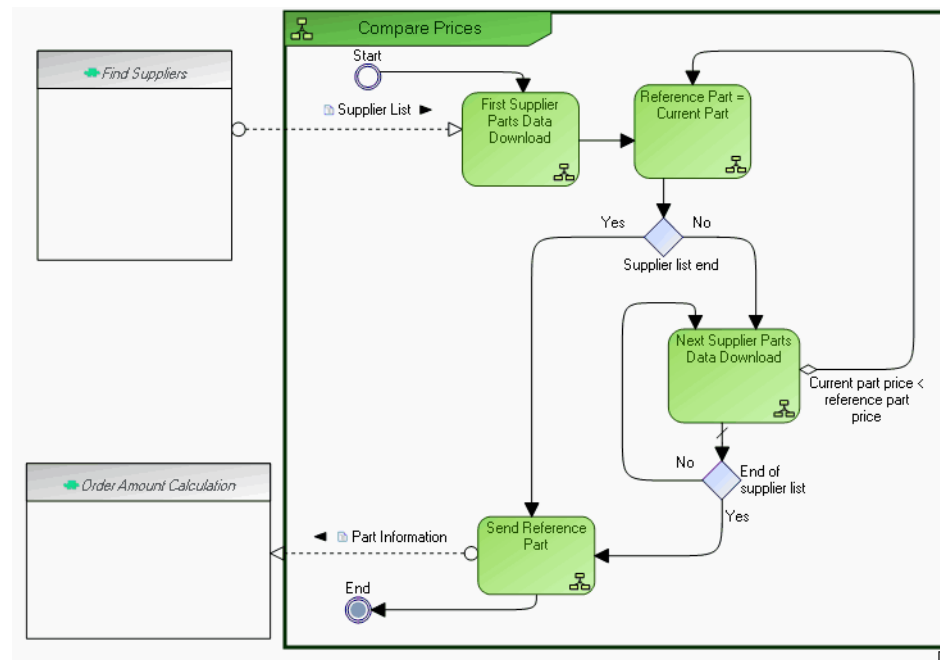
Consulting stock levels begins by display of a screen enabling identification of the required part. The list of parts found in the catalog is presented in the next screen. When the user has selected the required part, information on details is displayed. From this screen, it is possible to obtain information on another part, make an availability request for the part, or indeed order the part.

---

## Modeling Tasks of an IT Service

The phase of detailed analysis of system components impacted by the project consists of detailed modeling of the operation of IT services.

In the context of the urgent order request processing automation example, the service for comparing prices is represented by a system process.



This diagram describes the algorithm of the "Compare Prices" service, which should return the reference of the lowest-priced part.

The list of suppliers of the required part is given at input. The part proposed by the first supplier in this list becomes the reference part. Assuming the supplier list is not empty, data concerning the required part is then analyzed. If the price of the current part is lower than the price of the reference part, the reference part becomes the current part.

When the complete list of suppliers has been analyzed, information concerning the reference part is sent to the "Order Amount Calculation" service.

## DESCRIBING BATCH PROCESSING

With **HOPEX Application Design**, you can describe the sequencing of automated processing in **batch planning structure diagrams**.

This type of diagram is used to represent the execution schedule for batches, batch programs and their organization.

---

### Defining a Batch Process

A batch processing is a set of IT processing operations executed by a computer without human intervention, generally overnight or at the weekend.

A **batch process** is defined by a **batch plan** or by a **program**.

- ) *A batch planning defines all the IT processing operations to be executed on one or several machines over a given time period.*
- ) *A program is an elementary stage in execution of a batch processing that consists of running execution of a program using the appropriate parameters.*

A **batch plan** is a set of **batch processes**. Each is associated either with a **program** or with another **batch plan**. A **batch plan** can be described by a **batch planning structure diagram**.

- *For more details, see "[Building a Batch Planning Structure Diagram](#)", page 88.*

A **program** is a set of **batch processes**. Each of these can be associated with a single **program**. A **program** is described by a **batch program structure diagram**.

- *For more details, see "[Creating a Batch Program Structure Diagram](#)", page 90.*

---

### Building a Batch Planning Structure Diagram

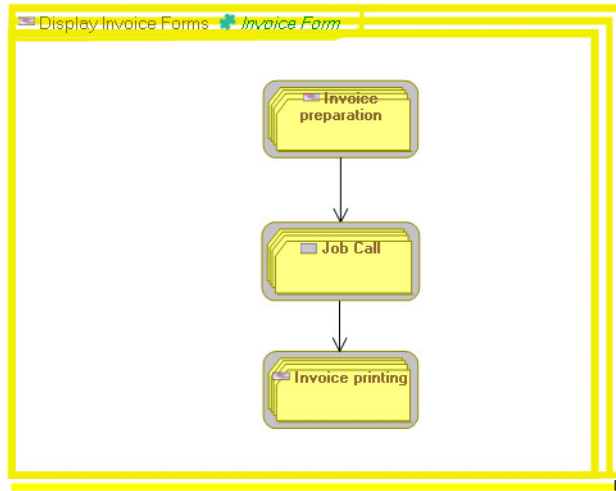
- ) *A batch planning defines all the IT processing operations to be executed on one or several machines over a given time period.*

#### Prerequisite

You must first define a work environment for a project in progress. See "[Le volet Projet en cours](#)", page 22.

## Creating a batch planning structure diagram

The sequencing of automated processes can be described in a **batch planning structure diagram**.



To create a batch planning structure diagram:

1. Click the **Current Design Project** navigation pane.
2. In the navigation window, click **Application Architecture Design**.
3. Click the **Batches** tile.  
The list of batch plans appears.
4. Right-click the batch planning that interests you and select **New > Batch Planning Structure Diagram**.  
The diagram opens.

## Adding a call for batch processing in the diagram

The components of a **batch plan** are defined with **batch processing calls** that are positioned in the diagram. This can be applied to batch plans or programs.

To add an operating type component to the string structure diagram for batch process:

1. Select the **Batch Processing Call** button and click in the diagram.  
The **Add a Batch Processing Call** dialog box opens.
2. Click the arrow at the right of the **Batch Process Called** box and select **Connect** in the drop-down list.  
A connection dialog box opens.
3. Select **Batch Planning** in the left part of the window and click the search arrow.  
The list of batch plans appears.
4. Select the report that interests you and click **Connect**.
5. Click **OK**.  
The call for batch processing appears in the diagram with the batch planning icon.

## Defining batch sequencing

To specify the execution order of processes:

1. Click **Batch Sequence**.
2. Click the initial batch processing call and, holding the left mouse button down, draw a link to the batch processing call.
3. Release the mouse button.  
The link representing the sequencing of the processes appears in the diagram.

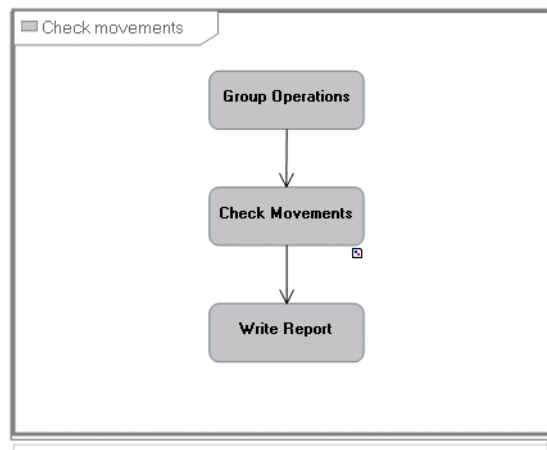
---

## Creating a Batch Program Structure Diagram

) A program is an elementary stage in execution of a batch processing that consists of running execution of a program using the appropriate parameters.

### Creating a batch program structure diagram

The sequencing of the processes of a program can be described in a **batch program structure diagram**.



To create the batch program structure diagram:

1. Right-click the program in question.
2. Select **New** > **Batch program structure diagram**.  
The diagram opens.

### Adding a programming call to the diagram

The components of an **program** are defined with **programming calls** that are positioned in the diagram.

To add a component to a diagram:

1. Select **Programming Call** and click in the diagram.  
The **Add a Programming Call** dialog box opens.

2. Click the arrow at the right of the **Batch Process Called** box and select **Connect** in the drop-down list.  
A connection dialog box opens.
3. Select **Program** in the left part of the window and click the search arrow.  
The list of batch plans appears.
4. Select the program in question and click **Connect**.
5. Click **OK**.  
The program call appears in the diagram.

The of program execution is defined by links. For more details, see ["Defining batch sequencing", page 90](#).

---

## Describing IT Service Implementation

The execution of IT services can be described by a batch planning structure diagram. For more details, see ["Creating a batch planning structure diagram", page 89](#).

To describe that an IT service is implemented by a batch plan, for example:

1. Open the properties window of the IT service in question.
2. Select the **Implementation** tab.
3. In the **Service/Implementation** section, right-click on the **Implementation per Batch Process** folder and select **New**.  
The window for creating an IT service implemented by a batch process appears.
4. Click the arrow at the right of the **Implementing Batch Process** box and select **Connect** in the drop-down list.  
A connection dialog box opens.
5. Select **Batch Planning** in the left part of the window and click the search arrow.  
The list of batch plans appears.
6. Select the report that interests you and click **Connect**.
7. Click **OK**.  
The IT service implemented by a batch process appears in the list. It is connected to the batch plan selected.

---

## Using Realizations

A realization mechanism is provided to associate an application process to an **organization schedule** or via a **Program**.

To describe that an batch plan is associated with an application process:

1. Open the properties pane for the batch plan in question.
2. Select the **Characteristics** tab and the **Realization** subtab.
3. In the **Composite Realization** section, click **New**.  
The window for creating a system process batch realization appears.
4. Opposite the **Base Type** field, select **Connect**.  
The query dialog box appears.

5. Select the application process that interests you and click **Connect**.
6. Click **OK**.  
The system process batch realization appears in the properties page of the batch plan.

## DEFINING USER INTERFACES

It is possible to describe interfaces connecting services or operations with the exterior. This description is carried out in a user interface diagram.

---

### Creating a User Interface

#### **Prerequisite**

You must first define a work environment for a project in progress. See ["Le volet Projet en cours", page 22.](#)

To create a user interface :

1. Click the **Current Design Project** navigation pane.
2. In the navigation window, select **Application Architecture Design**.
3. Click the **System User Interfaces**.  
The list of user interfaces appears.
4. Click **New**.
5. Enter the name of the interface.
6. Click **OK**.

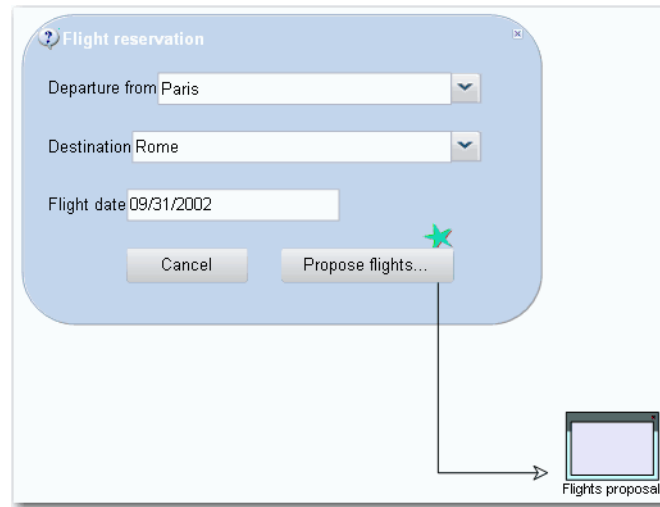
---

### Building a User Interface Diagram

To create an interface diagram:

1. Right-click the user interface in question.
2. Select **New > User Interface Diagram**.  
The UI diagram opens in the Edit window.

Take, for example, the "Flight Reservation" UI diagram.



The interface is presented in the form of a dialog box, in which various fields must be completed.

- Departure city
- Destination
- Flight date

A button cancels the request, another button opens a second interface.

---

## Drawing the Interface Diagram

The user interface diagram allows you to draw the interface of the operation or service.

### User interface element

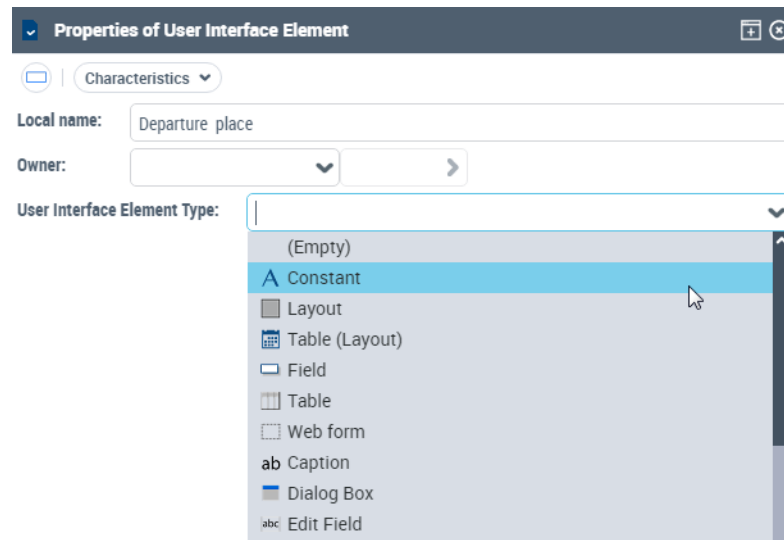
Buttons allow you to modify the appearance of the interface:

- Text Field
- List
- Radio Button
- Check Box
- etc.

To create an element:

1. In the diagram objects bar, select the button corresponding to the element required, then click in the diagram.
2. In the dialog box that appears, enter the name of the element.
3. Click **OK**.

You can also click the **User Interface element** button and indicate the element type in its properties dialog box.



## User interface event

You can connect an *event* to a user interface element. In our example, the "Propose flights" button is connected to an event, which when actuated opens another interface.

To create an event:

1. Click the **Interface event** button, then click in the diagram.
2. Enter the name of the event and click **OK**.

### Event type

There are various types of event. An event can be:

- Click on a button
- Entry in a text field
- etc.

To specify the type of event:

1. Open the properties dialog box of the event.
2. Click the **Characteristics** tab.
3. In the **User Interface Event Type** text box, click the arrow and select **Query User Interface Event Type**.  
The Query dialog box appears:
4. Click **Find**.  
The list of event types appears.
5. Select the type required and click **OK**.

## ***Connecting the event to an element***

To connect the event to an element, there are two possibilities:

- Select the event in the diagram and drag it onto the element.
- Or open the properties dialog box of the event and complete the **User interface element** text box.

# HOPEX APPLICATION DESIGN SYSTEM REPORTS

Two MS Word reports are available as part of the technical specification of a project with **HOPEX Application Design**.

---

## Launching a (MS Word) System Report on an Object

To launch a system report on an application design project:

1. Click the navigation menu, then **Current Design Project**.
  - *The access to the **Current Design Project** navigation pane requires that you have defined a working environment on the project concerned. See ["The Current Design Project pane", page 21](#).*
2. In the navigation window, click **Application Architecture Design**.
3. In the edit area, click the required report tile.
4. Click **New**.  
The report creation wizard opens.
5. Enter the name of the report.
6. Click Next then **OK**.

To open the generated document:

- > Click the icon of the document and select **Documentation > Open**.

---

## System Analysis Reports

This report template focuses more specifically on impacts of the system under study.

The generated report presents:

- all impact diagrams contained in project packages, with detail of elements affected.
- interaction scenario diagrams and sequence diagrams contained in project use case packages. Use cases must be "used" by the project (see ["Etat des livrables du projet", page 172](#)).
- detail of applications and services "used" in the project.
- data models "used" in the project.
- matrix of requirement traceability.

---

## System Design reports

This report template enables description of the internal structure of a project application, the IT services and the databases of this application.

The description of services concerns services "produced" or "updated" in the project. These are services that belong to the internal architecture of the analyzed application, and that are linked to the application by the "defined-service" link.

Each report relates to a specific application. Using this report template you can generate as many reports as there are applications.

A "System Blueprint - System Design" report template presents:

- the application tree of the described application.
- internal architecture diagrams of the application.
- data models that describe databases of the application.
- system processes of services.
- interaction scenario diagrams and sequence diagrams of use cases that implement services of the application.
- matrix of requirement traceability.

# USING IT ARCHITECTURE DIAGRAMS



This chapter explains how to build the main types of IT Architecture diagrams.

The table below draws up the list of solutions for which several descriptions are proposed for different types of objects.

| Object type             | Structure diagram                                    | Flow Scenario  | technical architecture                               |
|-------------------------|--|--|--|
| Application             | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 |
| Application System      | HOPEX IT Architecture V2                             | HOPEX IT Architecture V2                             | HOPEX IT Architecture V2                             |
| Application Environment | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 |
| IT service              | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 |
| Micro-service           | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 | HOPEX IT Architecture V2<br>HOPEX IT Architecture V2 |

- 6 [Structure diagrams and flow scenario diagrams initialization;](#)
- 6 [Creating a structure diagram;](#)
- 6 [Describing a scenario of flows;](#)
- 6 [Describing a Technical Architecture.](#)

# STRUCTURE DIAGRAMS AND FLOW SCENARIO DIAGRAMS

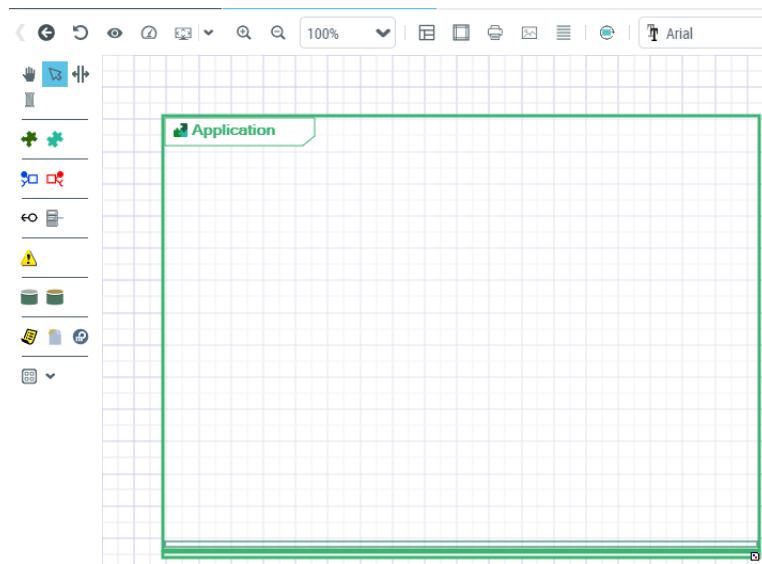
## INITIALIZATION

To create an Application Structure Diagram, for example:

1. Right click the application that interests you and select **New > Application Structure Diagram**.

The diagram opens in the edit area. You are now in the **HOPEX** graphic editor. The frame of the described application appears in the diagram.




### **Example**



### **HOPEX IT Architecture V2 diagrams initialization**

By default, a diagram is initialized with the current object represented by a frame; the described object components are displayed at the top of the diagram.

If the described object is used in an upper level diagram, the new diagram is initialized with service and request points corresponding to the upper level diagrams flows.

| Icon  | Description   |
|---|---|
|  | <b>Refresh diagram</b>  |
|  | <b>Properties of the diagram</b><br>Provides access to the diagram properties           |
|  | <b>Reinitializing components</b><br>Add the described object components in the diagram. |

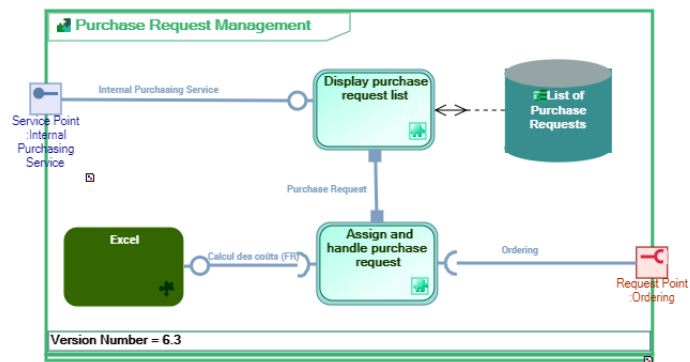
# CREATING A STRUCTURE DIAGRAM

With **HOPEX IT Architecture V2**, the components of an application object and their exchanges are described in a structure diagram.

A structure diagram can be built for an application environment, an application, an application system, an IT service or a micro-service.

## Creating an Application Structure Diagram

) An application structure diagram graphically shows first level components of an application, the access points (service point and request point) and the connections between components.



The "Office Supplies Purchasing Management" application uses two dedicated IT Services: "Display purchase request list" and "Assign and handle purchase request". The "Assign and handle purchase request" IT service invokes Excel micro-service.

## The components of an Application Structure Diagram

An Application Structure Diagram includes:

- **IT services** which represent the IT services used and deployed with the application.

In the example, the services are "Display purchase request list" and "Assign and handle purchase request".

) An IT service is a software component of an application, that can't be deployed alone and that realizes a sub-set of the functionalities of

*this application either for end users of this application or inside the application (or another application). This includes batch programs.*

- **micro-services** which represent the services used independently of the application.

In the example, this is the Excel application.

) *A micro-service is a software component that can be deployed autonomously, but which does not directly provide an end user service. It can interact with other application services, applications or application systems. This is a deployable software component that uses software technologies. For example: an authentication service, a PDF file printing service.*

- request and service points
  - For more details, see [Describing Service and Request Points](#).
- **interactions** between components.
 

) *An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.*
- **physical data stores** used by the application.
  - For more details, see [Managing Data](#).

## Adding an application service to an application structure diagram

To describe that an application uses an application service, go to:

1. In the object toolbar of the application structure diagram, select **Application Service** and click in the frame of the application described. A creation dialog box asks you to select the **Name** of an existing IT service.
2. Select an existing IT service.
3. Click **OK**.  
The application service appears in the diagram.

## DESCRIBING A SCENARIO OF FLOWS

The scenario of flows diagram describes the flows exchanged between the system elements represented.

Depending on the solutions available to you, two types of diagram are proposed:

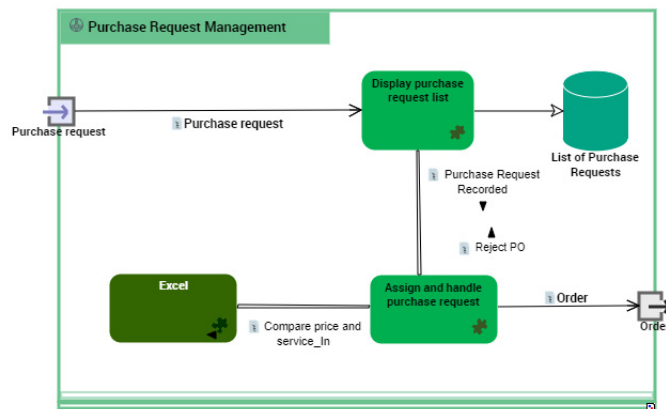
- With **HOPEX IT Architecture V2** and **HOPEX IT Architecture V2**, *Scenario of flows diagrams* to describe the flows exchanged in different use scenarios of the current object.
- With **HOPEX IT Architecture V2**, the *Scenario Sequence diagrams* to describe the chronology of the flows exchanged in different use scenarios of the current object.
  - To see the Flow Scenario Sequence Diagram, open the **Options** and check that **IT Architecture > Activate Flow Scenario Sequence Diagrams** is activated.

---

### Using a Scenario of Application Flows Diagram

An Application Flow Scenario Diagram can be built for an application environment, an application, an Application System, an IT service or a micro-service. This diagram is used to describe the exchanges inside the described object in a specific context.

The scenario of application flow diagram below describes the "Purchase request management" application.



*Example of a Scenario of Application Flows for "Managing Purchase Orders".*

In a scenario of application flows diagram, the elements represented are:

- IT services,
- micro services,
- stores of internal or external application data,
- input or output application ports.

The interactions offered between these elements:

- application flows that carry a content,
- application flow channels that group a number of application flows on a single link,
- application data channels that represent the interactions between the application data stores.

## Creating a Scenario of Application Flows diagram

To create a scenario of application flows:

1. Right-click the application and select **New > Scenario of Application Flows**.
2. (With **HOPEX IT Architecture V2**) in the window for choosing the diagram type, select **Scenario of Application Flows**.

## Adding an IT service to the scenario of application flows

) *An IT service is a software component of an application, that can't be deployed alone and that realizes a sub-set of the functionalities of this application either for end users of this application or inside the application (or another application). This includes batch programs.*

To add an **IT service**:

1. In the objects toolbar of the scenario of application flows, click **Application**.
2. Click in the described application frame.  
A creation window box prompts you to choose the **IT service** implemented (for example "Customer management").
3. Select the application service required and click **OK**.  
The application service appears in the diagram.

You can add a micro-service in the same way.

) *A micro-service is a software component that can be deployed autonomously, but which does not directly provide an end user service. It can interact with other application services, applications or application systems. This is a deployable software component that uses software technologies. For example: an authentication service, a PDF file printing service.*

## Managing application flows in a scenario of application flows

### Creating an application flow with content

) *An application flow represents the circulation of information between applications or within an application. An application flow can carry a content.*

The application flows exchanged between the IT services, the micro-services or the Application ports of a scenario of application flows are associated with a **content**.

) *The content designates the content of a message or an event, independent of its structure. This structure is represented by an XML schema linked to the content. A content may be used by several messages, since it is not associated with a sender and a destination. There can be only one content per message or event, but the same content can be used by several messages or events.*

You must directly specify the *content* of an *application flow* directly on flow creation.

To create the *application flow*:

1. In the insert toolbar of the scenario of application flows, click **Application Flow**.
2. Click the first object representing the sender of the flow and, holding the mouse button pressed, draw a link to the object receiving the flow. The **Application Flow Creation** dialog box opens.
3. In the **Content** drop-down list, select the content you wish to associate with the flow.  
The application flow is displayed with its content in the diagram.

### **Creating an application flow channel**

) *An application flow channel is used to graphically group a number of application flows into a single flow.*

To create an application flow channel, you must first create the channel and then link the application flows that it groups.

To create an *application flow channel*:

1. In the objects toolbar of the scenario of application flows, click **Application Flow Channel**.
2. Click the first object in communication and, holding the mouse button pressed, draw a link to the other object.  
The application flow channel appears in the diagram.

To connect the application flows to the *application flow channel*:

1. Open the **Characteristics** properties page of the application flow channel.
2. In the **Grouped Flow** section, click **Connect**.  
A selection dialog box opens and presents the list of the ungrouped application flows of the scenario of application flows.
3. Select the flows that you want to group and click **OK**.  
The application flow content appears with an arrow that marks the direction of the flow.

### **Reinitialize components in a scenario of flows**

If you insert in a scenario of flows diagram a component that is already described by a scenario of flows, you can note that a new section is created in the **Characteristics** property page of the component you have added. This section allows you to specify which scenario of flows of the component corresponds to the context of the current application system scenario of flows.

In the component scenario of flows diagram, the **Reinitialize components** button allows you to insert components coming from the upper level scenario of flow.

## **Adding an application data store to the scenario of application system flows**

) *An application data store materializes the usage of data in the context of a software component (for instance an application). An*

*application data store provides a mechanism to retrieve or update information stored outside of the current software component.*

- For more information on managing data stores, see [Managing Data](#).

A data store can be local or external to the application.

To add, for example, a local application data store to an scenario of application flows

1. In the scenario objects toolbar, click **Local Application Data Store**.
2. Click in the described application frame.  
A creation window prompts you to choose the *application data area* that represents the physical structure that will concretely support the application data store.  
  
  - ) *An application data area represents a set of data stored in entity/relationship format and used in the technical architecture of an application.*
  - *For more information on data stores, see "Logical and application data areas" chapter in **HOPEX Information Architecture** guide.*
3. Select the existing **Application data area** that interests you.
4. Click **OK**.  
The local application data store appears in the diagram with the name of the physical data domain selected.

## Creating an application data channel

The applications, the application systems and the micro-services can have read or right access to a local or external application data store.

To create an application data channel that represents a reading access:

1. In the diagram objects toolbar, click **Application Data Channel**.
2. Draw a link between the application data store and the object that reads the data.  
An application data channel appears in the diagram.

- *To create a link with write access, you must draw a link between the object that reads and the application data store.*

---

## Using a Flow Scenario Sequence Diagram

- *To see the Flow Scenario Sequence Diagram, open the **Options** and check that **IT Architecture > Activate Flow Scenario Sequence Diagrams** is activated.*

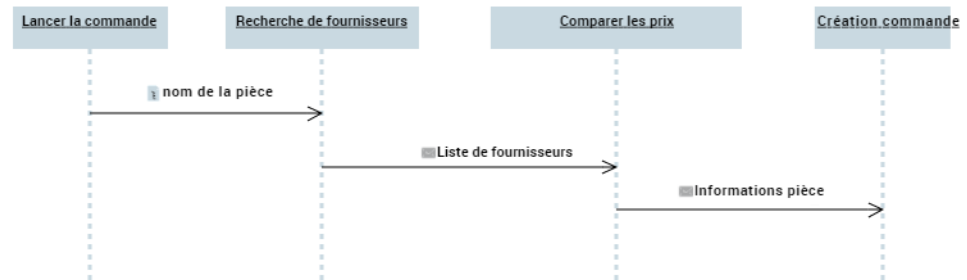
This type of diagram can be built for an application environment, an application, an application system, an IT service or a micro-service.

For each use context, you can create flow scenario sequence diagrams. A flow scenario sequence diagram presents the same exchanges between system elements, highlighting their chronology. The elements in the sequence scenario are represented in the diagram by lines.

A flow scenario sequence of flow diagram contains:

- Lines which define interaction participants: instances of applications, services or interfaces.
- Different types of messages exchanged between participants.
- Advanced functions that enable concise description of several execution sequences.

- For more information on sequence diagrams, see the "**HOPEX UML**" guide.



This diagram describes the operation of the "Order Unreferenced Parts" use case :

When a purchase request is entered in the user interface, the name of the part is received by the "Find Suppliers" service, which draws up the list of suppliers offering the requested part.

The "Compare Prices" service looks for the lowest-priced product and sends information to the "Order Amount Calculation" service.

When the order amount has been established, a final "Issue Purchase Order" service sends the order via the interface.

## Using a Flow Scenario Sequence Diagram

To create an application environment scenario sequence:

1. Right-click the application environment and select **New > Diagram**.
2. In the window for choosing the diagram type, select **Scenario of Application Environment Flows - Application Environment Scenario Sequence Diagram**.

## Instances of applications, IT services or interfaces

Depending on whether the diagram describes a user interface, an application or a, IT service, the interaction scenario diagram describes messages exchanged between application instances, *IT service* instances and *user interface* instances.

) A Human-Machine Interface enables definition of a screen of an application or an IT service.

) An IT service is a software component of an application, that can't be deployed alone and that realizes a sub-set of the functionalities of

*this application either for end users of this application or inside the application (or another application). This includes batch programs.*

To create an application service instance for example:

1. Click the **Application Service** button in the toolbar.
2. Click in the diagram.  
The **Add Application Service** dialog box appears.
3. Click the arrow to the right of the **Name** field and select **Connect Application Service** in the drop-down list.  
The list of application services accessible from the current library appears.
4. Select the IT service you require.
5. Click **OK**.  
The application service instance appears in the diagram.

## Message instance

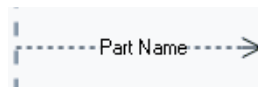
Message instances define the data exchanged between application instances, application services and the interfaces. The sequence described in the flow scenario sequence diagram indicates the message sending order.

Message instances displayed in an scenario sequence diagram correspond to messages owned by the application that have been previously defined in another diagram.

To create a message instance:

1. Click the link button in the toolbar.
2. Click the dotted line under the first object and, holding the mouse button down, draw a line to the second object.
3. Release the mouse button.

The message instance exchanged between the two objects is drawn.

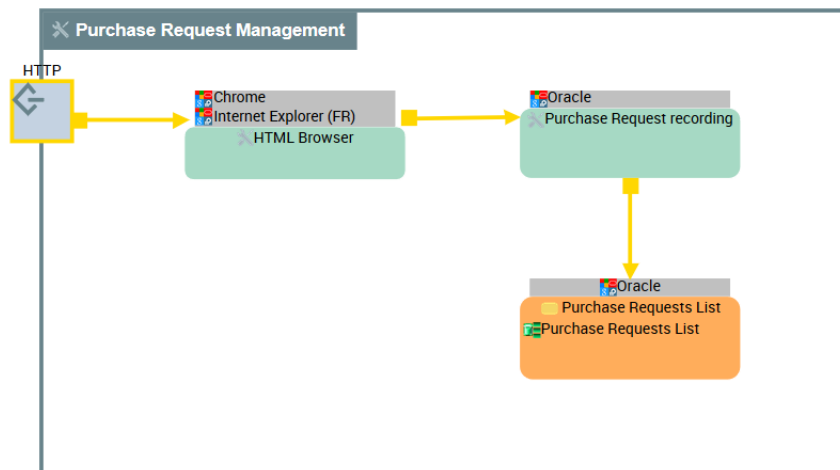


# DESCRIBING A TECHNICAL ARCHITECTURE

An *Technical Architecture* is used to represent the Technical Architectures and the Technical Data Areas of the described object as well as the techniques used for their communications.

The application technical architecture below presents the technical areas used by the purchase request management application.

The technical communication line is based on http.



Technical Architecture Example

) An application technical architecture describes one of the configurations possible for application deployment. It describes how the different technical areas of the application are connected to each other and the technologies and the communication protocols that they use. An application can have a number of possible technical architectures (E.g.: autonomous installation, horizontal or vertical deployment, etc.)

## Creating an Application Technical Architecture Diagram

Elements represented in an Application Technical Architecture diagram are:

- *application technical areas*,

) An application technical area represents an organizational element of an application according to technical criteria. For example, this can be the user interface or a process. Each application technical area is associated with one or more technologies. The deployment of several

*application technical areas is necessary for the application to be operation.*

- **Technical Data Areas,**
  - ) *A data technical area represents an organizational element of an application used to access the data necessary for the operation of this application. Each application technical area is associated with one or more technologies (E.g.: Oracle 12, SQL Server 2012, etc.). A data technical area can allow access to one or more data stores.*
- **Technical Server Port and Technical Client Port,**
  - ) *A technical server port is a point used to open communications with a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).*
  - ) *A technical client port is a point used to request the opening of communications from a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).*
- **Technical Communication Lines.**
  - ) *A technical communication line represents a technical connection between architectures or application technical areas through client and server ports. Client technical port of an architecture or a technical area requires opening the communication line to server technical port of the other area or technical architecture.*

## Adding an application technical area to an application technical architecture diagram

*) An application technical area represents an organizational element of an application according to technical criteria. For example, this can be the user interface or a process. Each application technical area is associated with one or more technologies. The deployment of several application technical areas is necessary for the application to be operation.*

To create an **application technical area**:

1. In the objects toolbar of the application technical architecture, click **Application Technical Area**.
2. Click in the described application frame.  
An addition window prompts you to choose the **Application Technical Area** that you wish to use.
3. Select the application technical area and click **OK**.  
The application technical area appears in the diagram.

You can add data technical areas in the same way.

*) A data technical area represents an organizational element of an application used to access the data necessary for the operation of this application. Each application technical area is associated with one or more technologies (E.g.: Oracle 12, SQL Server 2012, etc.). A data technical area can allow access to one or more data stores.*

## Defining the software technologies used by an application technical area

*) A software technology is a basic component necessary for operation of business applications. Software technologies include all basic software such as: application server, electronic mail server, software components for presentation, data entry, storage, business information sharing, operating systems, middleware, navigators, etc.*

To specify the **software technologies required** for an **application technical area**:

1. Open the **Characteristics** property page of the **Application Technical Area** that interests you.
2. In the **Software Technologies Required** section, click **Connect**.  
In the selection dialog box, select the **Software Technologies** that you want to use.  
The software technologies selected appear in the icon for the application technical area.

## Describing technical communications


Communications between application and data technical areas are supported by technical communication lines. A communication line

- ) *A technical communication line represents a technical connection between architectures or application technical areas through client and server ports. Client technical port of an architecture or a technical area requires opening the communication line to server technical port of the other area or technical architecture.*
- ) *A communication protocol is a set of standardized rules for transmission of information (voice, data, images) on a communication channel. The different layers of protocols can handle the detection and processing of errors, authentication of correspondents, management of routing.*

### Creating a technical communication line

To create a technical communication line, you must first create the line and then specify the **network application protocols** that are used.

To create a **technical communication line**:

1. In the diagram insert toolbar, click **Technical Communication Line** .
2. Draw a line between the two communicating objects.
3. In the creation dialog box, select **Network application protocols** and the **Network application connection**.
4. Click **Add**.  
The technical communication line appears in the architecture. The protocol name appears alongside the channel.

### Technical ports

**Technical ports** assure physical transfer of information exchanged between the technical architecture components.

- ) *A technical server port is a point used to open communications with a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).*
- ) *A technical client port is a point used to request the opening of communications from a technical architecture or an application technical area in compliance with a particular communication protocol (SMTP, HTTP, etc.).*

Technical ports comply with network application protocols.

- *Network application protocols supported by a communication port must be compatible with the protocols supported by communication ports to which they are connected.*

# DESCRIBING DATA EXCHANGES



This chapter explains how to describe exchange contracts between the components of a business or IT architecture.

- 6 [Managing Interactions;](#)
- 6 [Describing Exchanges;](#)
- 6 [Describing Exchange Contracts.](#)

# MANAGING INTERACTIONS

An *Interaction* represents the exchange of information between architecture components.

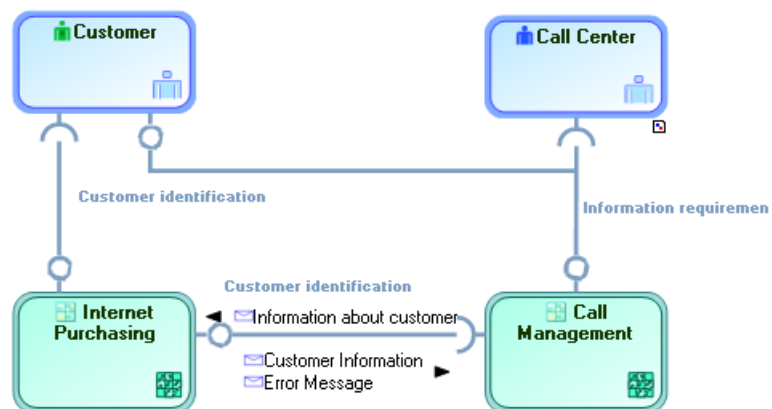
) An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.

Content of an interaction is described by an *exchange contract*.

) An exchange contract is a model of a contract between organizational entities. This contract is described by exchanges between an initiator role and one or several contributor roles.

- For more details on exchange contract concepts, see [Describing Exchange Contracts](#).

In a "Purchasing Requests Processing" application system structure diagram, two exchange contracts are used by different interactions.




Interactions in the "Purchasing Requests Processing" application system structure diagram

The clients must be identified before entering an order. They can enter orders directly from an eCommerce application or by using a Call Center. The Call Center uses the "Call Management" application which uses the client identification service offered by the "eCommerce Purchasing" application.

---

## Creating an Interaction

To create an interaction:

1. In the objects toolbar for a diagram, click **Interaction** 
2. Click the entity requesting the service and draw a link to the entity providing the service.
3. In the add interaction dialog box, specify the exchange contract you wish to use.
  - You can also create a new exchange contract. For more details, see [Creating an Exchange Contract from an Interaction](#).
4. Click **Add**.

---

## Describing Service and Request Points

In a service-oriented architecture, communication is based on access points: *service points* and *request points*.

- ) A request point is a point of exchange by which an agent requests a service from potential suppliers.
- ) A service point is a point of exchange by which an agent offers a service to potential customers.

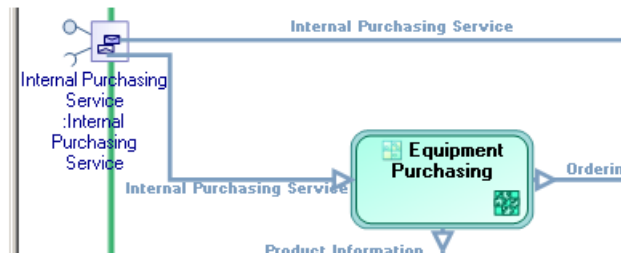
### Service points

An application system, for example, is created to ensure one or more services. These services are represented by *service points*.

The service is requested according to precise terms defined by an *exchange contract* assigned to the service point.

- ) An exchange contract is a model of a contract between organizational entities. This contract is described by exchanges between an initiator role and one or several contributor roles.
- For more details on exchange contracts, see [Describing Exchange Contracts](#).


Components activated to assure a service are linked to the service point by interactions. If it is necessary to activate several components, you have to create several interactions between the service point and the system components.



In the example presented here, the internal purchasing service is linked to two interactions based on the same exchange contract, representing the activation of the equipment purchasing application or the office supplies purchasing application.

- To create a service point, see [Creating a Service Point or a Request Point](#).

## Request points

A **request point**  enables representation of use of a service external to the described entity.

- ) A request point is a point of exchange by which an agent requests a service from potential suppliers.

A service point is a point of exchange by which an agent offers a service to potential customers. The service is requested according to precise terms defined by an **exchange contract** assigned to the request point.

- ) An exchange contract is a model of a contract between organizational entities. This contract is described by exchanges between an initiator role and one or several contributor roles.
- For more details on exchange contracts, see [Describing Exchange Contracts](#).

Components that issue a request are linked to the request point by an interaction.



In the example, request points represent requests for service executed by the "Adding Products to Shopping Cart" logical application to issue an order or book products.


- To create a request point, see [Creating a Service Point or a Request Point](#).

## Creating a Service Point or a Request Point

The process for creating a *service point* or *request point* is identical.

- ) *A request point is a point of exchange by which an agent requests a service from potential suppliers.*
- ) *A service point is a point of exchange by which an agent offers a service to potential customers.*

To create a service point:

1. In the diagram insert toolbar, click **Service Point** .
2. Position the object at the edge of the frame of the described object.  
A creation dialog box opens.
3. Click the arrow to the right of the **Exchange Contract** field to define the exchange contract enabling activation of this service point, and select, for example, **Connect Exchange Contract**.  
A query window opens.
4. Select the exchange contract associated with this service point and click **Connect**.
5. Click **Next**.  
A dialog box opens proposing a list of exchange contract roles that can be associated with the service point.
  - *This dialog box is not proposed if there is only one candidate role that can be associated with the service point.*
6. Select the role that interests you and click **OK**.  
The service point appears in the diagram.

To change the service point name:

1. Click the name of the service point and press key F2.
2. Enter the new name used when specifying interaction points.
  - *For more details on interaction points, see [Describing Service and Request Points](#).*

---

## Defining the Element Interaction Point

The interaction point of an element connects an interaction to one of the components in communication. This specifies:

- on the one hand, the service point, or the request point, that intervenes in the communication
- on the other hand, the role, consumer or supplier represented by the interaction point in the exchange contract.

## Characterizing the element interaction point

To modify the properties of the element interaction point:

1. Right-click the interaction beside the communication element.
2. Open the **Characteristics** properties page.

3. Select the **Played Service Role**, that is the role of the exchange contract played by the element interaction point.
  - For more details on the roles of an exchange contract, see [Creating an exchange diagram \(BPMN\)](#).
4. Select the **Interaction Endpoint Target**, that is the service (or request) point concerned by the interaction.
  - For more information on service points or request points, see [Describing Service and Request Points](#).
5. Click **OK**.

## DESCRIBING EXCHANGES

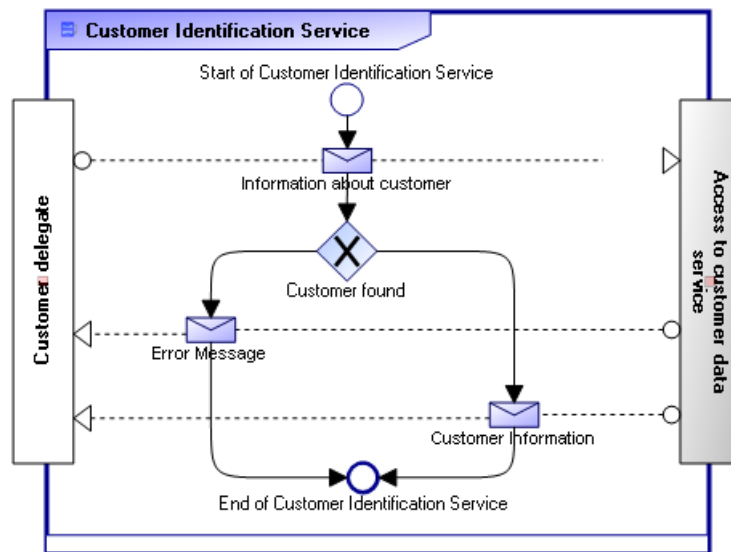
Content of an interaction is described by an *exchange contract*.

- ) An exchange contract is a model of a contract between organizational entities. This contract is described by exchanges between an initiator role and one or several contributor roles.
- For more details on exchange contracts, see [Describing Exchange Contracts](#).

An exchange contract is described by a sequence flow of exchanges or exchange contracts.

- ) An exchange specifies message flow exchanges between two participants.

An exchange diagram describes the sequence flows of an *exchange*.




"Customer Identification Service" Exchange Diagram

The customer identification service protocol begins by sending information enabling identification of the customer. An error message appears if the customer is not found, otherwise customer information is sent (customer identification, status of orders, etc.).

## Creating an Exchange

You can create an *exchange* from an exchange contract diagram (BPMN).

To create an *exchange* from an exchange contract diagram (BPMN).

1. Click the **Exchange Use** button  and click in the diagram within the described exchange contract frame.

) *An exchange use represents the usage of an exchange in another exchange contract.*

The Creation of Exchange Use dialog box opens.

2. Click the arrow to the right of the **Exchange Specification** field and select **Create Exchange** in the drop-down list.  
The Creation of Exchange dialog box appears.
3. Enter the **Name** of your exchange and click **OK**.
4. In the **From** field, select the exchange contract role described connected to the "Consumer" role of the exchange used.
5. In the **To** field, select the exchange contract role described connected to the Supplier role of the exchange used.
6. Click **Finish**.
7. Click **OK**.  
The exchange is automatically created.

---

## Describing Exchanges

### Creating an exchange diagram (BPMN)

An *exchange* is described by an exchange diagram presenting the sequence flow of messages exchanged.

To create an exchange diagram:

- > Right-click an **Exchange** and select **New > Exchange Diagram (BPMN)**.

The diagram opens. The exchange frame is positioned and the two roles (Consumer and Supplier) are created.

### Creating a message flow with content

You must specify the *message flows* and their *content* exchanged between the two exchange roles.

) *A message flow represents circulation of information within an exchange contract. A message flow transports its content.*

) *The content designates the content of a message or an event, independent of its structure. This structure is represented by an XML schema linked to the content. A content may be used by several messages, since it is not associated with a sender and a destination. There can be only one content per message or event, but the same content can be used by several messages or events.*

To create a message flow and its content:

1. In the exchange diagram, click the **Flow With Content** button.
2. Click the role that represents the message flow sender and, holding the mouse button down, draw a link to the message flow recipient.  
The **Creation of Message Flows With Content** dialog box opens.

3. In the **Content** drop-down list, select the content you wish to associate with the flow.  
The message flow is displayed with its content in the diagram.

## Managing events, gateways and sequence flows

"Start" and "End" *events* are required in the description of the service assured by the exchange contract.

) *An event represents a fact or an action occurring in the system, such as updating client information. It is managed by a broker. An application indicates that it can produce the event by declaring that it publishes it. If an application is interested in an event, it declares that it subscribes to the event.*

In compliance with the BPMN standard, in the object toolbar, several *gateway* types are available to you.

) *Gateways are modeling elements that are used to control how sequence flows interact as they converge and diverge within a process.*

A *sequence flow* is a directional link that represents the chronological organization of the different processing steps.

) *A sequence flow is used to show the order in which steps of an exchange contract will be performed. A sequence flow has only one source and only one target.*

- *For more details on events, gateways and sequence flows, see [Managing events, gateways and sequence flows](#)*

## DESCRIBING EXCHANGE CONTRACTS

An *Interaction* represents the exchange of information between architecture components.

) *An interaction represents a contract established in a specific context between autonomous entities that are internal or external to an enterprise. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this contract is described by an exchange contract.*

Content of an interaction is described by an *exchange contract*.

) *An exchange contract is a model of a contract between organizational entities. This contract is described by exchanges between an initiator role and one or several contributor roles.*

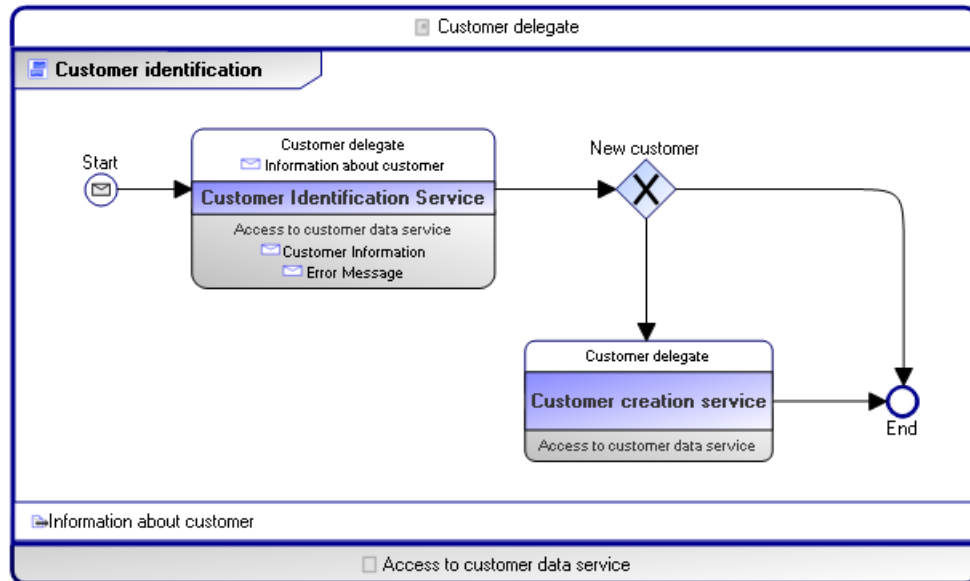
An exchange contract is described by a sequence flow of operations which are represented:

- by *exchange contract use*
  - ) *An exchange contract use is associated with an exchange contract. It enables representation of complex exchanges.*
- or by *exchange use*
  - ) *An exchange use represents the usage of an exchange in another exchange contract.*
    - *For more details on exchanges, see [Describing Exchanges](#).*

## Examples of Exchange Contract Diagrams (BPMN)

### Exchange contract diagram (BPMN)

The exchange contract diagram associated with the customer identification exchange contract describes, in BPMN formalism, the operations executed.



Exchange Contract Diagram (BPMN) "Customer Identification"

Customer identification protocol starts with a customer identification step. If the customer is found the exchange contract returns customer information, if not, a customer creation exchange contract is activated.

Progress steps are represented by *exchange use*.

) An exchange use represents the usage of an exchange in another exchange contract.

### Advanced communication exchange contract example

An exchange contract is described by a sequence flow of steps which are represented:

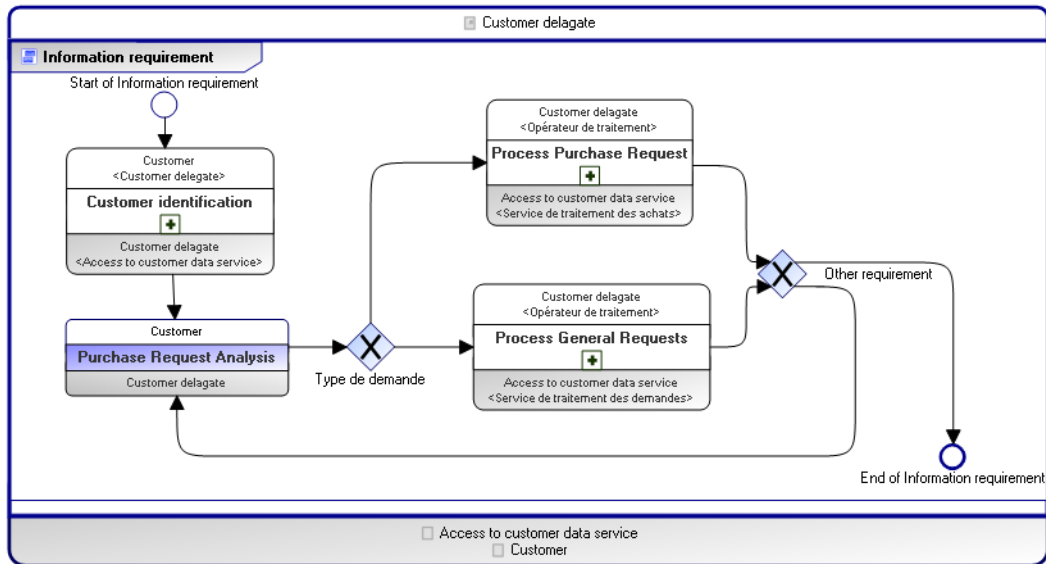
- either by *exchange use*
- or by *exchange contract use*

) An exchange contract use is associated with an exchange contract. It enables representation of complex exchanges.

The exchange contract roles, presented at the border of the frame, represent participants:

- customer/supplier, or
- sender/recipient

An exchange can be described by involving more than two participants. In this case, one role is the initiator of the exchange contract and the others are contributors.



*"Information Requirement" Exchange Contract Diagram (BPMN)*

The "Information Request" exchange contract is used by the call center to take account of a customer request online. There are therefore three participants in this exchange contract: the customer, the IT applications and the customer representative who is the effective requester of the service (in this case the call center).


This exchange contract consists of identifying the customer, then analyzing the request. The request is then processed as a purchase request or as another request if it is an information request for example.

## Creating an Exchange Contract from an Interaction

You can also create a new exchange contract:

- from a library,
- from an interaction in a diagram.

To create an exchange contract, in a diagram, from an interaction:

1. In the diagram insert toolbar, click the **Interaction** button. 
2. Draw a link between the two communication entities.
3. In the add interaction dialog box, click the arrow at the right of the **Exchange Contract** box and select **New**.  
The **Creation of Exchange Contract** dialog box opens.
4. Enter the name of the exchange contract in the **Name** box.
5. Click **OK**.  
The interaction and exchange contract are created.

---

## Creating an Exchange Contract Diagram (BPMN)

An exchange contract is represented by an **Exchange Contract Diagram (BPMN)**.

To create an Exchange Contract Diagram (BPMN) from an interaction:

1. Right-click the interaction.
2. Select the associated exchange contract and, in its pop-up menu, click **New > Exchange contract diagram (BPMN)**  
The diagram opens with the exchange contract frame and the two *roles* representing consumer and the supplier.

) A role is a participant in an interaction, workflow or process. It can be the initiator, that is the requester of a service, or it can represent a sub-contractor carrying out processing outside the service. A role is an integral part of the object that it describes, and is not reusable. It can subsequently be assigned to an org-unit internal or external to the organization or to an IT component. Examples: client, traveler.

The *events*, *gateways* and *sequence flows* of your diagram follow the BPMN standard.

- For more details on events, gateways and sequence flows, see [Managing events, gateways and sequence flows](#)


---

## Defining an Exchange or an Exchange Contract Use

In an Exchange Contract Diagram (BPMN), operations are described by:

- *exchange contract use*
- *exchange use*
  - ) An exchange contract use is associated with an exchange contract. It enables representation of complex exchanges.
  - ) An exchange use represents the usage of an exchange in another exchange contract.

To create an *exchange contract use*:

1. Click the **Exchange Contract Use** button  and click in the diagram within the exchange contract frame.  
The creation dialog box opens.
2. Click the arrow to the right of the **Specification of an Exchange Contract Use** box.

3. Select **Connect Exchange Contract** from the drop-down list and choose the exchange contract that you want to use.
4. In the **From** field, select the described exchange contract role connected to the "Consumer" role of the exchange contract use.
5. In the **To** field, select the described exchange contract role connected to the "Supplier" role of the exchange contract used.
6. Click **Finish**.



# **UML Modeling**



# ABOUT UML IMPLEMENTATION



UML (Unified Modeling Language) is established as the standard for the graphic modeling of information systems. **HOPEX Application Design** offers a set of tools allowing you to model your IS in compliance with version 2.3 of this standard.

The aim of this guide is to introduce you to the main functionalities of HOPEX UML.

- 6 [Overview](#)
- 6 [Organization of UML Diagrams](#)

# OVERVIEW

With **HOPEX Application Design**, you can:

## Analyzing use cases

Before designing a system, there must be a careful analysis of the functions expected of it. The system components will be used by the “actors” in the organization to perform their tasks. The various “use cases” for the system will be presented in **use case diagrams**.

These are used as the starting point for discovering objects.

They then allow validation of the use of these objects in the interaction diagrams.

Then they provide criteria for grouping the discovered objects into “packages”.

See [Use Case Diagram](#).

## Identifying objects

Objects with a similar structure, the same behavior, and the same types of relations with other objects, are placed in the same class.

The **class diagram** identifies the objects involved within the system and defines their structure in terms of attributes and operations, as well as the relationships between them. The **object diagram** shows the instances compatible with a particular class diagram, and can be used as an example to verify it.

See [The Class Diagram](#).

## Describing behaviors

The **state machine diagram** enables definition of the behavior of an object in response to internal or external requests it may receive. It indicates each possible object state, and the reaction of the object to a given event when in that state.

The **activity diagram** also describes a behavior, but in terms of actions.

See:

- [State Machine Diagram](#)
- [Activity Diagram](#)

## Representing interactions between objects

The resulting dialog that is initiated between the different objects concerned by the event can be represented in **interaction diagrams**.

Interaction diagrams emphasize the exchanges that take place between objects.

The **sequence diagram** shows the same exchanges, but indicates the chronology.

The **communication diagram** highlights structural organization of objects that send and receive messages.

The **interaction overview diagram** provides a general view of control flow.

See [Interaction Diagrams](#).

## Dividing classes between packages

Once the objects are identified, it is easy to divide the classes that implement them into different packages. These classes are grouped in the **package diagram** so as to minimize exchanges between different packages. They meet two criteria: the first is more technical and concerns the execution environment, while the other is more structural and is related to the use it will be put to by the users for each use case.

See [The Package Diagram](#).

## Defining interfaces

To respect the principle of encapsulation, there is strict distribution of elements between components. This means interfaces must be provided between elements that have relationships but belong to different components.

The **component diagram** and the **composite structure diagram** present the interdependence between components or component elements.

Defining object interfaces while complying with a standard exchange protocol (CORBA2, DCOM/OLE) is key to interoperability, enabling objects developed and used in heterogeneous environments to work together.

See:

- [The Component Diagram](#)
- [Composite Structure Diagram](#)

## Specifying deployment

Deployment of objects in an actual work environment can be specified in the **deployment diagram**.

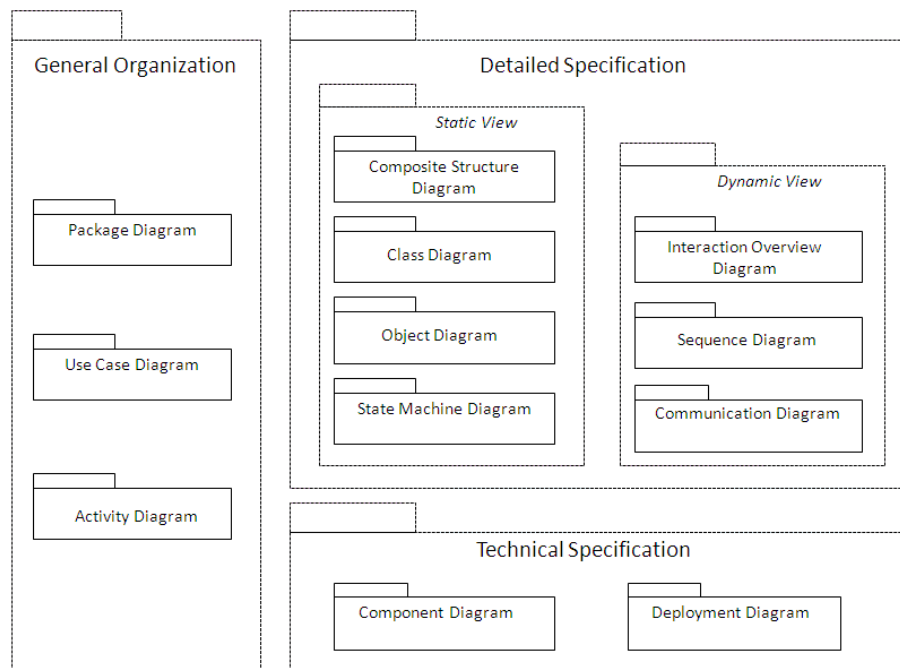
See [The deployment diagram](#).

# ORGANIZATION OF UML DIAGRAMS

## General organization

**Use case diagrams** show the main interactions between the system being analyzed and its environment, and indicate its main sub-systems.

**Package diagrams** provide a more technical breakdown of the system. Dividing a system into packages imposes some structure, as an object can only be in one package. You can begin drawing package diagrams as soon as you have identified the main components of your system (Sales, Production, Invoicing, etc.).



## Detailed specification

The main diagram is the **class diagram**. It describes the essential semantics of the objects in the system. This is where designers will generally spend most of their time. Classes are generally discovered by iteration between class and sequence diagrams.

The **state machine diagram** describes the static aspects of an object: the different states it can be in and the possible state transitions. This fleshes out the class description.

The **interaction diagrams** describe the dynamic aspects of the system, by showing the interactions between objects. They provide a detailed description of the different scenarios in a use case. The sequence diagram specifies how a scenario progresses

over time, while the communication diagram stresses the interactions between objects.

## Technical specification and deployment

The **component diagram** describes the different technical components of an application and shows their interactions.

The **composite structure diagram** specifies collaborations between components or component elements in execution of a common task.

The **deployment diagram** is used to specify the system architecture, indicating the workstations or nodes in the information system where the different application components are to be installed.

## UML diagram entry points

| Diagram                     | Entry points                                      |
|-----------------------------|---|
| Class diagram               | Package, class, use case                          |
| Object diagram              | Class, component, package, use case               |
| Component diagram           | Component, package                                |
| Composite structure diagram | Component, class, collaboration                   |
| Deployment diagram          | Package   |
| Package diagram             | Package, library                                  |
| Use case diagram            | Package, Use case, Application environment (ADES) |
| Sequence diagram            | interaction                                       |
| Communication Diagram       | interaction                                       |
| Interaction                 | interaction                                       |
| overview diagram            |   |
| Activity diagram (UML2)     | Activity  |
| State machine diagram       | State machine, , protocol state machine           |

In **HOPEX Application Design**, the entry points above are accessible in the **Application Resources > UML Implementation** navigation pane.



# USE CASE DIAGRAM



The use case diagram constitutes a first step in description of an information system. It enables identification of the functionalities to be provided by the system to meet the requirements of organization actors; it therefore describes interactions between the system and the actors.

The following points are covered here:

- 6 [Creating a Use Case Diagram](#)
- 6 [Use Case Diagram Elements](#)

# CREATING A USE CASE DIAGRAM

A use case diagram is used to describe the interactions between the organization actors and the system, for each of the planned *use cases*.

) *A use case is a series of actions leading to an observable result for a given actor. Scenarios illustrate use cases for example.*

These use cases are grouped into *packages* representing the system boundaries.

) *A package partitions the domain studied and the associated work. It enables grouping of various elements, in particular use cases and classes. A package can also contain other packages. Packages are interconnected through contractual reports defining their interface.*

You can create a use case diagram from a package. However, for complex systems, you can create this type of diagram from a use case in order to detail the latter.

With **HOPEX Application Design**, you can also create a use case diagram for the application environment of a project. See ["Creating a Use Case Diagram for an Application Environment:"](#).

---

## Creating a Package

### **Prerequisite**

You must first define a work environment for a project in progress. See ["The Current Design Project pane"](#), page 21.

To create a package with **HOPEX Application Design** :

1. Click the navigation menu, then **Current Design Project**.
2. In the navigation pane, select **Project Scope**.
3. In the edit area, click **Packages**.
4. In the edit area, click **New**.  
The Creation of Package dialog box appears.
5. Enter its **Name**.
6. Indicate the library and owner package if necessary.
  - *The default library is used to store an object if there is no current library at the time of its creation.*
7. Click **OK**.

The package is created and added to the list of packages.

---

## Creating the Use Case Diagram of a Package

To create a use case diagram:

1. Click the icon of the package concerned and select **New > Use Case Diagram**.

The diagram opens in the edit window. The frame of the package is positioned within the drawing.

# USE CASE DIAGRAM ELEMENTS

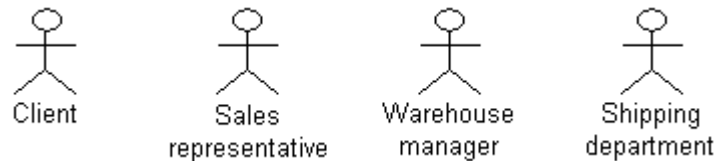
- 6 [Actors](#)
- 6 [Use Cases](#)
- 6 [Packages](#)
- 6 [Participations](#)
- 6 [Associations Between Use Cases: Extension and Inclusion](#)
- 6 [Generalizations](#)
- 6 [Interfaces](#)

---


## Actors

) An org-unit represents the role played by something or someone within the enterprise environment of the modeled system. It is related to the business activities of the enterprise, and interacts with the system in different use cases. It can be an element in the enterprise structure such as a division, a department, or a workstation.

Examples of actors:



To create an *actor* in a use case diagram:

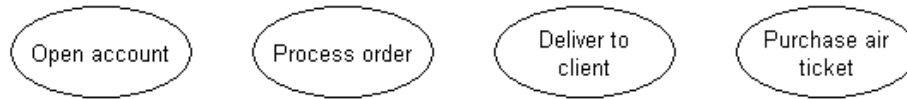
1. In the object toolbar, click **Actor** 
2. Click in the diagram.  
The **Add Actor** dialog box opens.
3. Enter the name of the actor, "Receptionist" in this example.
4. Click **Add**.  
The actor then appears in the diagram.

M You can create several elements successively without clicking in the toolbar each time by double-clicking the **Actor** button.


M To return to normal mode, press <Esc>, or click on another button in the toolbar.

## Use Cases

Examples of *use cases*: processing an order, delivering to a client, opening an account, sending an invoice, establishing credit, purchasing an airline ticket, etc.



To create a use case in a diagram:

1. In the use case diagram objects toolbar, click the **Use Case**  button. The **Add Use Case** dialog box opens.
2. Enter the use case **Name** and click **Create**.

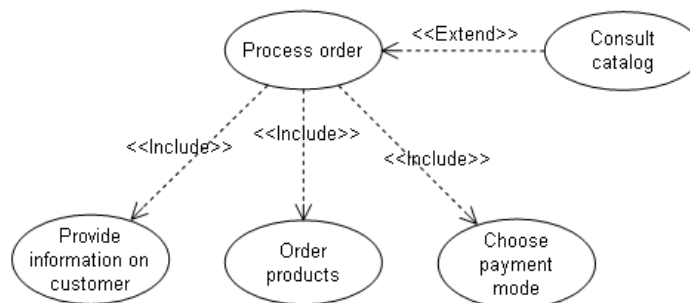
The use case appears in the diagram.

## Zooming in on a use case

To open the diagram that describes a use case directly from the package diagram:

1. Right-click the use case.
2. Select **Use Case Diagram**.

The use case diagram opens.

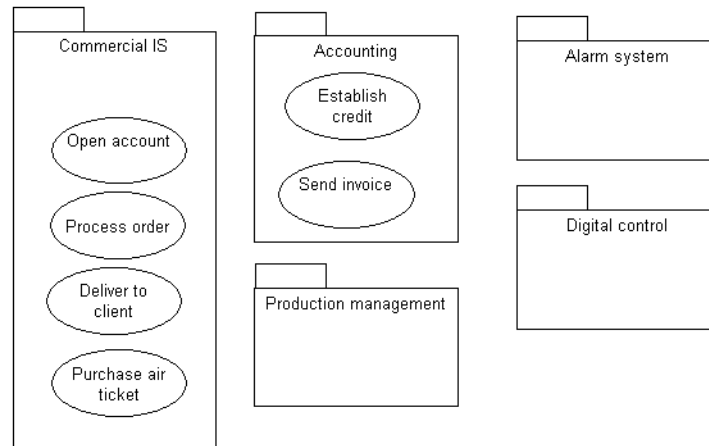


*M Zooming in on the description of an object is possible for all elements described by a diagram.*

## Packages

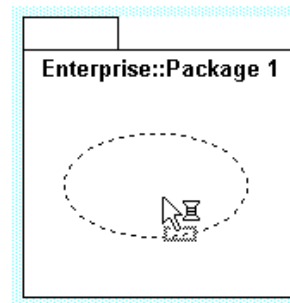
*) A package partitions the domain studied and the associated work. It enables grouping of various elements, in particular use cases and classes. A package can also contain other packages. Packages are interconnected through contractual reports defining their interface.*

Examples of *packages*: the commercial information system, accounting, production management, digital control of a machine, an alarm system, etc.



You can create a package using the **Package**  button in the toolbar. You can then increase its size in order to place use cases within it.

You can link a use case to a package simply by placing it within the package. When you have moved the object within the package, the package shape is highlighted to indicate that the object will be connected to it



- If the linked objects disappear under the package, click the package and select the **Send to Back** button in the Edit toolbar.

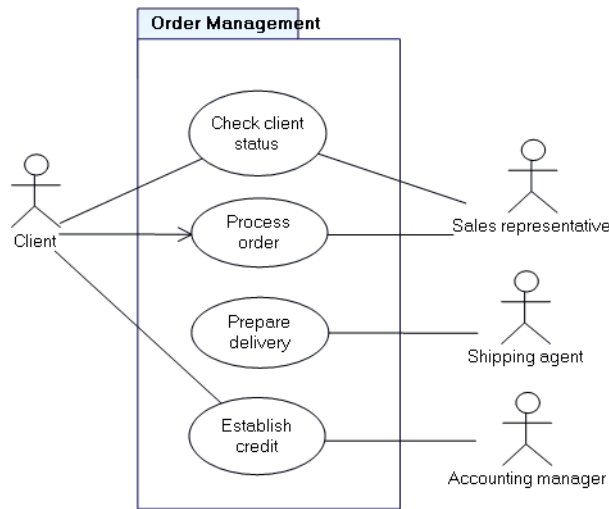
When you move a use case from one package to another using the mouse, it is unlinked from the first one and linked to the second. When you move it with the keyboard arrows, however, the links remain unchanged.

## Participations

You can indicate which actor participates in each of the different use cases.

- ) A participation indicates that an actor plays a role in a use case.


## Examples of participation



- The sales representative participates in order processing and in checking client status;
- The shipping agent participates in delivery;
- The accounting manager participates in setting up loans, etc.


## Creating participations

To create a *participation* in a use case diagram:

1. In the insert tool bar, click the **Participation** button .
2. Click the actor concerned, and drag the mouse to the use case before releasing the button.  
A dialog box appears:
3. Enter the name of the participation and indicate if the actor is the initiator.
  - It is possible to specify the beginning of the use case by selecting the **IsInitiator** check box in the properties dialog box of the corresponding participation. An arrow appears on the line representing the participation.
4. Click **OK**.

The link representing the participation appears in the diagram.

**P A participation is represented by a link, but it is in fact an object with its own properties.**

- The spool  is not used to create participations. It is used to create certain types of links, such as those between packages and other objects.
- If you make a mistake, you can delete an object by right-clicking it and selecting the **Delete** command in its pop-up menu. You can also

delete a link by right clicking on it and selecting **Delete** or **Disconnect** from the link pop-up menu.

## Multiplicities of a participation

Multiplicity can be specified on a participation:

- From the actor, to indicate that several instances of the actor participate in the same instance of the use case (example: participants in a meeting).
- From the use case, to indicate that the same instance of the actor participates in several instances of the use case (example: a sales representative processes several orders from the same customer).

To define multiplicities on a participation:

1. Select the activity concerned and display its **Properties**.
2. In the properties page that opens, click the drop-down list and select **Characteristics**.

A first section allows you to define multiplicity of the actor, a second frame that of the use case.

Having been defined, the multiplicities appear in the diagram.

---

## Associations Between Use Cases: Extension and Inclusion


When the system to be described is large, it is useful to have modeling mechanisms that can be adapted to the desired level of detail. Associations between *use cases* provide this ability.

When a use case includes too many alternatives and exceptions, these are represented separately as relationships that extend the standard use case.

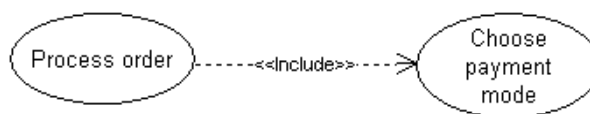
### Inclusion relationship

One use case can be called automatically following another, for example validation of an order necessarily includes selection of a means of payment.

To indicate that one use case includes another:

1. In the use case diagram, click the **Link** button .
2. Click the use case, for example "Process Order" and drag the mouse to the case used, for example "Choose Payment Mode" before releasing the mouse button.
3. Select the link of type "Uses use case" and click **OK**.

The link appears in the diagram, labeled "Include".



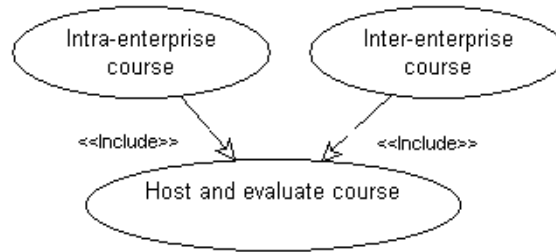
## Examples of inclusion

In a training establishment, the following use cases:

- Inter-enterprise course (where the participants come from several different companies)
- Intra-enterprise course (where the participants all come from the same company)

can both include the following use case:

- Host and evaluate the course



In a company doing direct sales, the use case:

- Place an order

can reuse the following use cases:

- Provide client information
- Place a manufacturing order
- Propose a payment method

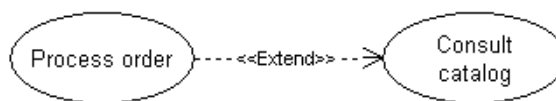
## Extend Relation

One use case can result in execution of another. Unlike inclusion, which is automatic, extension is optional.

To indicate that one use case is an extension of another:

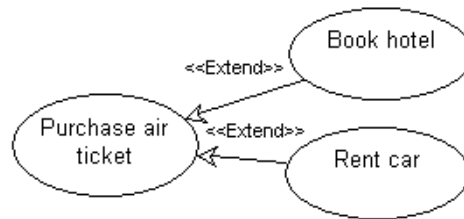
1. In the use case diagram, click the arrow associated with the **Link** button and click **Extension**.
2. Click the use case, for example "Consult Catalog" and drag the mouse to the extension case, for example "Process Order" before releasing the mouse button.  
The Creation of Extension dialog box appears. You can define a constraint or an extension location.
3. Click **OK**.

The link appears in the diagram, labeled "Extend".



### Extension example

The purchase of an airline ticket can also include booking a hotel room or renting a car.



### Extension point

The extension can intervene at a precise point in the extension case. This point is called the extension point.

To create an extension point on the extension case:

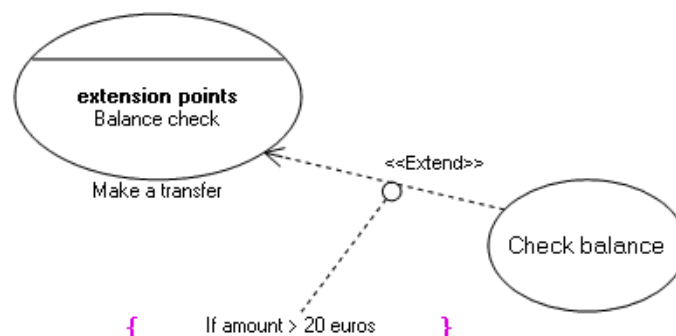
1. Open the properties window of the extension.
2. Select the **Characteristics** page.
3. In the **Extension Point** section, click **Add**.  
The query dialog box appears.
4. Select the desired extension point and click **Connect**.  
The extension point appears in the extension properties window.

An extension point can be associated with a *constraint* which indicates the moment at which the extension intervenes. You can add a constraint at creation of the extension or later, in the extension properties dialog box.

) A constraint is a declaration that establishes a restriction or business rule generally involving several classes.

### Extension point example

The following example presents the use case of a bank transfer; above a sum of 20 euros, customer credit check is triggered.

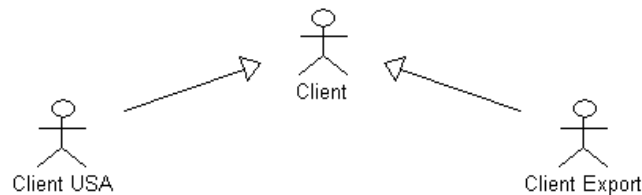


## Generalizations

) A generalization represents an inheritance relationship between a general class and a more specific class. The more specific class is fully consistent with the more general class and inherits its characteristics and behavior. However, it also contains additional information. Any instance of the more specific class is also an instance of the general class.


The concept of *generalization* was initially used for classes, but has been extended to other UML concepts such as actor and use case.

Examples of generalizations between actors:

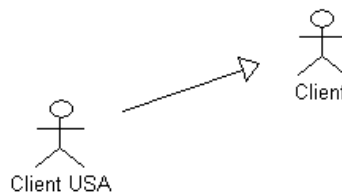


The "Client" actor can have the USA or Export specialization.

To create a generalization between actors in a use case diagram:

- > Click the  button and drag the link from the specialized actor (eg: USA client) to the more general actor (eg: Client).

The generalization then appears in the drawing.



- In the same way you can create a generalization between two use cases.


When creating a second generalization, a dialog box allows you to reuse the existing generalization if it involves the same subject.

## Interfaces

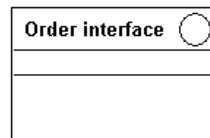
It is possible to complement the description of a use case or actor by describing the *interfaces* by which it communicates with its environment.

## Creating an Interface

To create an interface in a use case diagram:

1. In the diagram objects toolbar, click the **Interface** button. 
  - If the **Interface** button does not appear in the toolbar, select **View Views and Details** and select **Classes**.
2. Click in the diagram.
3. In the dialog box that appears, enter the name of the interface and click the **Add** button.


The interface appears in the diagram.



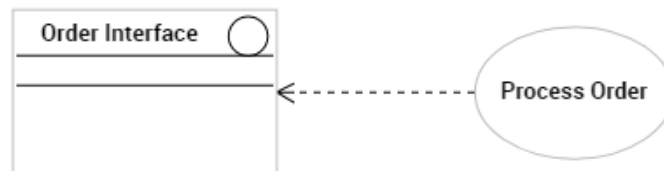
## Connecting an interface to a use case

When you connect an interface to a use case, you must specify if it is a supported interface or an interface required by the use case.

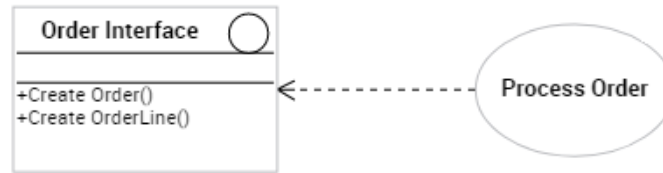
To specify the type of link between an interface and a use case:

1. Click **Connect**  and drag the link from the use case (eg: Process Order) to the interface (eg: Order Interface). A dialog box appears:
2. Indicate the type of link to be created.
  - Required interface
  - Supported interface
3. Click **OK**.

The link then appears in the diagram.



You can detail which operations the use case can carry out via this interface.



- 6 For more details on required and supported interfaces, see ["Linking interfaces to other objects", page 200](#).



# THE CLASS DIAGRAM



A class diagram is used to represent the static structure of a system, particularly the types of objects manipulated in the system, their internal structure, and the relationships between them. An object diagram provides examples to illustrate a class diagram.

The class diagram specification is often considered the most important part in the modeling of an information system. The following points are covered here:

- 6 [Presentation of the Class Diagram](#)
- 6 [Creating a Class Diagram](#)
- 6 [Classes](#)
- 6 [Attributes](#)
- 6 [Operations](#)
- 6 [Signals](#)
- 6 [Associations](#)
- 6 [Generalizations](#)
- 6 [Specifying Interfaces](#)
- 6 [Specifying Dependencies](#)
- 6 [Specifying Parameterized Classes](#)
- 6 [Constraints](#)
- 6 [Object Diagram](#)

## PRESENTATION OF THE CLASS DIAGRAM

A class diagram is used to represent the static structure of a system, particularly the types of *objects* manipulated in the system, their internal structure, and the relationships between them.

) *An object is an entity with a well-defined boundary and identity that encapsulates state and behavior. Its state is represented by the values of its attributes and its relationships with other objects. Its behavior is represented by its operations and methods. An object is an instance of a class.*

Examples of objects:

- Business objects:
  - John Williams, Elizabeth Davis and Paul Smith are instances of the "person" class.
  - Orders 10533 and 7322 are instances of the "order" class.
  - SPD-1730 Monitor is an instance of the "item" class.
- Technical objects used for programming:
  - Dlg\_Order\_Create, Dlg\_Client\_Query are instances of the window class.
  - Str\_Client\_Name, Str\_Product\_Comment are instances of the "string" class.

Data modeling consists of identifying the classes representing the activity of the company, and defining the associations existing between them.

The classes and associations comprising the class diagram associated with a business area of the company must provide a complete semantic description.

In other words, one should be able to describe the activity of a company by using only these classes and associations.

This does not mean that each word or verb used in the explanation maps corresponds directly to an object in the class diagram. It means one must be able to state what is to be expressed, using these classes and associations.

The class diagram specification is often considered the most important part in the modeling of an information system.

An object diagram provides examples to illustrate a class diagram.

In particular, it is possible to illustrate a class diagram by showing the corresponding object diagram in the same drawing.

## The Class Diagram: summary

A class diagram includes:

- Classes, which represent the basic concepts (client, account, product, etc.).
- Associations, which define the relationships between the different classes.
- Attributes, which describe the characteristics of classes and, in certain cases, of associations.
- Operations, which can be executed on objects of the class.
  - *Operations are not taken into account by **HOPEX Information Architecture** tools (synchronization, generation etc.).*

The class diagram also contains multiplicity definitions.

See the glossary at the end of this guide for definitions of these and other concepts.

---

## Creating a Class Diagram

A class diagram is created from a package.

To create a class diagram:

- > Click the icon of the package concerned and select **New > Class Diagram**.  
The diagram opens in the edit window.

A class diagram can describe a package, a use case, a class, or an instance.

# CLASSES

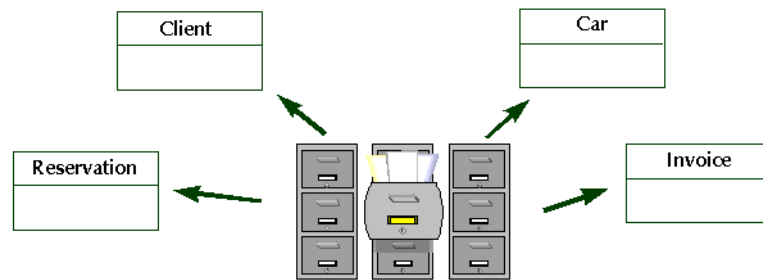
- 6 Definition: Class
- 6 Creating a Class
- 6 Class Properties
- 6 Class Stereotype

## Definition: Class

A *class* is described by a list of attributes and operations.

A class is linked to other classes via *associations*. The set of classes and associations forms the core of a class diagram.

We can illustrate the class concept by comparing classes to index cards filed in drawers.



Classes can be technical objects used for programming.

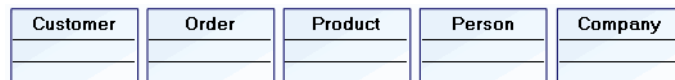
Examples: dialog box, rectangle, string, table, etc.

Classes can represent technical objects used in industry.

Examples: Alarm, Sensor, Zone

Classes can also represent business objects:

Examples: customer, order, product, person, company, etc.




A class can also express a process, such as "Confirm client request", or implement a business rule, such as "Consistency in cost accounts".

- See the glossary at the end of this guide for definitions of these and other concepts.


---

## Creating a Class

To create a class:

1. In the class diagram, click **Class**  in the insert toolbar.
2. Click in the diagram.  
The **Add Class** dialog box opens.
3. Enter the **Name** of the class and click the **Add** button.  
The class is then placed in the diagram.
  - *In the examples presented in this guide, object names may include spaces, upper case characters, and accented characters. It is important to note that if you have a generation tool using specifications created with **HOPEX UML**, and this tool is more restricted in authorized characters and name lengths, it is preferable to adhere to the more restrictive rules of the **HOPEX UML** specification.*

You can create several classes successively without needing to click on the toolbar each time. To do so, double click the **Class** button.

To return to normal mode, click the  arrow.

You can use the complete name of a class throughout by adding the name of the package to which it belongs to its name, separated by two colons.

Example:

Enterprise::Sales Management::Client.

If one of the packages in the name does not exist, it is automatically created and linked to the class.

## Finding an existing class

To find an existing class:

1. In the **Add Class** dialog box, select **List** in the drop-down list box using the arrow.  
The list of classes appears.
2. Select the desired class and click **OK**.  
The name of the selected class appears in the Add UML Class dialog box
3. Click **Add**.  
The class then appears in the drawing.

---

## Class Properties

The properties displayed depend on the class stereotype.

To open the Properties dialog box of a class:

- > Select the class concerned and click the associated **Properties** button in the edit window.  
It contains several pages where you can define the class properties.

## characteristics page

The **Characteristics** page is used to enter different characteristics of the class:

- Its **Name**, which you can modify.  
*M You can also modify the name of a class by clicking directly on the name in the drawing.*
- The owner of the class (for example, the package).
- The **Visibility** of the class as related to its package:
  - "Public": the class is visible to any element outside the package. This is the default visibility.
  - "Protected": the class is visible to elements that inherit it or have import dependencies with it, and to friends.
  - "Private": the class is only visible to elements that have import dependencies with it and to friends.
  - "Not specified".  
*) Friends of a class are the classes that are authorized to access its internals. It is possible to specify the friends of a class in the complements tab of the properties dialog box of the class.*
- Its **Stereotype** : see [Class Stereotype](#).
- **Comment**: Comments can be used to add key information to diagrams when certain details cannot be displayed in the drawing. These comments are included in the document describing the class diagram.

The other characteristics you can specify are the abstraction, persistence, and activity:

- If the class is **Abstract**, it has no instances. It is only used to group operations or attributes common to its subclasses.
- **Persistence** specifies whether the objects in the class need to exist after the process or thread that created them, or whether they only last as long as the processing.
- Instances of a class which **Is Active** are able to trigger control flows without user intervention.

*Example: An instance of the printer class can send an "Out of paper" message to the network administrator screen.*

- An **IsRoot** class is a class that has no superclasses in the tree of class generalizations.
- An **IsLeaf** class is a class that has no subclasses in the tree of class generalizations.

You can also specify the Parameters of a parameterized class (for C++).

- See [To specify a parameterized class](#): for further information.

## Other properties pages

Other pages allow you to define or view:

- Attributes of a class (see [Attributes](#))
- Operations of a class (see [Operations](#))
- Associated classes (see [Associations](#))
- Instances of a class (see [Object Diagram](#))
- Redefined elements

## Class Stereotype

A stereotype is a type of modeling element that extends the semantics of the metamodel. Stereotypes must be based on existing types or classes whose structure they use. Other stereotypes can be created by the user.

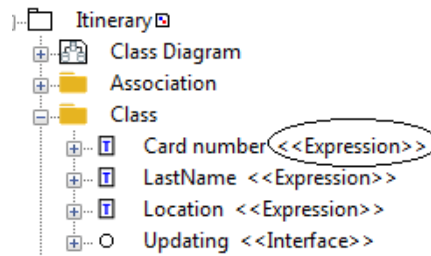
Stereotypes available for a class are:

- **Org-Unit**: represents the role played by something or someone within the enterprise environment of the modeled system.
- **Auxiliary**: class that supports another central or fundamental class, generally by implementing a secondary logic or a control flow.
- **Implementation class**: is used to characterize the classes needed for physical implementation of the system.
- **Metaclass**: class of which the instances are themselves classes. As a general rule, metaclasses are used to build metamodels.
- **Control**: is used for classes that perform processing internal to the system. These generally require contributions from several classes.
- **Entity**: enables description of classes that are passive; that is that do not initiate interactions on their own. They can participate in several use cases and generally outlive any single interaction. They represent objects shared between the different actors that handle them.
- **Enumeration**: datatype containing a list of tabulated values.
- **Expression**: expressions of complex datatypes based on types.
- **Focus**: class that defines the main logic or control flow for the auxiliary class(es) that support it.
- **Boundary**: used to describe classes that are in direct contact with the system environment. Man-machine interfaces are of this type.
- **Interface**: an interface is a named set of operations that describe the behavior of an element. In particular, an interface represents the visible part of a class or package in a contractual client-supplier type relationship.
  - *These are interfaces between the different components of the computer system. These are not interfaces with system users, as those are considered boundary stereotypes. See [Specifying Interfaces](#) for further information.*
- **Worker**: represents a human actor who interacts with the system. A worker interacts with other workers and manipulates entities while participating in use case realizations.
- **Case worker**: a case worker interacts directly with actors outside the system.
- **Internal worker**: an internal worker interacts with other workers and other entities within the system.
- **PowerType**: metatype of which instances are sub-types of another type.
- **Structure**: class that describes a structure used in the programs.
- **Thread**: stereotype used in implementation of an active object as a light business process.
- **Primitive Type**: used to describe the datatypes.

- **Utility:** a class of this stereotype groups global variables and procedures useful for programming, and described as attributes and operations of this class.
- **Schema group:** class describing a type of XML element, the sub-elements of which form a group.
- **XML Document Definition Root:** class that describes the structure of a message exchanged between two systems using the XML language syntax.

## Stereotype display option

An option allows you to display stereotypes in the navigation window of objects.



To activate this option:

1. In the **HOPEX** workspace, click **Main Menu > Parameters > Options**.
2. In the left pane of the options window, select the **Workspace** folder.
3. In the right pane, select the option **Display stereotype of UML objects in navigator**.
4. Click **OK**.

# ATTRIBUTES

- 6 Definition: Attribute
- 6 Specifying Class Attributes
- 6 Attribute Properties

## Definition: Attribute

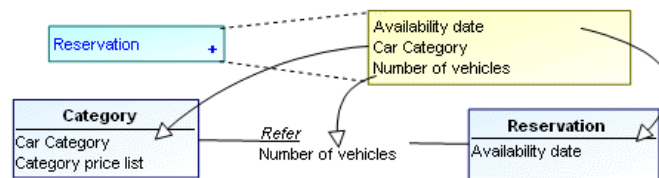
An attribute is a named property of a class. This is the most basic data saved in the enterprise information system.

Examples:

"Client Name" (attribute of the client class).

"Client No." (identifier of the client class).

"Account balance" (attribute of the account class).

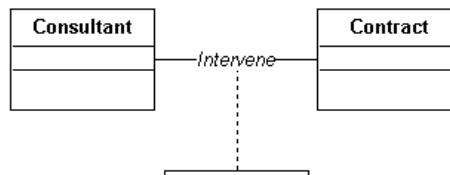


Classes and association classes may be characterized by attributes.

These attributes can be found by studying the content of messages circulating within the enterprise.

An attribute characterizes an association when its value depends on all the classes participating in this association.

In the diagram below, the "Role" that a "Consultant" plays in a "Contract" depends on the consultant and on the contract, and therefore on the "Intervene" association.



## Specifying Class Attributes

### Adding a standard attribute

To define class attributes:

1. Select the class concerned and display its properties.
2. In the properties window, click the drop-down list then **Components**.
3. In the **Attributes** section, click **Add Attribute**.  
The new attribute appears.
4. Click the name to modify it.

For each attribute, you can specify:

- Its **Type**, which can be expressed as an expression.

Example: Integer.

- The expression must comply with UML syntax. See [Operation or Signal Signatures](#) for further information.
- See also: [Attribute type](#).

- Its **Visibility**:

- "Public": this is the default visibility. The attribute is visible to all.
- "Protected": the attribute is visible to those inheriting its package, or to its friends.
- "Private": the attribute is visible to its class or to its *friends*.

) *Friends of a class are the classes that are authorized to access its internals. It is possible to specify the friends of a class in the complements tab of the properties dialog box of the class.*

- Its **Multiplicity**, which is the number of times this attribute can be repeated in the class.

### Adding a computed attribute

A computed attribute is connected to a calculation rule.

The calculation rule defines the input and output objects as well as the expression of the rule.

The input objects can be classes, types or data views. The output objects are classes only.

To define a computed attribute on a class:

1. Select the class concerned and display its properties.
2. In the properties window, click the drop-down list then **Components**.
3. In the **Attributes** section, click **Add Computed Attribute**.  
The new attribute appears.
4. Open the properties of the attribute to specify:
  - the list of input parameters
  - the description of the rule (text expression)

## Inherited attributes

When a generalization exists between a general class and a more specialized class, the specialized class inherits the attributes of the general class.

- > Click the **Inherited Attributes** button to view attributes inherited from other classes.

---

## Attribute Properties

To open the Properties window of an attribute:

1. In the properties window of the owner class, select the **Components** page.
2. In the **Attributes** section, select the attribute in question.
3. Click **Properties**.

- The  button displays the hidden commands.

In the **Characteristics** page, you can specify:

- The **Type** of the attribute in the form of an **Expression** (see [Attribute type](#)).
- Whether it is a **Static** attribute: specifies if the attribute can take specific values for each instance of the class or take one value characterizing the entire class.
  - "Yes": the attribute has a value that characterizes the entire class. The attribute "Telephone number length" for the "USA Client" class is 10 digits.
  - "No": the attribute can take a different value for each class instance. For example, the "Telephone number" attribute has a different value for each instance of the "Client" class.
- If the attribute has **Persistence**, specifying whether its value needs to exist after the process or thread that created it, or whether it only lasts as long as the processing.
- Its **Multiplicity**, which is the number of times this attribute can be repeated in the class.
- Whether it is **Read Only**, that is if its value can be modified once it has been specified.
- Whether it is a **Calculated Attribute**, specifying if its value is determined from the value of one or more other attributes.
- The **Initial Value** of the attribute, assigned when an instance of the class is created.

## Attribute type

A datatype defines the type of values that a data can have. This can be simple (whole, character, text, Boolean, date, for example) or more elaborate and composite.

Types are implemented as classes.

Any class can be used to type an attribute or parameter.

Example: Client, Order, Window, Table.

Classes of the "Primitive type" stereotype are created only for typing attributes or parameters. They are fixed.

Examples of primitive types:

String.

Integer.

Export address.

Monetary amount.

You can list the existing types or create new ones.

- *The types listed include the classes owned or used by the current package.*
- *To specify the structure of a type, place the corresponding class in the same diagram or in another diagram, and select the Properties command in its pop-up menu.*

# OPERATIONS

- 6 Definition of an Operation
- 6 Specifying Class Operations
- 6 Operation Properties
- 6 Operation or Signal Signatures
- 6 Operation Parameters
- 6 Operation Methods (opaque behavior)
- 6 Object Diagram
- 6 Operation Exceptions
- 6 Displaying Class Attributes and Operations

---

## Definition of an Operation

An operation is a service that can be requested from an object to affect a defined behavior. An operation has a signature, which may be used to specify the parameters it requires.

Examples:

"Age Calculation" (operation of the client class).

"Print" (operation of the drawing class).

"Calculate due dates" (operation of the loan class).

- Operations are not taken into account by **HOPEX Information Architecture** tools (synchronization, generation etc.).

---

## Specifying Class Operations

To specify class operations:

1. Select the class concerned and display its properties.
2. In the properties window, click the drop-down list then **Components**.
3. In the **Operations** section, cliquez **New** to create an operation or **Connect** to connect an existing operation.

The operation appears in the properties of the class.

You can specify its signature.

## Inherited operations

When a generalization exists between a general class and a more specialized class, the specialized class inherits the operations of the general class.

- › Click the **Inherited Operations** button to view operations inherited from other classes.

## Operation Properties

To open the Properties dialog box of an operation:

1. In the properties window of the owner class, select the **Components** page.
2. In the **Operations** section, select the operation in question.
3. Click **Properties**.

– The  button displays the hidden commands.

You can indicate for each operation:

- Its **Stereotype** to specify its use:
  - **Constructor**: creates an instance of the class.
  - **Destructor**: destroys an instance of the class.
  - **Iterator**: iterates through all instances of the class.
  - **Selector**: selects certain instances of the class.
- Whether it is a **Static** operation: if the operation can take specific values for each instance of the class or take one value characterizing the entire class.
- The **Concurrency**, to specify how the operation behaves when it is called several times simultaneously.
  - **Concurrency**: the operation responds simultaneously to the different calls.
  - **Protected**: the operation answers the first call and rejects ensuing ones.
  - **Sequential**: the operation responds successively to each call.
- If it is an **Is Query** operation, indicating that the object state is not modified.
- If the operation **Is Polymorphic**, to enable methods for this operation to be redefined in the subclasses.

The following indications are used to further describe the operation signature.

- The **Expression type** of the operation.
 

) *The expression type of an operation specifies the type of the variable returned by the operation on completion of its execution.*
- Its **Signature**.

## Operation or Signal Signatures

An operation or signal signature consists of the name of the operation (or signal), its return type, and its parameters with their types. Standard UML syntax is used for signatures, in the form: Ope0 (Param0: M-Bool): M-Bool.

The signature can be defined:

- Either in the properties dialog box of the operation or signal.
- Or in the Properties window of the class to which the **Operations** section.

| Operations   |            |            |
|--|------------|------------|
| <div> <span>New</span> <span>Connect</span> <span>Reorganize</span> <span>Properties</span> <span>Remove</span> </div> |            |            |
| Local name ↑   | Signature  | Visibility |
| Compute  | Compute () | Public     |
| Clear  | Clear ()   | Public     |

The saved signature includes a reference to the type. If the type is renamed, the signatures that use it will reflect this change.

## Signature syntax

The standard syntax for signatures is:

```
operationname (parameter1:typeexpression1,parameter2:typeexpression2,...):returnexpressiontype
```

Names containing spaces or special characters must be enclosed in single quotes ('Client name'). When a name contains an apostrophe, the apostrophe must be typed twice: 'Buyer's Name'

Examples of signatures:

```
Unstock (Product0: Integer(3), Quantity0: Integer): Boolean
'Create order' ('Client name' : Client): Byref Variable
```

In a signature specification, it is possible to specify the package to which a class belongs, followed by two colons.

```
Example: Enterprise::'Sales Management'::Client.
```

The listed class is linked to the parameter or return type. If it does not exist, it is created. Any packages listed in the path that do not exist are also created, and linked to the class.

If the package is not specified, a dialog box will enable you to select from similarly named classes.

---

## Operation Parameters

A parameter is the specification of a variable, which can be modified, sent or returned. A parameter can specify a name, a type and a direction. Parameters are used for operations, messages and events.

An argument is a specific value corresponding to a parameter.

In the Properties dialog box of an operation, the **Parameters** section allows you to specify:

- The operation **ExpressionType**, eg. Integer(5).
- Its **defaultValue**, eg. 0.
- Its **Direction**: at input and/or output of the operation.

To create a *parameter* on an operation:

1. Open the operation properties.
2. Select the **Characteristics** page.
3. In the **Parameters** section, click **New**.  
The dialog box for creating a parameter opens.
4. Enter the name of the parameter and click **OK**.

---

## Operation Methods (opaque behavior)

A method - or opaque behavior - is a textual representation of implementation of an operation, class or component. It specifies the algorithm or procedure that produces results of an operation or behavior of an element.

To define the method that implements an operation:

1. Open the Properties dialog box of the operation.
2. Select the **Characteristics** page.
3. In the **Method** section, click **Add**.  
The dialog box for adding a method appears.
4. Enter the name of the method to be created or search for an existing operation.
5. Click **OK**.

To enter the body of the text and the method that implements the operation:

1. Open the properties dialog box of the method.
2. Select the **Characteristics** page.
3. Define the method in the **Body** frame.

When a class has several subclasses, each subclass can perform the operation using a different method.

The **Method** section presents the method relating to the selected class.

---

## Operation Conditions

You can define operation conditions in the form of constraints.

The condition types are:

- A **PreCondition** that must be met before the operation is executed.
- The condition on the **Body** that must be checked at operation execution.
- A **PostCondition** that must be met after executing the operation.

To define a condition on an operation:

1. Open the Properties dialog box of the operation.
2. Select the **Conditions** page.

3. In the **Conditions** section, select the condition type:
  - precondition
  - condition on the body
  - postcondition
4. Click **New**.  
The dialog box for adding a restriction appears.
5. Enter the name of the restrictions to be created or search for an existing operation.
6. Click **OK**.

To enter the body for the condition:

1. In the properties window of the holding operation, select the condition.
2. Click the **Properties** button.

- The  button displays the hidden commands.

3. Click the **Characteristics** page.
4. In the **Expression Body** section, enter the expression.

## Operation Exceptions

If a condition is not respected, an exception is generated.

The **Exceptions** tab allows you to define error messages sent by the operation when an exception occurs and to specify their signature.

---

## Displaying Class Attributes and Operations

To modify how the attributes and operations for a class are displayed:

1. Right-click the class or classes whose attributes you want to display.
2. Select **Shapes and Details**.  
Use the Display dialog box to select what elements are to be displayed.
3. In the tree on the left, click **Attribute**.
4. Select the attributes you want to see displayed.

You can display **All the** attributes, **Some of the** attributes (select them from the list), or **None of the** attributes.

You can request display of the **Visibility**, **Type**, ... of each of the attributes.

) A datatype is used to group characteristics shared by several attributes. Datatypes are implemented in the form of classes.

M You can hide or show the compartment containing the class attributes in the drawing, by selecting or clearing the "Display of" check box.

Proceed in the same manner to indicate how operations are to be displayed, but instead, select **Operations** in the tree.

## SIGNALS

---

### Defining a Signal

A signal is an event that can be explicitly invoked. A signal can have parameters. A signal can be sent to an object or set of objects. It can be invoked as part of the participation of an actor in a use case.

A message can be sent or received by a class. It can also be sent by an operation after an exception.

---

### Specifying Class Signals

#### Creating a sent or received signal

To specify what signals can be sent or received by a class:

1. Select the class concerned and display its properties.
2. In the properties window, click the drop-down list and select **Complements**.
3. In the menu tree presented, select **sentSignal** or **receivedSignal** then click **Add**.
4. Indicate the name of the signal and click **OK**.

#### Signal properties

To open the properties dialog box of a signal:

- > In the properties window for a class, in the **Complements** page, select the signal and click **Properties**.

- The  button displays the hidden commands.

The Properties dialog box of the signal appears.

You can indicate for a signal:

- Its **Stereotype** to specify its use:
  - **Exception**: an error signal is generated when an exception occurs during the execution of an operation.
- Its **Visibility** related to the package:
  - **Public**: this is the default visibility. The signal is visible to any element outside the package.
  - **Protected**: the signal is visible to inherited elements or friends.
  - **Private**: the signal is visible to its class or to its friends.
    - ) *Friends of a class are the classes that are authorized to access its internals. It is possible to specify the friends of a class in the complements tab of the properties dialog box of the class.*
- The **ExpressionType** of the signal (see *expression type*)..
  - ) *The ExpressionType of a signal specifies the type of variable returned by the signal on its receipt by the addressee.*

A signal can be a request to **Vote** sent to each active object, asking if it is possible to perform a specific action such as closing a Windows session.

A signal can be a general **Broadcast** to all active objects.

## Signal parameters

The *Parameters* of the signal are specified in the **Parameters** tab of its Properties dialog box. You can specify:

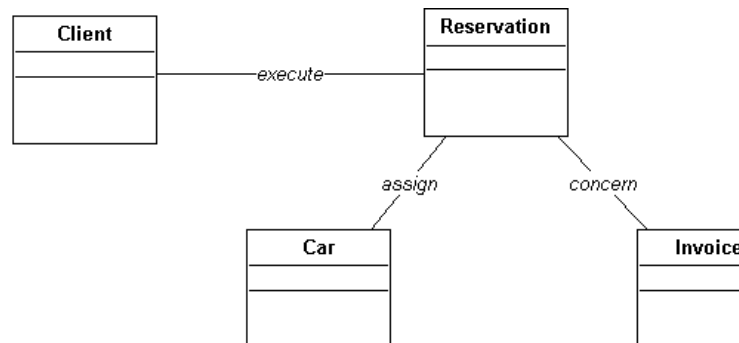
- The operation **ExpressionType**, eg. Example: Integer(5).
- Its **Default Value**. Example: 0.
- Its **Direction**: at input and/or output of the operation.
  - ) *A parameter is the specification of a variable, which can be modified, sent or returned. A parameter can specify a name, a type and a direction. Parameters are used for operations, messages and events.*
  - ) *An argument is a specific value corresponding to a parameter.*

## ASSOCIATIONS

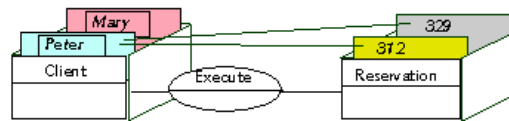
An association is a relationship existing between two classes.

An association is binary when it links two classes, ternary when it links three classes, etc.

Associations can be compared to links between index cards.



The following drawing provides a three-dimensional view of the situations a class diagram can store.



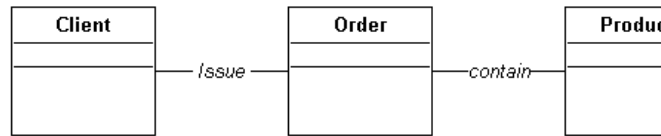
Peter and Mary are clients. Peter has made reservations numbers 312 and 329.

A class diagram should be able to store all situations in the context of the company.

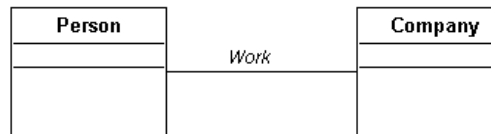
- The diagram should not allow representing unrealistic or aberrant situations.

Examples of associations:

- A client issues an order.
- An order includes several products.



- A person *works* for a company.



- An alarm is triggered by a sensor.


A sensor covers a zone.

A window displays a string of characters.

---

## Creating an Association

To create an *association*:

1. In the class diagram, click **Association**  in the objects toolbar.
  2. Click one of the classes concerned and drag the mouse to the other class before releasing the button.  
The Creation of Extension dialog box appears.
  3. Enter the name of the association to be created.
    - You can also select an existing association.
  4. Click **Add**.  
The association is indicated by a line in the diagram.
    - If you make a mistake, you can delete an element or a link by right-clicking it and selecting the Delete command in the pop-up menu.
- 

## Roles (or Association Ends)

It is possible to describe the different *roles* played by the classes in associations and to specify their multiplicity and their navigability.

Each end of an association specifies the role played by the class in the association.

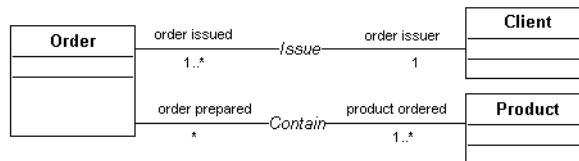
The role name is distinguished from the association name in the drawing by its position at the link end. In addition, the role name appears in a normal font, while the association name is italicized.



When two classes are linked by only one association, the name of the classes is often sufficient to describe the role. Role names are useful when several associations link the same two classes.

Examples of roles:

- A client is the *order issuer*.
- An order is *issued* by a client.
- An order is *prepared* from products.
- A product is *ordered*.



A person is an *employee* of a company.

A company is the *employer* of these persons.

An alarm is *triggered* by one or more sensors.

A zone is *covered* by a sensor.

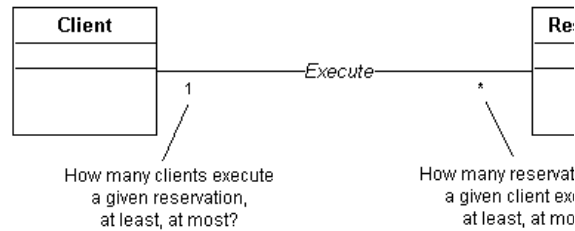
One or more strings are *displayed* in a window.

## Multiplicity of a Role

) *Multiplicity specifies the interval between minimum and maximum values of cardinalities for a set. This is primarily indicated for each role that classes play in an association. It can assume the values \*, 0..1, 1, 1..\*, 2..\*, 4..10, etc. The default value is \*.*

) *Cardinality is the number of elements contained in a set.*

The *multiplicity* expresses the minimum and maximum number of instances of a class that can be linked by the association to each instance of the other class.

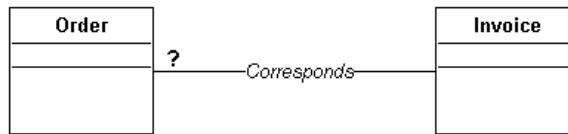


The usual multiplicities are "1", "0..1", "\*" or "0..\*", "1..\*", and "M..N" where "M" and "N" are integers:

- The "1" multiplicity indicates that one and only one instance of the class is linked by this association to each instance of the other class.
- The "0..1" multiplicity indicates that at most one instance of the class can be linked by this association to each instance of the other class.
- The "\*" or "0..\*" multiplicity indicates that any number of instances of the class can be linked by the association to each instance of the other class.
- The "1..\*" multiplicity indicates that at least one instance of the class is linked by the association to each instance of the other class.
- The "M..N" multiplicity indicates that at least M instances and at most N instances of the class are linked by the association to each instance of the other class.

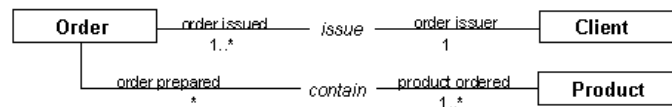
|       |                               |
|-------|-------------------------------|
| 1     | One and one only              |
| 0 / 1 | Zero or one                   |
| M..N  | From M to N (natural integer) |
| *     | From zero to several          |
| 0..*  | From zero to several          |
| 1..*  | From one to several           |

The following example illustrates the significance of the different multiplicities:



- 0..1 : An order corresponds to zero or at most one invoice.
- \* : No restriction is placed on the number of invoices corresponding to an order.
- 1: Each order has one and only one corresponding invoice.
- 1..\* : Each order has one or more corresponding invoices.

Other examples of multiplicity:



- 1..\* : A client can issue one or more orders.
- 1: An order is issued by one and only one client.
- 1..\* : An order contains one or more products.
- \* : A product can be contained in any number of orders, including no orders.
- 0..1 : A person works for a company.
- 1..\* : An alarm is triggered by one or more sensors.
- 1: A sensor covers one and only one zone.
- 1..\* : A window displays one or more strings.

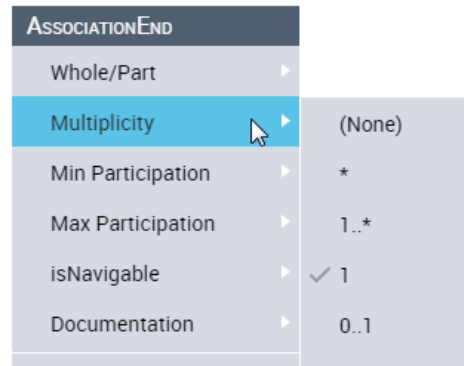
## Specifying role multiplicity

To specify association end multiplicity:

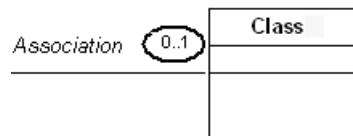
1. Right click on the part of the line of the association that is located closest to the class.

2. Select **Multiplicity** then the desired value.

- If the menu you see does not propose multiplicity, check that you clicked on that part of the line indicating the role and not the association.



The multiplicity now appears on the role.



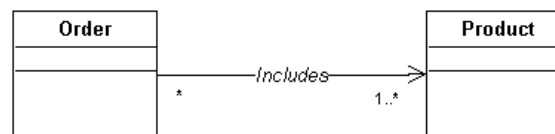
M All the information specified with the pop-up menu can also be viewed and modified in the role properties dialog box.

## Association End Navigability

IsNavigable specifies in which direction(s) an association between two classes can be traversed. To avoid crowding the drawing, this is only indicated when only one direction is possible.

Example of navigability:

- It is important to be able to find out what products are contained in an order.
- However, it is rarely useful to be able to find all orders that concern a product.



## Specifying navigability for a role

To indicate that an association is navigable in one direction only:

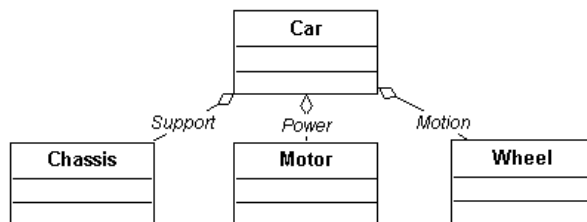
1. Right-click the non-navigable role.
2. Select **IsNavigable > No**.  
An arrow representing the navigability now appears for the other role.

---

## Association End Aggregation

Aggregation is a special form of association, indicating that one of the classes contains the other.

Example: A car includes a chassis, an engine, and wheels.



## Specifying role aggregation

To specify role aggregation:

1. Right-click the role.
2. Select **Whole/Part > Aggregate**.  
 - If the menu you see does not propose aggregation, check that you clicked on that part of the line indicating the role and not the association.

A diamond now appears on the role, representing the aggregation.

---

## Association End Composition

A composition is a strong aggregation where the lifetime of the components coincides with that of the composite. A composition is a fixed aggregation with a multiplicity of 1.

Example: An order consists of several order lines that will no longer exist if the order is deleted.

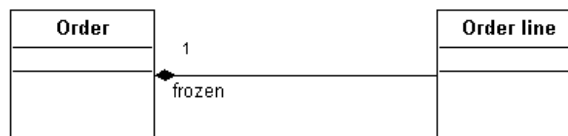
Composition is indicated by a black diamond.



## Role Changeability

**Read Only** specifies whether the role played by a class in an association may be modified after it has been created. By default, the role of a class in an association is considered changeable.

Example: An order includes an order line for each of the ordered products. These order lines can no longer be modified after the order has been saved.



You can indicate whether a role is changeable using the role pop-up menu or the role properties dialog box.

The **Read Only** characteristic of the role can have the following values:

- **Add only**: it is still possible to link new objects with this association, but already linked objects cannot be unlinked.
- **Read Only**: linked instances can no longer be unlinked. Nor is it possible to add a new link.
- **No Restriction**: new instances can be linked or unlinked at any time with no constraints.

## Role Order

It is possible to specify whether or not a role is Ordered. For example, for a client order, it can be useful to store the sequence in which its lines appear.

To specify that a role is ordered:

1. Open the **Properties** dialog box of the role.
2. In the **Characteristics** page, select the **IsOrdered** check box.

## Role Static Property

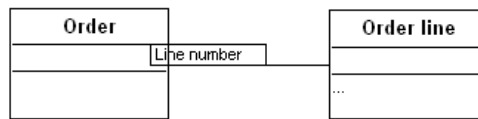
As for an attribute, it is possible to specify if a role can take specific values for each class instance, or take a value characterizing the entire class:

1. Open the properties dialog box of the role.
2. Click the **Characteristics** page.
3. In the **Static** box, select:
  - "Yes": so that the role can take a value characterizing the entire class.
  - "No": so that the role can take a different value for each class instance.

## Role Qualifier

A qualifier is an attribute whose values partition the set of objects related to an object across an association.

Example: An order includes several order lines. The order line number can be used as the qualifier that identifies each line.

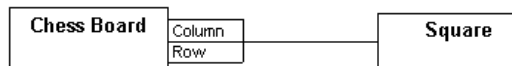


To define a qualifier:

1. Right-click the role and select **Properties**.  
The Properties dialog box of the role opens.
2. Select the **Qualifiers** page.
3. To add a new qualifier to the role, click **Add**.
4. Enter the name of the qualifier.
5. Click **Add**.

Several qualifiers may be needed to uniquely identify each object in a class.

For example, each square on a chessboard is identified by its row number and column number on the chessboard.



## Overloading a Role

A role can inherit a role defined at higher level. Overloading enables definition of additional properties on an inherited role.

To overload a role:

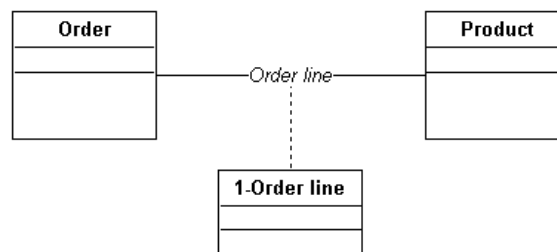
1. Open the properties dialog box of the role.
2. In the properties window, click the drop-down list and select **Characteristics**.
3. In the **Roles** section, select **Overloaded Role**.
4. Click **Add**.  
The Query dialog box appears:
5. Search and select the role in question.
6. Click **OK**.

## Association Classes

An association class is an association that also has class properties as attributes.

It is helpful to create an association class in order to specify the characteristics of an association.

For example, the quantity of the requested product needs to be specified on each order line.



To create an association class:

1. Create a new class.
2. Using the **Link** button, create a link between the class and the association.

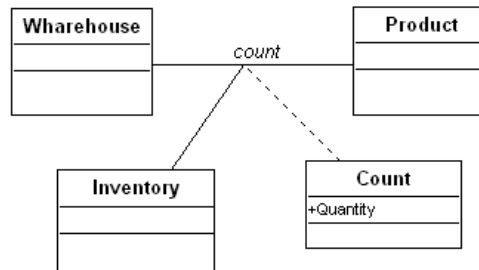
The association class is linked to the association by a dotted line.

- As for standard classes, it is possible to hide the compartments and resize the association class using the Display command in its pop-up menu.


## Displaying an N-ary Association

Certain associations associate more than two classes. These associations are generally rare.

Example: When taking inventory, a certain quantity of product was counted in each warehouse.

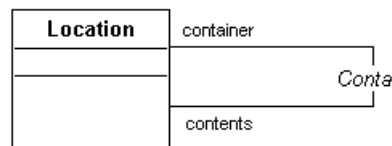


To create a ternary association:

1. First create the association between the two classes.
2. Click the **Association Role** button .
3. Draw a link between the association and the third class.

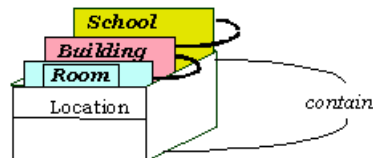
You can then proceed as described above to create an association class if needed.

## Reflexive Associations



Certain associations use the same class several times.

A classroom, a building, and a school are all locations.




A classroom is contained in a building, which is contained in a school.

A reflexive association concerns the same class at each end.

## Creating a reflexive association

To create a reflexive association:

1. Click the **Association** button  in the toolbar:
  2. Click on the class concerned and drag the mouse outside the class, then return to it and release the mouse button.
- The reflexive association appears in the form of a half-circle.

- *If there is an association of a class to itself, the roles need to be named in order to distinguish between the corresponding links in the drawing.*

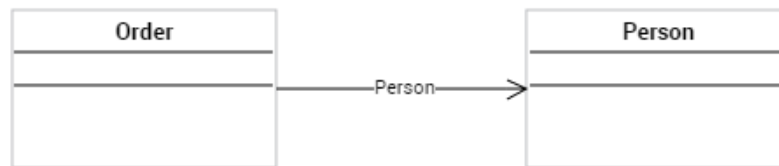
## THE PARTS

In a class diagram, a part represents a role played by an instance of a class or component at execution of a task.

A part belongs to a class. Ownership is specified on the link of the part.

In the example below, the "Order" class comprises the "Person" class.

The part is owned by the "Order" class and references the "Person" class.




---

### Creating a Part between two Classes

A part is a directional link that connects two classes only.

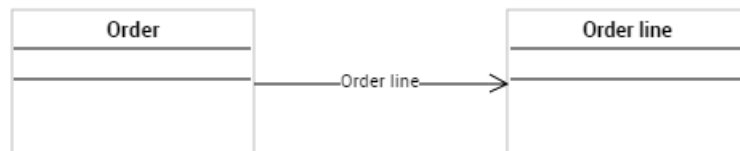
To build an part between two classes:

1. In the objects toolbar of the class diagram, click **Part**.
2. Draw a link from the owner class to the referenced class.  
The name of the part is automatically defined.

---

### Defining the Identifier of a Class via a Part

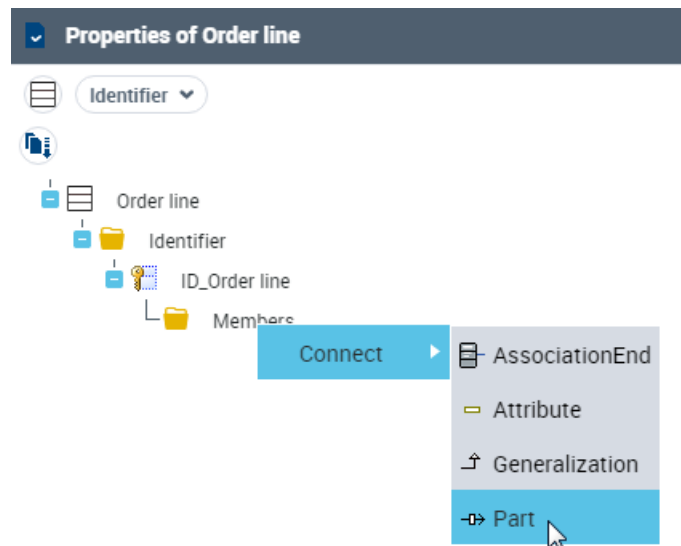
In the example below, the identifier of the "Order line" class can be defined from the "Order" class through the "Order line" part.



To define the identifier of the "Order line" class:

1. Display the properties of the "Order line" class.
2. Select the **Identifier** page.

3. Right-click the **Members** folder and select **Connect > Part**.



4. Select the proposed part.
5. Click **OK**.

## Multiplicities of the Associated Classes

With multiplicities you can specify the minimum and maximum number of instances linked by the part.

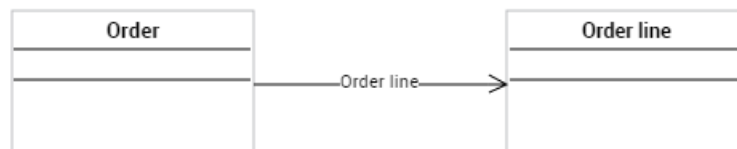
Example: 1 order comprises 1 or several order line(s).

### Multiplicity of the class referenced by the part

The multiplicity of the referenced class must be indicated on the part link.

To define the multiplicity of the referenced class:

1. Right-click the part link.

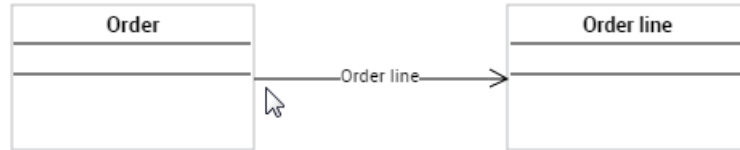


2. Select **Multiplicity** then the desired value.

## Multiplicity of the owner class of the part

To define the multiplicity of the owner class of the part:

1. Right-click the part role associated with the owner class.



2. In the pop-up menu that appears, select **Multiplicity** then the desired value.

---

## Aggregation and Composition Relationships

On the part that links two classes, you can define an aggregation or composition relationship.

) *Aggregation is a special form of association, indicating that one of the entities contains the other.*


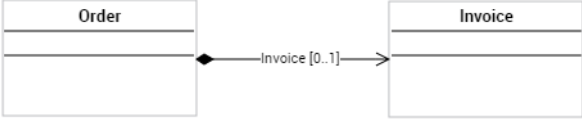

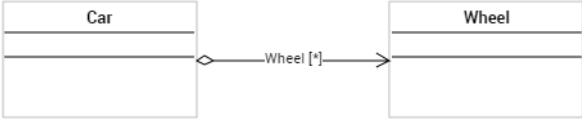

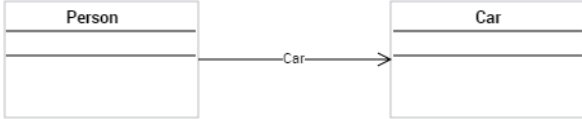
) *A composition is a strong aggregation where the lifetime of the components coincides with that of the composite. A composition is a fixed aggregation with a multiplicity of 1.*

To define a composition or an aggregation link between two classes:

1. Right-click the part.
2. Select **Whole/Part** then the desired value:
  - **Aggregate**
  - **Composite**

## Associated multiplicities

The following table presents the multiplicities automatically associated with aggregations and compositions.

|             |   | Corre-<br>sponding<br>multiplicity | Example  |
|-------------|---|------------------------------------|--|
| Composition |    | 1                                  |  <p>1, 1..*</p>    |
| Aggregation |    | 0 / 1                              |  <p>0..1, 1..*</p> |
| None        |  | *                                  |  <p>*, *</p>     |

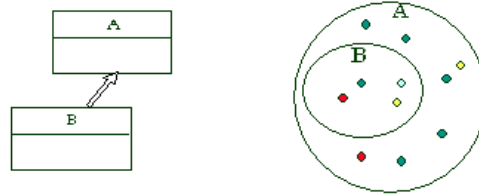
## GENERALIZATIONS

A generalization represents an inheritance relationship between a general class and a more specific class. The more specific class is fully consistent with the more general class and inherits its characteristics and behavior. However, it also contains additional information. Any instance of the more specific class is also an instance of the general class.

- 6 [What is a Generalization?](#)
- 6 [Multiple Subclasses](#)
- 6 [Advantages of Subclasses](#)
- 6 [Multiple Inheritance](#)
- 6 [Creating a generalization](#)
- 6 [Discriminator](#)

---

### What is a Generalization?



Class A is a generalization of class B. This implies that all objects in class B are also objects in class A. In other words, B is a subset of A.

B is then the subclass and A the superclass.

Example A: Person, B: Bostonian.

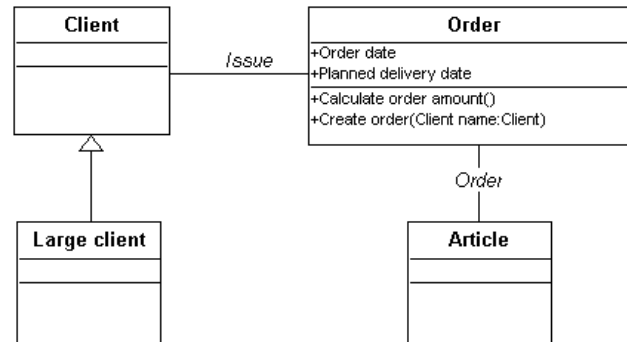
B is a subset of A, so the objects in class B inherit the characteristics of those in class A.

It is therefore unnecessary to redescribe for class B:

- Its attributes
- Its operations
- Its associations

## Example

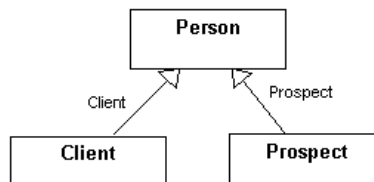
The "Large Client" class, representing Clients with a 12-month revenue exceeding \$1 million, can be a specialization of the Client class (origin).



In the above example, the associations and attributes specified for "Client" are also valid for "Large client".

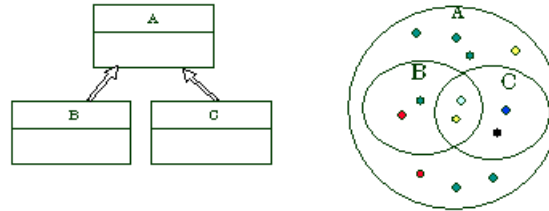
Other examples of generalizations:

- Prospect and client are two subclasses of "person".



- Export order is a subclass of the "order" class.
- Individual person and corporate person are two subclasses of the "person" class.
- Polygon, ellipse, and circle are subclasses of the "shape" class.
- Oak, elm, and birch are subclasses of the "tree" class.
- Motor vehicle, all-terrain vehicle, and amphibious vehicle are subclasses of the "vehicle" class.
- Truck is a subclass of the "motor vehicle" class.

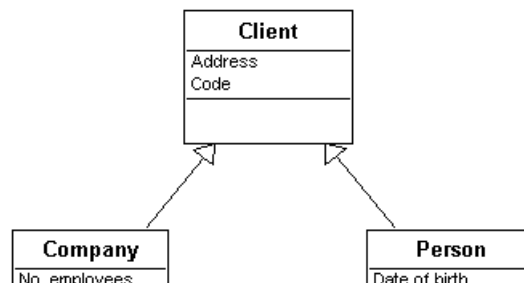
## Multiple Subclasses



When a class has multiple subclasses, they:

- are not necessarily exclusive.
- do not necessarily partition the set.

## Advantages of Subclasses

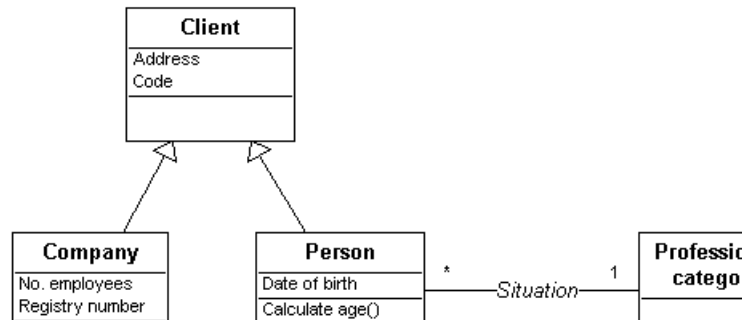


A subclass inherits all the attributes, operations, and associations of its superclass, but can have its own attributes or associations that the superclass does not have.

A subclass can also have specific attributes. These only have meaning for that particular subclass. In the above example:

- "Registry number" and "number of employees" only have meaning for a "company".
- "Date of birth" is a characteristic of a "person", not a "company".
- It is also useful to calculate the "age" of a "person". This attribute and this operation are generally not needed for a "company".

A subclass can also have specific *associations*.



- A "person" falls into a "socio-professional group": "manager", "employee", "shop keeper", "grower", etc. This classification makes no sense for a "company". There is also a classification for companies, but it differs from that for persons.


## Multiple Inheritance

It is sometimes useful to specify that a class has several superclasses. The subclass inherits all the characteristics of both superclasses. This possibility should be used carefully.

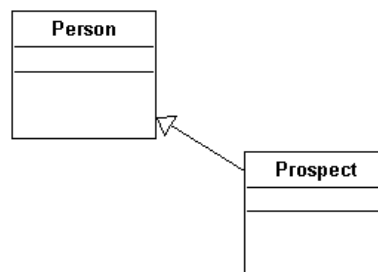
- *Multiple inheritance is not taken into account when generating tables.*

## Creating a generalization

To create a generalization:

1. Click the **Generalization** button  in the toolbar.
2. Click the subclass concerned, and drag the mouse to the superclass before releasing the button.

The generalization is now indicated in the diagram by an arrow.

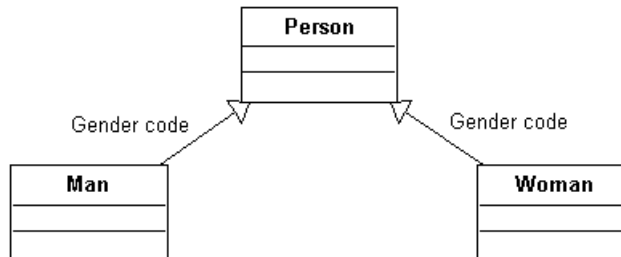


---

## Discriminator

The discriminator is the attribute of a generalization, the value of which distributes objects into the sub-classes associated with the generalization.

For example, the gender code attribute divides the objects in the person class into the man and woman subclasses.



You can define discriminator(s) in the generalization properties dialog box, under the **Discriminators** page.

- It is also possible to specify whether a generalization:

*Is Disjoint:* An instance cannot belong to two subclasses of the generalization simultaneously.

*Is Complete:* All instances of the superclass belong to at least one of the subclasses of this generalization.

## SPECIFYING INTERFACES


An interface represents the visible part of a class or package in a contractual client-supplier type relationship. The interface is a class stereotype.

An interface is a named set of operations that describe the behavior of an element. In particular, an interface represents the visible part of a class or package in a contractual client-supplier type relationship.

These are interfaces between the different components of the computer system.

### Creating an Interface

To create an interface class in a class diagram:

1. In the toolbar, select **Interface** .
2. Click in the diagram.
3. In the dialog box that appears, enter the name of the interface and click the **Add** button.  
The interface class then appears in the diagram.

You can then specify the operations of the interface as for any other class.


### Connecting an interface to a class

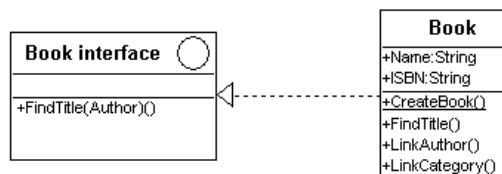
When you connect a class to an interface, you must specify if it is an interface required or provided by the class.

A required interface is an interface necessary for object operation.

A provided interface is an interface made available by an object to other objects.


To connect an interface to a class:


1. Click the **Link** button .
2. Create the link from the class to the interface.  
A dialog box appears:
3. Indicate the type of link to be created: provided interface or required interface.  
- *Other types of links, specific to classes, can be displayed.*
4. Click **OK**.



## SPECIFYING DEPENDENCIES

In the class diagram, to indicate that a package references a class or another package:

1. Click the **Link** button 
2. Then carry out the link from a package to the package or class that it references.

- The **Views**  button allows you to specify the buttons that you want to appear in the objects toolbar.

## SPECIFYING PARAMETERIZED CLASSES

A parameterized class enables definition of characteristics and a behavior that varies as a function of the value of certain parameters. For example, a parameterized class can be used to manage object lists. In this case the parameter will be the object type to be managed in the form of a list. This type of class is implemented in particular in C++ language.


To specify a parameterized class:

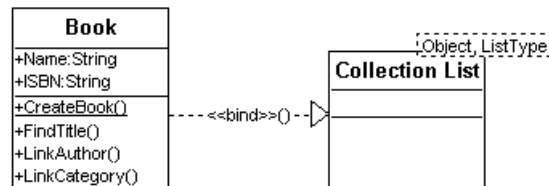
1. Open the **Properties** window of the class and select the **Characteristics** page.
2. You can enter the parameters and specify their type if necessary.

TemplateParameters: Object, ListType

The class parameters are displayed at top right.

To link a class to a parameterized class:

1. Click the **Link** button 
2. Create the link from the class to the parameterized class.



## CONSTRAINTS

A constraint is a declaration that establishes a restriction or business rule generally involving several classes.



Most constraints involve associations between classes.

Examples of constraints:

- The person in charge of a department must belong to the department.
- Any invoiced order must already have been delivered.
- The delivery date must be later than the order date.

A sensor covering a zone can trigger an alarm for that zone only.

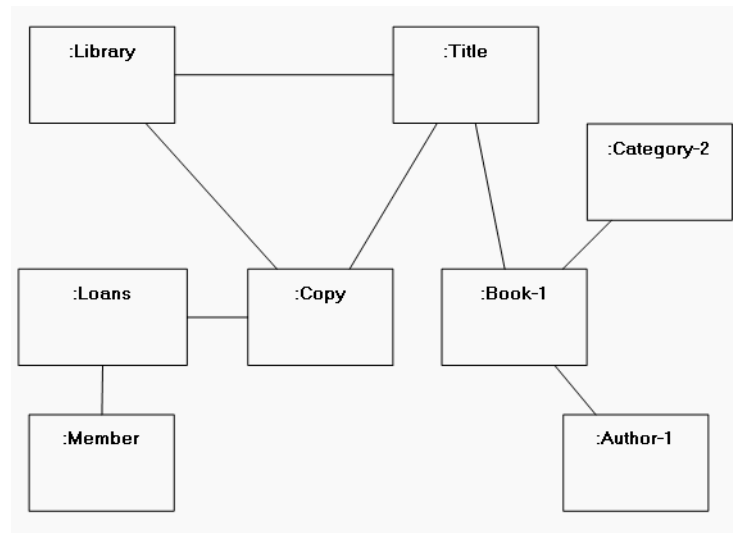
To create a constraint in the class diagram:

1. Click the **Constraint** button  in the object toolbar.
  - If it is not displayed, select **View Views and Details** and select the "Constraints" check box.
2. Then click one of the associations concerned by the constraint, and drag the mouse to the second association before releasing the mouse button. The Add Constraint dialog box appears.
3. Enter the name of the constraint, then click **Add**. The constraint then appears in the drawing.
  - You can link a constraint to other classes or associations using the **Link** button .

## OBJECT DIAGRAM

An object diagram or instance diagram contains objects with values illustrating their attributes and links. It shows in detail the state of the system at a given moment.

You can create the object diagram of a class, component, package or use case.



---

### Objects


An object is an entity with a well-defined boundary and identity that encapsulates state and behavior. Its state is represented by the values of its attributes and its relationships with other objects. Its behavior is represented by its operations and methods. An object is an instance of a class.

Examples of objects:

- Business objects:
    - John Williams, Elizabeth Davis, Paul Smith are instances of the person class.
    - Orders 10533 and 7322 are instances of the order class.
    - Sony SPD-1730 Monitor, Compaq Deskpro 200 are instances of the item class.
    - Dupont and Burger King are instances of the company class.
  - Technical objects used for programming:
    - Dlg\_Order\_Create, Dlg\_Client\_Query are instances of the window class.
    - Str\_Client\_Name, Str\_Product\_Comment are instances of the string class.
- The objects represented in an object diagram can be instances of a class, package, use case, component, or node, to enable defining sequence diagrams at the desired level of detail.*

## Creating an object (instance)

To create an object:

1. Click the **Instance**  button.  
You can create objects of different types. The arrow at the right of the button offers a shortcut to Class and Component object types, the most frequently used.
2. Then click in the diagram work area.  
The window for adding an instance opens.
3. Enter the instance **Name**.
4. Specify the **Instance Type** if necessary.
5. Click **Add**.

The instance appears in the diagram.

## Instance properties

To open the properties dialog box of an instance:

- > Select the instance in question and click **Properties** in the edit window if it is not activated.

It contains several pages where you can define the properties of an instance.

In the **Characteristics** page, you can:

- Select the **Instance kind** (Actor, Class, etc.).
- You can specify of which **Class, Actor**, etc. this object is an instance.
- Indicate a name for this instance.
- Specify its **Stereotype**.

## Value of an attribute

To specify the value of an attribute:

1. Display the properties of the instance of the class that contains the attribute.
2. Select the **Attributes** page.
3. In the corresponding column, indicate the value of the attribute. You can specify an instanced value or a constant value.
  - **Instanced value**: click in this column to display the list of possible instances for the selected attribute. These are variable values.
  - **Value**: click in the column and enter the value of the attribute.

---

## Links


A link represents an instance of an association between two objects.

Examples of links between objects:

- Order no. 10733 was placed by John Williams.
- Order no. 10733 includes the products Sony SPD-1730 Monitor and Compaq Deskpro 200.
- John Williams works for Dupont.
- The window Dlg\_Client\_Query displays the string Str\_Client\_Name.

## Creating a link

To create a link:

1. Click the **Link** button  in the diagram toolbar.
2. Click one of the objects concerned, and drag the mouse to the second object before releasing the mouse button.

The link then appears in the diagram.

If there is already a link between the two objects, a dialog box asks you to choose an existing link or create a new one.

## Link properties

To open the properties dialog box of a link:

- › Select the center of the link to display its **Properties**.
  - *If you do not click on the center of the link, the properties dialog box for one of the roles will be displayed.*

Under the **Characteristics** page, you can specify:

- The **Name** of the link.
- The link **Stereotype**.
- The **Association** corresponding to the link.
- The **Visibility** of the link.
- The **Package** containing the link.

In the **Link Role** page:

- For each **Instance** connected by this link, the name of the **Role** and its **Multiplicity**.
  - *Only the associations between the classes of the two instances are listed.*

## Role properties

To open the properties dialog box of a role:

1. In the properties window of a link, select the **Link Role** page.
2. Select the role in question and click **Properties**

- The  button displays the hidden commands.

The Properties dialog box of the role opens.

In this dialog box you can specify:

- A **Name** for the role.
- The **Role** for this instance.
- The **Multiplicity** for the role.
- The values for the role *Qualifiers*, defined at the class level.



# STRUCTURE AND DEPLOYMENT DIAGRAMS



In addition to class and object diagrams, structural diagrams include:

- 6 [The Package Diagram](#), enabling organization of elements of the model
- 6 [The Component Diagram](#), highlighting dependency relationships between components
- 6 [Composite Structure Diagram](#), describing interactions between components and their parts

## THE PACKAGE DIAGRAM

A package diagram enables organization of modeling elements, in order to partition the work involved in specification and development.

An element should only appear in a single package.

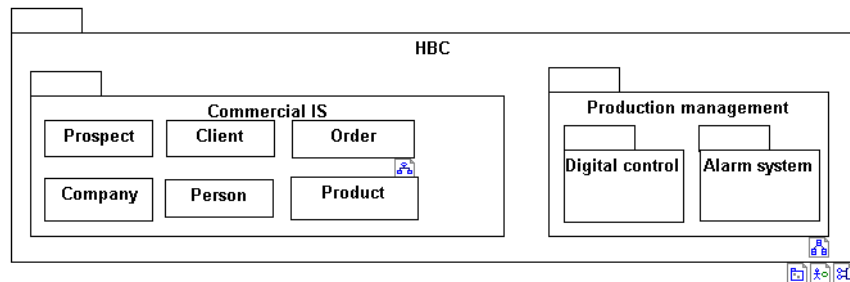
Dividing into packages is generally carried out so as to minimize interactions between different packages.

### ***Example of a package diagram***

The "HBC" package contains the "Commercial IS" and "Production Management" packages.

The "Production Management" package can be divided into two packages, "Digital Control" and "Alarm System".

The "Commercial IS" package contains the "Prospect", "Client", "Company", "Person", "Order", and "Product" classes.



## Creating a Package Diagram

A package diagram is created from a package.

To create a package diagram in **HOPEX Application Design** :

1. Click the navigation menu, then **Application Design Resource**.
2. In the navigation pane, select **OO Implementation (UML)**.
3. In the edit area window, click the **Packages** tile.
4. Display all packages.
5. Right-click the name of the package concerned and select **New > Package Diagram**.

The diagram opens in the edit window.

---


## Defining Packages

A package partitions the domain studied and the associated work. It enables grouping of various elements, in particular use cases and classes. A package can also contain other packages. Packages are interconnected through contractual reports defining their interface.

Examples of *packages*:

- The commercial information system.
- Accounting.
- Production management.
- Digital control of a machine.
- Inventory management.
- Alarm system and telephone management.

To add in the diagram an existing package:

1. In the package diagram, click the **Package** button in the object toolbar, then click the workspace.
2. In the **Add Package** dialog box, select **List** in the drop-down list box using the arrow  .  
The list of packages appears:
3. Select the desired package and click **OK**.  
The name of the diagram appears in the **Add UML Package** dialog box.
4. Click **Add**.


The package appears in the diagram.

---

## Defining Classes

The package diagram can be used to place classes in different packages.

To quickly add a set of classes to the package diagram:

1. Click **Main Menu > Advanced Query** to open the Query assistant.
2. In the wizard, select the "Class" metaclass and click **Query**  .  
The list of repository classes appears.
3. Select the classes you want and drop them in the diagram.

---

## Specifying Dependencies in a Package Diagram

Links allow you to indicate if a package contains or references a class or another package.

To indicate that a package references a class or another package:

1. Click the  button.

2. Then carry out the link from a package to the package or class that it references.  
A dialog box asks you the type of link to be created.
3. Select "Referenced package" or "Referenced class" as required.

## THE COMPONENT DIAGRAM

A *component diagram* shows the interdependency of software components and *interfaces* (it defines who uses what).

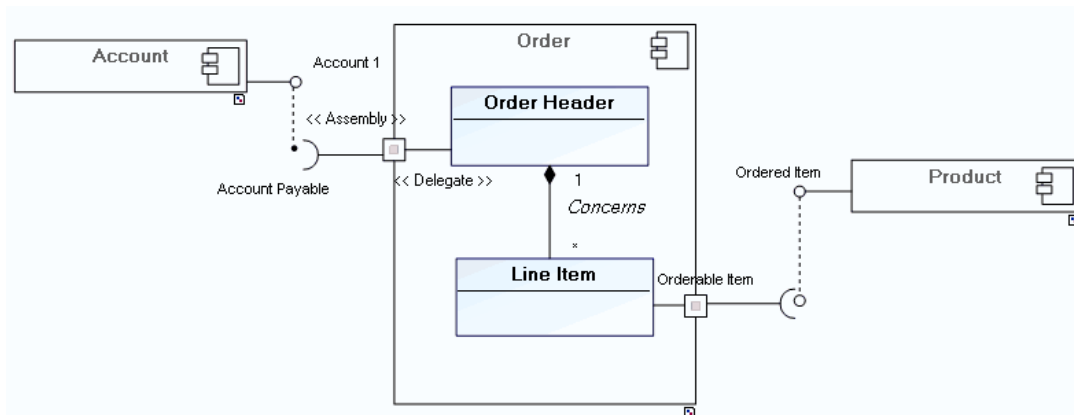
) A component is an implementation element of the system: it can be software, a program, a code element, or a physical element such as a work document.

) An interface represents the visible part of a class or package in a contractual client-supplier type relationship. The interface is a class stereotype.

A component diagram contains components and classes of the "Interface" stereotype. It is also possible to specify packages implemented by the components.

### Example of a component diagram

This diagram describes the elements contained in the "Order" component and the interactions of these elements with external components.



## Creating a Component Diagram

In **HOPEX Application Design**, you can create a component diagram using a component or package.

To create a component diagram from a package:

1. Click the navigation menu, then **Application Design Resource**.
2. In the navigation pane, select **OO Implementation (UML)**.
3. In the edit area window, click the **Packages** tile.
4. Display all packages.
5. Right-click the name of the package concerned and select **New > Component Diagram**.

The diagram appears in the edit window.


---

## Components

A component represents a modular part of a system that encapsulates its content, and which can be replaced in its environment. A component defines its behavior by means of interfaces that it provides and requires.

One component can be replaced by another if their interfaces conform.

A component can be a software package, program, code unit, etc.

It is represented by the following icon: 

---


## Interfaces

### Creating component interfaces

An interface represents the visible part of a class or package in a contractual client-supplier type relationship.

The interface is a particular type of class.

To create a class of "Interface" stereotype in the composite structure diagram:

1. Click the **Interface** button , then click in the diagram.
2. In the dialog box that appears, enter the name of the class.
3. Click **Add**.

– You can specify the details for the interface in terms of attributes and operations in the class diagram in the same way as for a class.

### Linking interfaces to other objects

Two link types enable differentiation of required interfaces and provided interfaces.

A required interface is an interface necessary for object operation.


Example: the "Purchasing Management" component requires the "Product" interface for its operation to be able to associate a purchase order with products ordered.

A provided interface is an interface made available by an object to other objects.

Example: the "Product Management" component makes available the "Product" interface.

You can define interfaces required and provided by an object independently of other objects.

To specify that an interface is supported or required by an object:

1. Click the **Connect**  button and drag the link from the object to the interface.  
A dialog box appears:

2. Indicate the type of link to be created.
  - Required interface
  - Supported interface
3. Click **OK**.

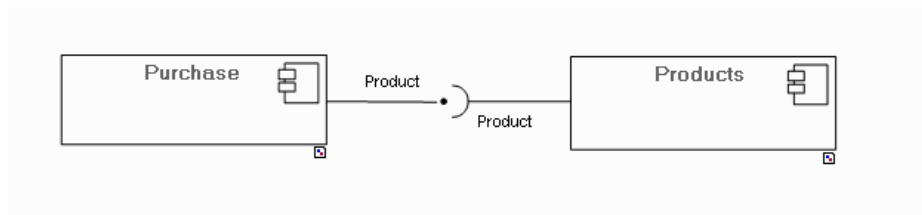
The link then appears in the diagram.

The interface shape differs according to link type:

## Connecting interfaces

Two interfaces can be interconnected. This connection is modeled by a connector.

You can also indicate that an interface provided by an object is required by another. Here it is one and the same interface.



## Ports

Ports enable connection of a component to its parts or to its environment.

Ports are represented by a square in the diagram, placed at the edge of the described element when they assure connection with the exterior.

They are connected to components by connectors.

Ports can specify queries sent and services provided by the component, as well as queries and services they may require from other parts of the system. These queries and services are represented by classes of interface type.

You can view interfaces associated with a port in the properties dialog box of the port, in the **Provided and Required Interfaces** tab.

## Connectors

Connectors enable connection of diagram objects.

Connectors of simple type do not specify a particular connection type, they are notably used to connect instances of objects described in collaborations.

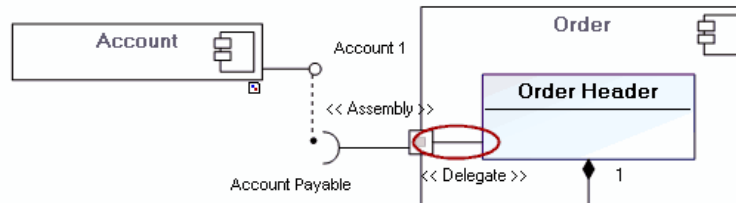
In the composite structure diagram, it is possible to specify the type of connector between two objects: Assembly or Delegate.

## Delegate connector

A "Delegate" type connector indicates the redirection of queries to a component element responsible for their execution.

The delegation link can be made directly between the component port and the component element, or between the component port and the element port.

Below, the "Order" component delegates management of accounts to be debited to the "Order Header" class.



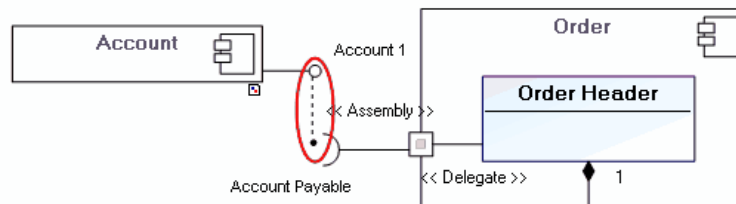
## Assembly connector

An "Assembly" type connector is a connector between two or more components or ports indicating that one or more components provide services that others use.

- These can be other objects or components.

To connect ports or components that share an interface, you can also use "Provided Interface" and "Required Interface" links.

An "Assembly" type connector connects the interface provided by the "Account" component to the interface required by the "Order Header" class.



## COMPOSITE STRUCTURE DIAGRAM

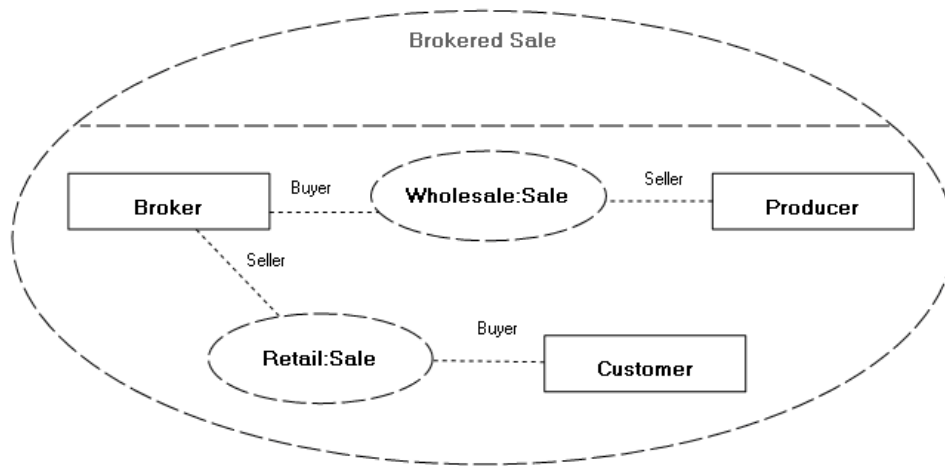
The composite structure diagram enables description of the internal structure of a component, a package or a structured class.

It also enables specification of collaborations that intervene between elements of the structure in execution of a task, highlighting the role played by each element in the collaborations.

Elements of this diagram are *parts*, ports by which parts interact with the exterior, and connectors linking the parts between themselves and with the ports.

### Example of a composite structure diagram

This diagram describes the role played by parts in the "Brokered Sale" collaboration.



## Creating a Composite Structure Diagram

To create a component diagram from a component:

1. In **HOPEX Application Design**, click the navigation menu, then **Application Design Resource**.
2. In the navigation pane, select **OO Implementation (UML)**.
3. In the edit window, click the **Components** tile.
4. Right-click the name of the component concerned and select **New > Component Diagram**.

The diagram appears in the edit window.

---

## Parts

A part represents a role played by an instance of a class or component at execution of a task.

Parts are interconnected by connectors or dependencies.

A part can also be connected, via a connector, to a port which acts as interface between the described component and the exterior.

For more details on these elements, see:

- 6 ["Connectors", page 201](#)
- 6 ["Dependency links", page 205](#)
- 6 ["Ports", page 201.](#)

Multiplicities defined on parts indicate the number of instances created. Multiplicities on connector roles indicate the number of links that can be created for each of these instances.

To define multiplicity of a part:

1. Open the properties dialog box of the part.
2. Select the **Characteristics** page.
3. Click the arrow in the **Multiplicity** box and select the required multiplicity.
4. Click **OK**.

---

## Collaborations

In the composite structure diagram, a *collaboration* describes the role played by each part (instance) in execution of a task.

*) A collaboration (UML) describes a collaborative structure between several elements (roles), each accomplishing a specialized function and collectively producing an expected functionality of the system. Its objective is to show how a system functions independently of a specific use. We therefore generally remove the precise identity of the participating classes or instances.*

It is represented by a dotted line oval containing the collaboration instances.

These instances are interconnected by *connectors*. The role that corresponds to the instance name is displayed at each end of the connector.

*) A connector is a link used to establish communication between several objects. A delegation connector links the external contract of the object (as specified by its ports and/or inters) to internal objects that execute it. An assembly connector between a number of objects (or their ports) specifies how one of the objects supplies the interface required by another.*

The model of a collaboration can be applied to different instances.

## Collaboration use

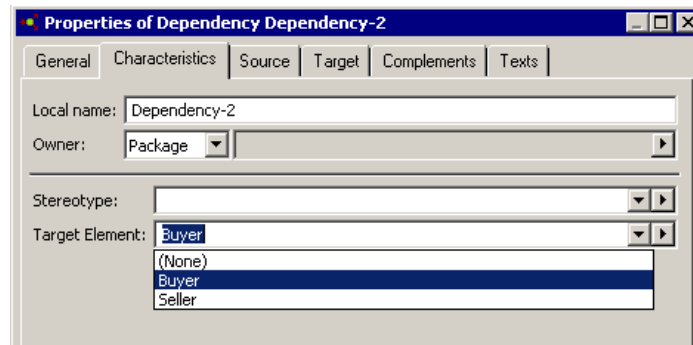
A collaboration use represents application of the structure described by a collaboration to a particular situation implementing classes or specific instances. These classes or instances therefore play roles defined in the collaboration.

The instances are connected to the collaboration use by a *dependency* link on which the role played by the instance must be specified.

) A dependency specifies that the implementation or operation of one or more elements requires the presence of one or more other elements. There are several dependency stereotypes.

## Collaboration use example

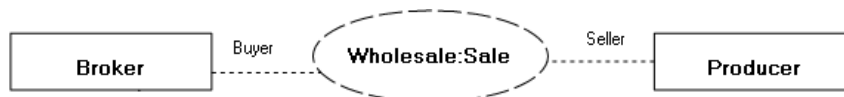
In the case of a purchasing request between two instances of an actor, a collaboration is used. This collaboration connects two roles: the role of buyer and the role of seller. On the dependency that connects each instance to the collaboration, you can indicate the role played by the instance.



## Dependency links

A dependency specifies that implementation or operation of one or several elements requires the presence of one or several other elements.

A dependency is a supplier/customer type relationship indicating source and target elements in the collaboration.




A stereotype on the dependency enables specification of dependency type:

- **Binding**: relationship between a template and a modeling element generated from the template. It includes a list of arguments corresponding with template parameters.
- **Derive** : indicates a derivation relationship between modeling elements that are generally, but not necessarily, of the same type. Such a dependency relationship implies that one of the elements can be calculated from the other.
- **UML/XML Mapping** : a mapping expression that defines the relationship between elements (classes, attributes, etc.) of a schema or class diagram and those of another schema or class diagram.
- **Refine**: specifies a dependency relationship between modeling elements at different semantic levels, such as analysis and design.
- **Trace**: specifies a traceability relationship between modeling elements or sets of modeling elements that represent the same concept in different models.

To specify dependency type:

1. Open the properties dialog box of the dependency.
2. Select the **Characteristics** page.
3. In the **Stereotype** box drop-down list, select one of the proposed stereotypes.

The arrow  also allows you to create new stereotypes.

# STATE MACHINE DIAGRAM



A state machine diagram enables description of possible behaviors of an object, depending on the events it experiences during its life cycle.

The following points are covered here:

- 6 [Presentation of the State Machine Diagram](#)
- 6 [Creating a State Machine Diagram](#)
- 6 [States](#)
- 6 [State Transitions](#)

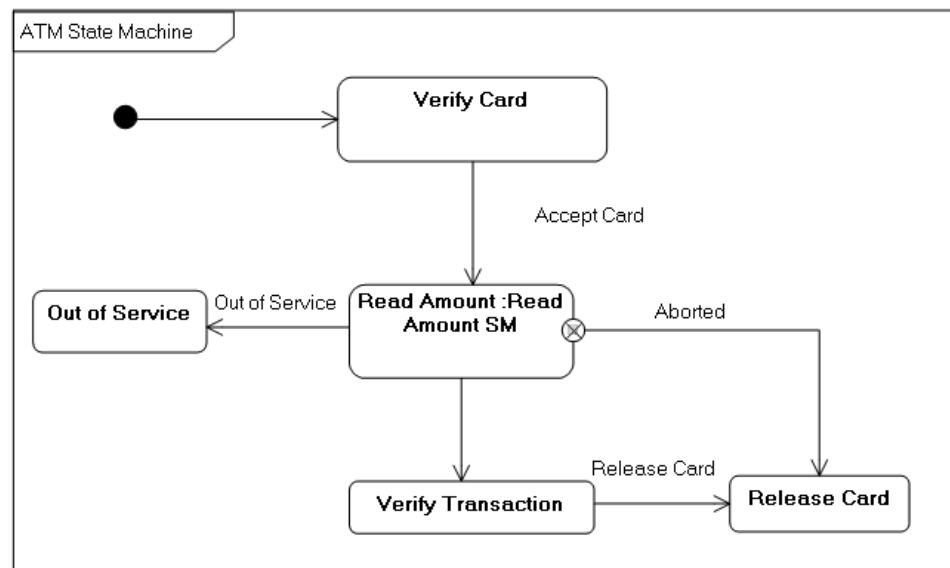
## PRESENTATION OF THE STATE MACHINE DIAGRAM

A state machine is the set of states and transitions between states that define the life cycle of an object that is variable over time.

The state machine diagram enables representation of the sequence of states that an object can take in response to interactions with the objects (internal or external to the studied system) in its environment.

### **Example of state machine diagram**

The diagram below describes possible behaviors of an automated teller machine:



## Creating a State Machine Diagram

A state machine diagram is created based on a state machine.

You can create a state machine using a package, class or component.

To create a state machine diagram in **HOPEX Application Design** :

1. Click the navigation menu, then **Application Design Resource**.
2. In the navigation pane, select **OO Implementation (UML)**.
3. In the edit window, click the **State Machine** tile.
4. Right-click the name of the state machine concerned and select **New > State Machine Diagram..**  
The diagram opens in the edit window.

The diagram is initialized by creation of a region. A region is part of a composite state or state machine which contains states and transitions and of which execution is autonomous.

## STATES

A state is a condition or situation in the life of an object, during which it satisfies some condition, performs some activity, or waits for some event. A state represents an interval of time delimited by two events. It is a phase an object passes through during its life cycle.


### *Examples of object states*

- A person can be:
  - Unmarried
  - Married
  - Divorced
- An item can be:
  - Available
  - In stock
  - At reorder level
  - Out of stock
  - etc.

---





## Creating a State

To create a state in a state machine diagram:

1. Click the arrow associated with the **State** button of the object insert toolbar 
  2. Select a state type.
  3. Click in the diagram work area.  
The **Add State** dialog box opens.
  4. Indicate the **Name** of the state and click **Create**.  
The state appears in the diagram.

### State types

It is necessary to specify the state type at the time of its creation. It can be:

-  A normal state: has no sub-structure.
-  A composite state: comprises several states, described in the diagram.
-  A sub-machine state: calls the descriptor of a state machine described elsewhere. See [Detailing Behavior of a State](#).
-  A final state

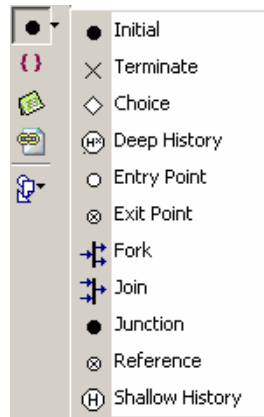
When you place a state in another state, it is automatically connected as a component of this state.

## Pseudo-states

Pseudo-states are used to specify complex paths by combining several transitions between states.

They can be of different types: initial, final, choice, deep history, shallow history, input, output, fork, join, junction or reference.

### ***Initial***



An initial pseudo-state has a single output transition to the initial state of the object at its creation.

### ***Deep history***

A deep history pseudo-state represents the last active configuration of a composite state containing it; that is the configuration that was active the last time the composite state was exited.

### ***Simple history***

A simple history pseudo-state represents the most recent active sub-state of a composite state (without the sub-states of this sub-state).

### ***Fork***

A fork separates a transition into several concurrent transitions.

### ***Join***

A join is the grouping of several transitions into a single transition.

### ***Choice***

Represents the choice of a transition between several possible transitions.

### ***Junction***

A junction is used to define paths of complex transitions between several states.

**Input**

Entry point of a state machine or of a composite state.

**Output**

Exit point of a state machine or of a composite state.

**Reference**

Reference to an input or output of a state machine or of a composite state.

**Final**

Input in this pseudo-state involves complete shutdown of the state machine.

**Deep history**

A **Deep History** state represents the last active configuration of a composite state; that is the configuration that was active the last time the composite state was exited.

A **Simple History** state represents the most recent active sub-state of the composite state.

Example:

Consider the "Married" state as the last active configuration. Sub-states of this state are "With children" and "Without children". In the case of a deep history, the "With children" and "Without children" sub-state is specified. In the case of a simple history, only the "Married" state is taken into account.

---

## Detailing Behavior of a State

A state can be made up of sub-states.

To describe composition of a state in a diagram:

1. Open the pop-up menu of a state and select **New > Detailing Behavior**.  
The state machine diagram creation window opens.
2. Click **New**.  
The diagram opens.

You can also define composition of a state by associating it with a new or existing state machine:

1. Open the properties dialog box of the described state.
2. Click the **Characteristics** tab.
3. In the **Detailing Behavior** box, create a state machine or query an existing state machine.

---

## State Properties

To access the state properties:

1. Right-click the state.
2. Select **Properties**.  
The properties dialog box of the state appears:

This can be used to:

- Modify the state **Name**.
- Indicate whether the sub-states are **Concurrent**, meaning they can be executed simultaneously.
- Indicate the **Detailing Behavior** (in the case of a complex state). See [Detailing Behavior of a State](#).
- Specify the **Activities** that can be performed at input, output or while the object is in this state.
  - *The contents of the properties dialog box of a state vary depending on state type.*

## STATE TRANSITIONS

Passage from one node to another is represented by a *transition*.

) A transition is passage of an object from one state to another. A transition is the response of an object to an event it receives. When an event occurs and certain conditions are satisfied, the object executes certain actions while still in the first state, before passing to the second state.

All authorized transitions must be defined. Those that are not defined are prohibited.

Examples of transitions:

For the marital status of a person, certain transitions are possible:

- It can change from the "unmarried" to the "married" state
- It can change from the "married" to the "divorced" state.


Other transitions are not possible:

- The state cannot change from "unmarried" to "divorced".

---

### Creating a Transition

To create an transition between two states:

1. In the state machine diagram, click **Transition (UML)**  in the insert toolbar.
2. Click the source state and drag the mouse to the target state.
3. Release the mouse button. The association is created.

---

### Transition Types

A transition can be external, internal or local.

You can specify the transition type in the transition properties dialog box, in the **Characteristics** tab.

#### External transition

An external transition is a transition that modifies the active state.

#### Internal transition

An internal transition enables an object to react to the arrival of an event that does not result in a state change but has an effect such as calling an operation or sending a message. For example, when pulling items from inventory, an item may not change state if the quantity remaining in the inventory is sufficient and does not fall below the reorder level or shortage level.

## Local transition

A local transition applies to sub-states of a composite state. It can cause a change of state only within the composite state.

---

## Transition Effects

Triggering of a transition can be accompanied by an effect. The effect can be represented by:

- An activity
- A collaboration
- An interaction
- A state machine

To define effect of a transition:

1. Open the properties dialog box of the transition.
2. Select the **Characteristics** tab.
3. Click the arrow in the **Effect (Behavior)** box and create or connect the object that defines the effect.

## Transition Effect Display

To modify how the transition effects are displayed.

1. In the state machine diagram, right-click the transition and select **Shapes and Details**.
2. Then select "Effect" in the tree that appears.

You can now specify whether to display all or part of the transition effects and their characteristics.

---

## Transition Triggering Event

In the properties dialog box of a transition in the **Event** tab, you can indicate the **Event Kind** that triggers a transition.

It can be:

- Any event
- Calling an operation
- Changing the object concerned by the transition
- Creating an object
- Destruction of an object
- Sending a signal
- Sending an operation
- Sending a signal from the object
- Receiving a signal
- Receiving an operation
- A *timer*

) *A timer is an event determined only by time elapsed. Example: Monday, at 4 pm, etc.*

Fields displayed under **Event Kind** vary according to the event kind selected.

You can select the object concerned by the effect.

In the case of an operation or signal, you can specify values of parameters sent.

# ACTIVITY DIAGRAM



The activity diagram is very similar to the state machine diagram. Unlike the state machine diagram which describes object behavior via state sequencing, the activity diagram describes element behavior in terms of actions.

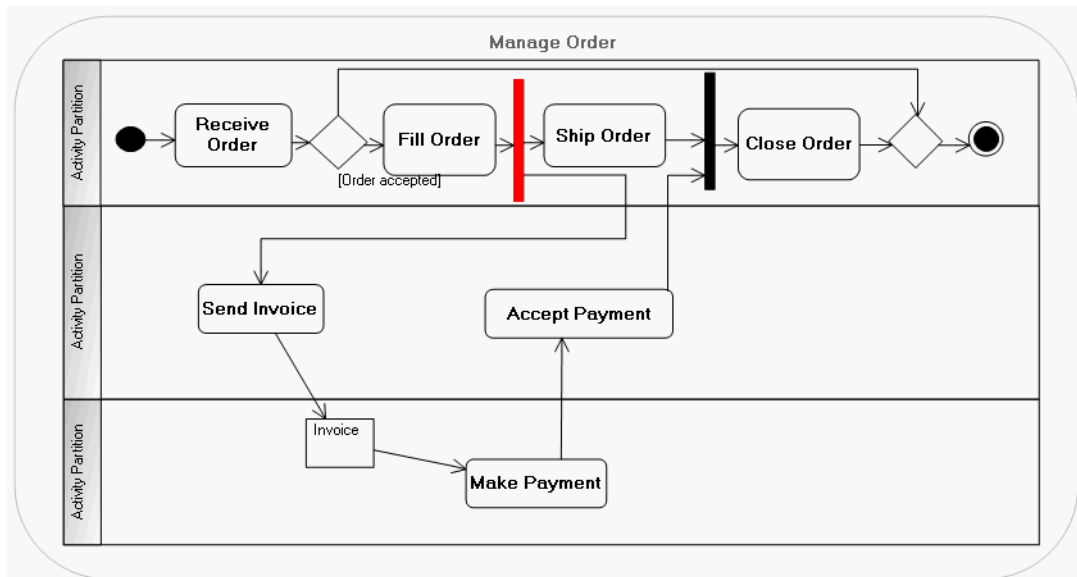
- 6 [Activity Diagram](#)
- 6 [Partitions](#)
- 6 [Nodes](#)
- 6 [Flows](#)

## ACTIVITY DIAGRAM

An activity diagram represents sequencing of steps describing behavior of a system element.

Steps are modeled by nodes - nodes of action, configuration or control - coordinated by data flows or control flows.

### *Example of an activity diagram*



## Creating an Activity Diagram

In **HOPEX Application Design**, an activity diagram is created based on a package or an activity.

You can create an activity for a package, component or class.

To create an activity diagram:

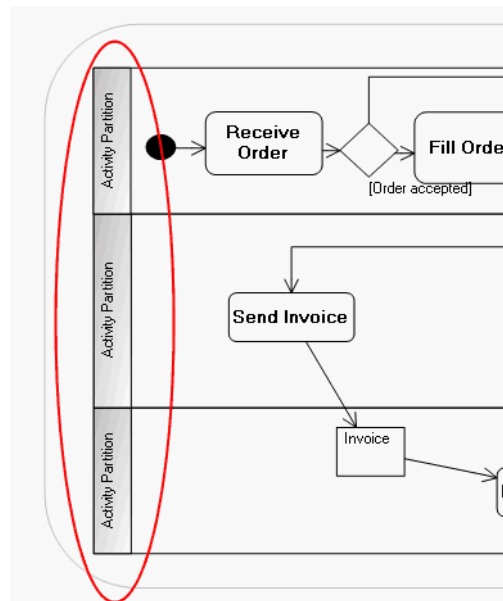
1. Right-click the package or the activity concerned.
2. In the pop-up menu that appears, click **New** > **Activity Diagram**.  
The new activity diagram opens.

# PARTITIONS

An activity diagram can be divided into partitions. Each partition contains nodes or actions as well as the flows between these elements.

You can use partitions to organize tasks or to specify the element responsible for implementation of multiple tasks.

For more details on swimlanes, see the **HOPEX Common Features** guide, "Handling Repository Objects" chapter, "Using Swimlanes" section.



---

## Creating a Partition

To create a partition in the activity diagram:

1. Click the **Partition** button  in the object insert toolbar.
2. Specify its name.
3. Click **Add**.

---

## Partition Properties

The **State** page presents the states contained in the partition.

The **Complements** page is used to specify the element represented by the partition. This is the element that implements the elements of the partition. It can be an actor, class or component.

# NODES

Nodes enable modeling of activity steps. There are different node types in **HOPEX**:

- Object nodes
- Parameter nodes
- Control nodes
- Object nodes: Input, Output and Exchange Pins

---

## Object nodes

Actions are the basic steps of behavior represented by the activity.

Coordination of actions is by control flows and data flows.

### Creating an Action

To create an action in an activity diagram:

1. In the diagram object insert toolbar, select the button corresponding to the action type then click the work plan.  
The dialog box for adding an action of the selected type opens.
  - *The insert toolbar offers three main types of actions.*
2. Specify its name and click **Add**.

### Modifying the Action Type

In the properties window of the action, in the **Characteristics** tab, you can modify the action type. It can be:

- Calling an operation of another object
- Creating an object
- Destruction of an object
- Local execution of an operation of the object
- Sending a signal from the object
- Terminating the object
- etc.

---

## Parameter nodes

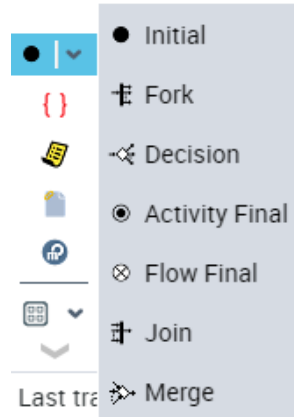
The parameter nodes of an activity describe the inputs and outputs of this activity.

They transmit parameters to the activity via flows which they send and receive.

## Control nodes

A control node coordinates the flows between nodes of an activity.

A control node can be of initial, final, decision, merge, fork or join type.



### Control node types

#### ***Initial***

An initial node indicates where the control flow starts when the activity is invoked. An activity can have several initial nodes.

#### ***Final***

When a token reaches a final node of an activity, all flows of the activity are stopped. Conversely, a final node of a flow destroys tokens that arrive, but has no effect on other tokens of the activity.

#### ***Decision***

A decision makes a choice of one flow from among several possible output flows. Output flows are selected according to their guard conditions.

#### ***Merge***

A merge fusion (merge) groups several alternative input flows into a single output flow. It is not used to synchronize concurrent flows, but to accept a single flow from among several.

#### ***Fork***

A fork separates a flow into several concurrent flows. Tokens arriving at a fork are duplicated through the output flows.

**Join**

A join synchronizes multiple flows. The flow is triggered when all input flows are available.

---

**Object nodes: Input, Output and Exchange Pins**

To specify input values of an action and return values, we use object nodes called input and output pins. The action can only start when a value is assigned to the input pin. Similarly, when the action is completed, a value must be assigned to the output pin.

**Input pin**

An input pin supports input values consumed by an action that it receives from other actions.

**Output pin**

An output pin supports output values produced by an action and supplies these values to other actions through flows.

**Exchange pin**

An exchange pin is used to represent data exchanged between two actions.

---

**Flows**

Passage from one node to another is represented by a flow.

**Control flow**

A control flow starts an action node when the previous node is completed. Objects and data cannot be transmitted by a control flow.

**Object flows**

An object flow enables transmission of data or objects from one node to another within an activity.



# INTERACTION DIAGRAMS



Interaction diagrams, that is the sequence diagram, communication diagram and interaction overview diagram, represent a series of interactions between objects, ordered in time. They show one or more possible illustrations of a system.

The following points are covered here: :

- 6 [Interactions](#)
- 6 [Sequence Diagram](#)
- 6 [Communication Diagram](#)
- 6 [Interaction Overview Diagram](#)

# INTERACTIONS

An interaction describes behavior of a system in a particular context by exchanges of messages between system elements.

While state machine diagrams or activity diagrams study individual behaviors, interaction diagrams concentrate on cooperation of a group of objects.

---

## Creating an Interaction

You can create an interaction from a package, class or component.

To create an interaction in **HOPEX Application Design** :

1. Click the navigation menu, then **Application Design Resource**.
2. In the navigation pane, select **OO Implementation (UML)**.
3. In the edit window, click the **Interactions** tile.
4. Click the **New** button.
5. Enter the name of the interaction and an owner if necessary.
6. Click **OK**.

---

## Creating an Interaction Diagram

The sequence diagram, communication diagram and interaction overview diagram are created using an interaction.

To create an interaction diagram:

1. Right-click on an interaction.
2. In the pop-up menu that appears, click **New** > **Interaction Diagram**.

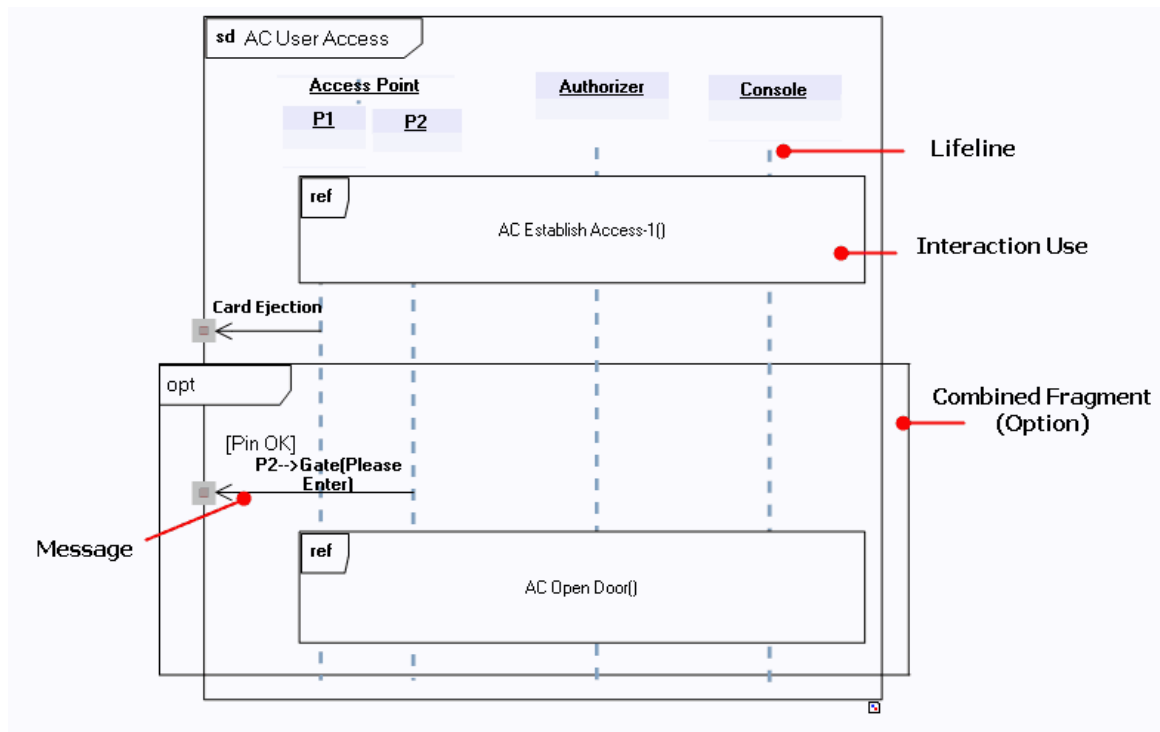
## SEQUENCE DIAGRAM

The sequence diagram highlights the chronology of messages exchanged between objects participating in an interaction. These objects are represented in the diagram by their lifelines.

### Example of a sequence diagram

The diagram below describes behavior of an automated teller machine:

- Two entry points (represented by lifelines) have a user access check. This check is described in an interaction.
- Depending on the result of the check, either access is refused and the user card is rejected, or door opening is actuated;
- An optional behavior (represented by a combined fragment) can influence door opening.



---

## Creating a Sequence Diagram

To create a sequence diagram in **HOPEX Application Design** :

1. Right-click on an interaction.
2. In the pop-up menu that appears, click **New** > **Interaction Diagram**.

See also [Creating an Interaction](#).

---

## Lifelines

A lifeline represents a participant in an interaction.


Lifelines are instances of different types (of classes, of actors, etc.).

In a sequence diagram, time is represented as passing from top to bottom along the lifelines of these objects. Message instances transit between these objects.

- *The instances represented in a sequence diagram can be instances of a class, actor, package, use case, component, or node, used to define the sequence diagrams at the desired level of detail.*

### Creating a lifeline

To create a lifeline/

1. Click the **Lifeline** button 
2. Click on the diagram.  
A dialog box opens.
3. Enter the name of the lifeline.
4. Click **Add**.  
The lifeline appears in the diagram.

### Lifeline properties

To access properties of a lifeline:

- > Select the instance and click **Properties** in the edit window if it is not activated.

You can select the **Type** of the object (Actor, Class, etc.), specify the **Class**, **Actor**, etc. of which it is an instance, and indicate its *Stereotype*.

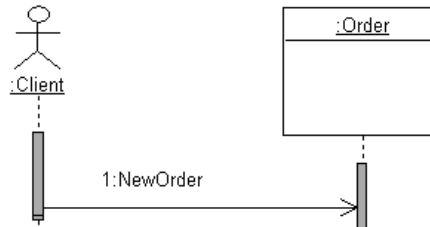
---

## Messages

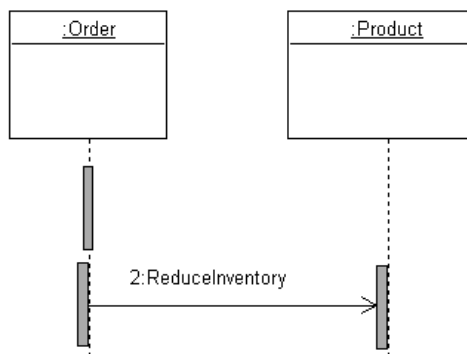
A message defines a particular communication between lifelines of an interaction. It specifies the sender and receiver via intermediate occurrence specifications, as well as the type of communication. This communication can be, for example, sending a signal, calling an operation or deleting an instance.

## Examples of exchanged messages

- 1) The message sent by the "Client" actor to the "Order" class carries the "New Order" signal.



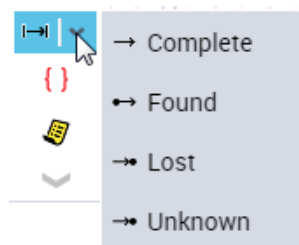
- 2) The message sent by the "Order" class to the "Product" class calls the "Reduce inventory" operation.



## Creating a message

To create a message in the sequence diagram:

1. Click the **Message** button in the insert toolbar, selecting the required message type.



2. Click on the dotted line under the first object, and hold down the mouse button while dragging the cursor to the dotted line under the second object.  
The message exchanged between the two objects is drawn.

## Message types

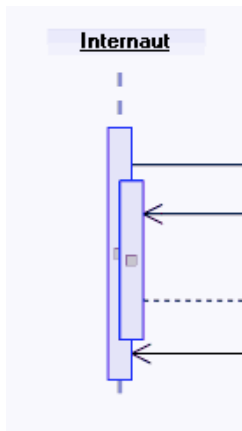
You can create four types of message:

- In a message type "Complete", the sender and receiver are both defined.
- In a message type "Lost", only the sender is known. Here we consider that the message never reaches its destination.
- In a message type "Found", only the receiver is known. This is the case when origin of the message is outside the description context.
- In a message type "Unknown", neither sender nor receiver are defined.

---


## Execution Specification

An execution specification represents an action or behavior unit that progresses from a start occurrence specification to an end occurrence specification.



## Creating an execution specification

To create an execution specification:

1. In the sequence diagram, click the **Execution Specification** button  in the object insert toolbar.
2. Position it on the lifeline concerned.  
The specification appears in the diagram.

---

## Occurrence specification

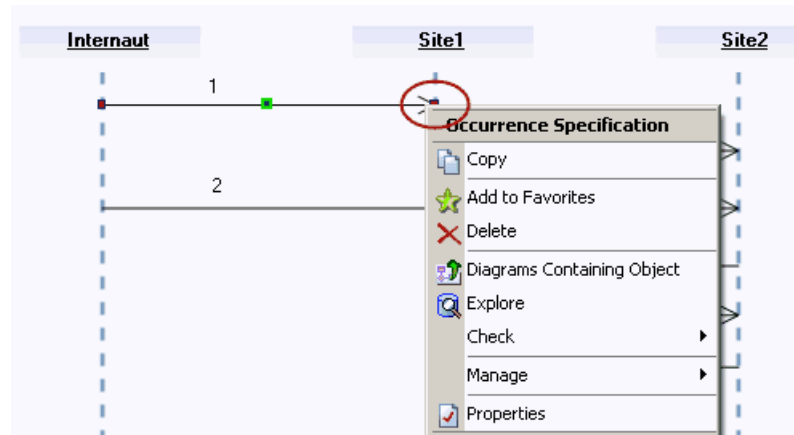
Creation of a message or an execution specification automatically creates occurrence specifications.

An occurrence specification is a syntax point at the extremity of a message or at the start or end of an execution specification.

Occurrence specifications are ordered along a lifeline.

These are basic semantic units of an interaction.

You can access the pop-up menu of an occurrence specification by right-clicking one of the extremities of a message.



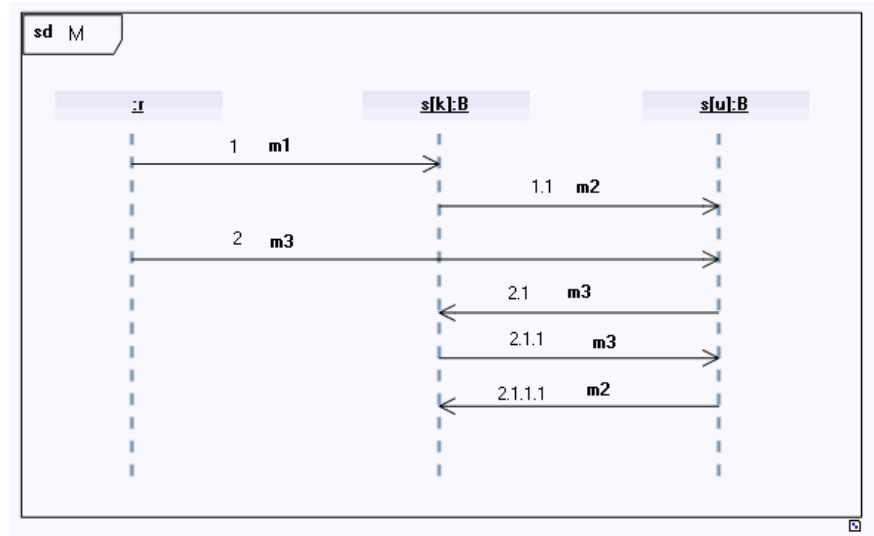
## Calculating sequence numbers

From positioning of occurrence specifications, a calculation tool enables ordering of messages and execution specifications.

To order messages circulating between lifelines of an interaction:

1. Open the pop-up menu of the described interaction.
2. Select **Calculate Sequence Numbers**.  
The tool automatically applies numbers to messages.

## Example



You can manually modify the sequence number of a message in the message properties dialog box:

- > Select the **Characteristics** tab and change the value in the **Sequence Expression**.

When you restart calculation of sequence numbers, this updates sequencing according to the modifications made.

---

## Combined Fragment

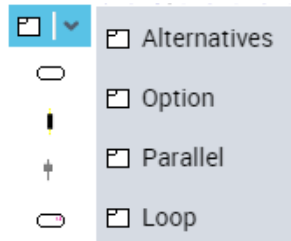
A combined fragment enables concise description of several execution sequences.

A combined fragment is defined by an interaction operator and the corresponding interaction operands.

## Creating a combined fragment

To create a combined fragment:

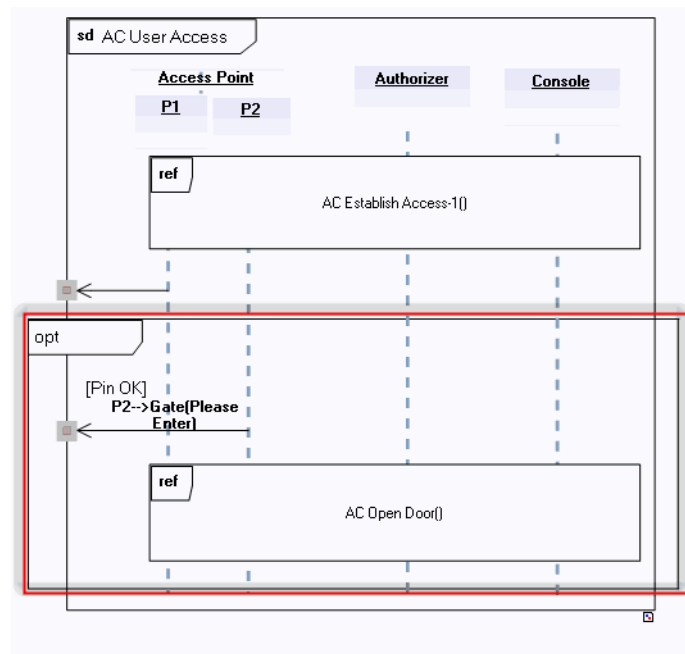
1. In the sequence diagram insert toolbar, click the **Combined Fragment** button.  
You can associate different types of interaction operator to a combined fragment. The arrow at the right of the button offers shortcuts to four of these. See [Interaction operator type](#).



2. Click on the diagram.  
The combined fragment creation dialog box appears.
3. Specify its **Name** and the **Interaction Operator Type** if not already indicated.
4. Click **Finish**.

A combined fragment is represented by a rectangle with the interaction operator type displayed at the top left-hand corner.

In the example below, a combined fragment of option type translates a behavior that could disturb normal operation (door opening).



## Interaction operator type

The interaction operator type conditions meaning of the combined fragment. There are various operator types: seq, alt, opt, break, par, strict, loop, region, neg, assert, ignore and consider.

### Alternatives

Alt expresses the possibility of choosing between different possible behaviors by evaluating guard conditions associated with each of the operands. Only one of these operands can be executed.

The Else operand is selected when none of the other conditions is satisfied.

### Option

Opt represents a choice between the unique operand proposed, or none.

### Break

Break represents a stop scenario that is executed instead of the rest of the containing interaction fragment.

**Parallel**

Par means that the different operands can be executed in parallel. Occurrence specifications of different interaction operands can be sequenced in various ways as long as the order imposed by each operand is maintained.

**Weak Sequencing**

Seq designates weak sequencing between behaviors of operands defined by three properties:

- Order of occurrence specifications within each of the operands is maintained in the result.
- Occurrence specifications of different lifelines from different operands can appear in any order.
- Occurrence specifications of the same lifeline from different operands are ordered so that the occurrence specification of the first operand appears before that of the second.

**Strict Sequencing**

Strict defines strict sequencing of operand behaviors.

**Negative**

Neg represents an invalid operand.

**Critical Area**

Critical represents an area that must be processed atomically, meaning that occurrence specifications cannot be sequenced with those of this critical area.

**Ignore/Consider**

Ignore and consider require that a list of relevant messages be specified.

Ignore indicates that the types of certain messages are ignored in the combined fragment.

Consider indicates that certain messages will be considered in the combined fragment. This is equivalent to defining all other messages as 'ignored'.

**Assertion**

Assert represents a sequence that is the only one valid for a given message.

Therefore any sequence defined by an interaction fragment that starts with messages leading to the sequence defined by the Assert block and continuing with an exchange of messages that do not respect the Assert block must be defined as invalid.

Assertions are frequently used in combination with Ignore and Consider types.

## Loop

Loop indicates that the interaction operand will be repeated a certain number of times. It is possible to specify minimum and maximum number of loops, as well as an expression of loop continuation.

## Interaction operands

An interaction operand is contained in a combined fragment, and represents an operand of the expression given by the containing combined fragment. It can be conditioned by an interaction constraint, which acts as guard condition.

### Creating an Interaction Operand

To create an interaction operand:

1. Right-click the combined fragment which contains the interaction operand.
2. Select **New > Interaction Operand**.
3. Name the operand and click **OK**.

### Creating an Interaction Constraint

To create the interaction constraint that will condition the operand:

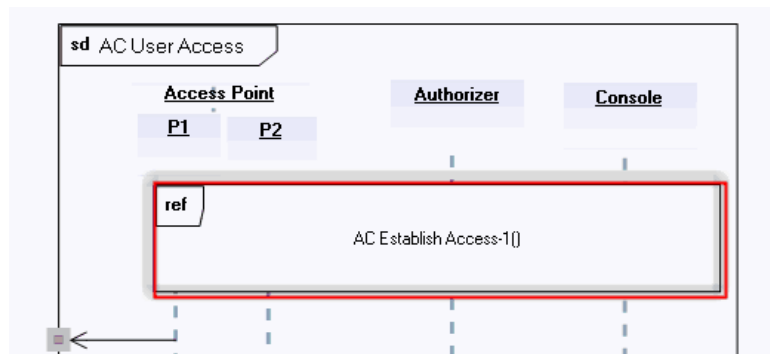
1. Open the properties dialog box of the interaction operand.
2. Click the **Characteristics** tab.
3. In the **Condition** frame, click **New**.
4. The condition is represented by a constraint. Define the constraint and click **OK**.

---


## Interaction Use

An interaction use refers to an interaction. It is a means of copying content of the interaction referenced at the interaction occurrence location.

### Example



To create an interaction use:

1. Click the **Interaction Use** button 
2. Click on the diagram.
3. In the dialog box that appears, specify the name and the interaction called.
4. Click **Finish**.

You can specify arguments of an interaction use. An argument is a specific value corresponding to a parameter of the interaction called. In addition, when the argument has been created on the interaction use, you must align it with the interaction parameter called.

To create an argument:

1. Open the properties dialog box of the interaction use.
2. Click the **Characteristics** tab.
3. In the **Arguments** frame, click the **New** button.  
A value specification is created.  
You can rename it and specify its characteristics by opening its properties dialog box.

To align the argument with the interaction parameter called:

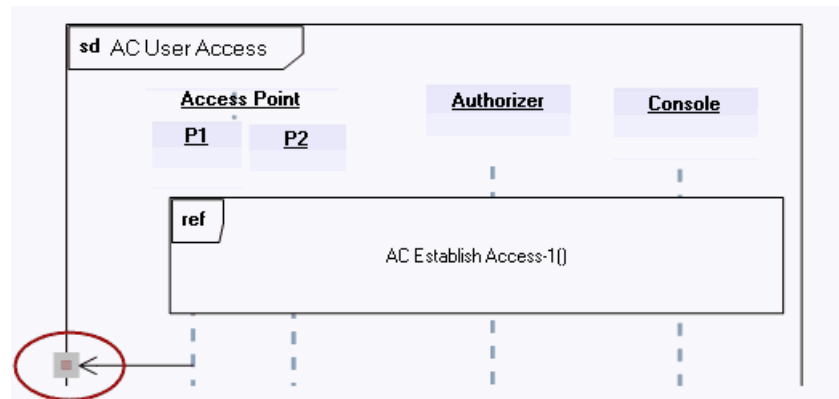
1. In its properties dialog box of the interaction use, select the **Characteristics** tab.
2. Click the arrow at the right of the **Interaction called** box and select **Modify**.  
A dialog box displays characteristics of the interaction called.
3. For each parameter, click in the value column and select the corresponding value specification.

---


## Gate

A gate is a connection point between a message external to an interaction fragment and a message belonging to this interaction fragment.

### Example



To create a gate in the sequence diagram:

1. Click the **Gate** button  in the object insert toolbar.
2. Click on the frame outlining the interaction at the point you wish to position the gate.  
The gate then appears in the diagram.

---

## Continuation

A continuation is a syntax means for defining the continuation of sequences of different branches of an Alternatives combined fragment. Continuations are similar to labels representing intermediate points in a control flow.

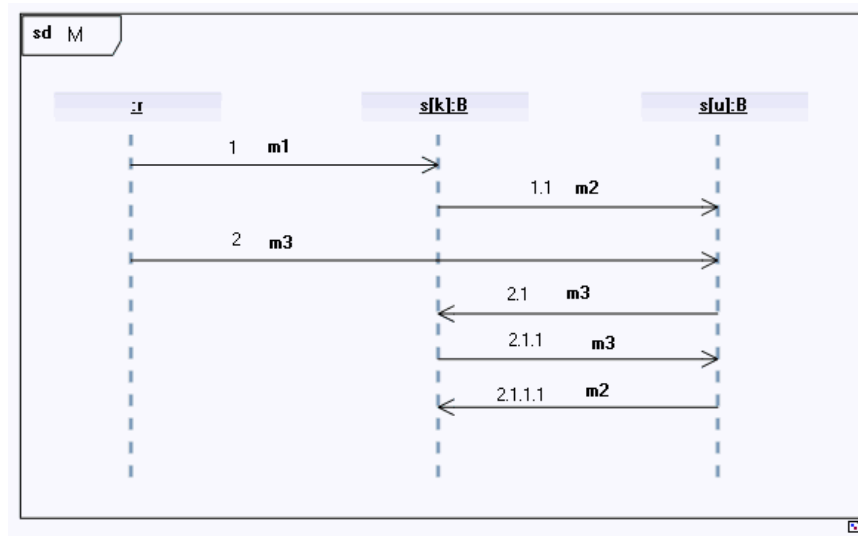
## COMMUNICATION DIAGRAM

The communication diagram is a simplified representation of the sequence diagram, concentrating on message exchanges between objects within an interaction.

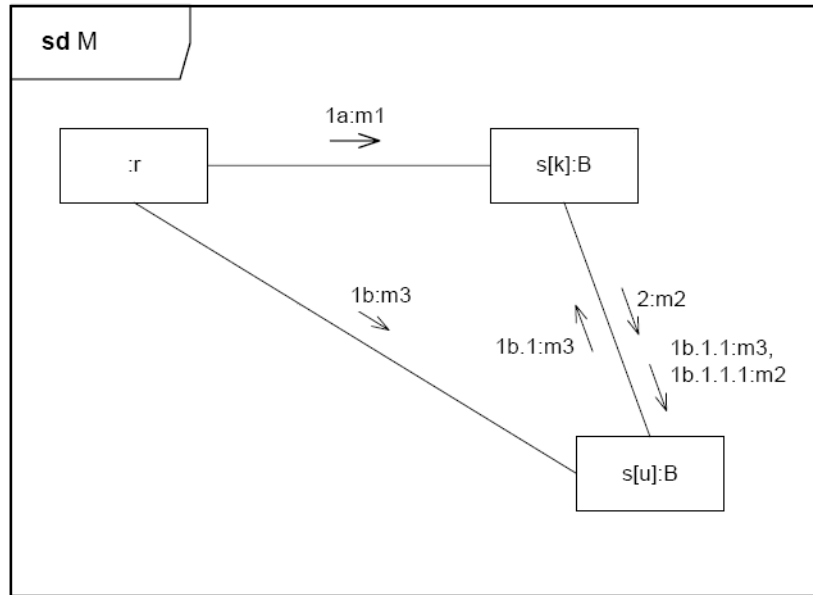
The sequence and communication diagrams are isomorphic. When a communication diagram relates to an interaction already described in a sequence diagram, it is automatically initialized from the information contained in the sequence diagram.

### Example

#### *Sequence Diagram*




## Communication Diagram



### Diagram objects

Communication diagram objects are lifelines and messages transmitted by connectors.

When you connect two lifelines with a connector , the connector creation dialog box proposes messages that may be transmitted.

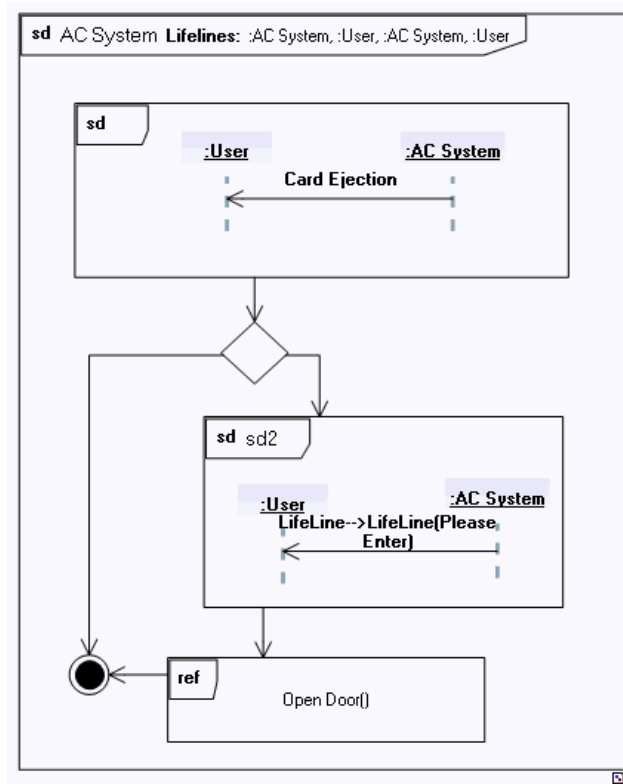
When the connector has been created, you can associate new messages in its properties dialog box, in the **Message** tab.

The sequence of messages is given by a sequence number associated with each message. See [Calculating sequence numbers](#).

For more details on connectors, see "[Connectors](#)", page 201.

## INTERACTION OVERVIEW DIAGRAM

The interaction overview diagram describes sequences possible between scenarios previously identified in the form of sequence diagrams. It gives an overview of control flows.



Objects represented in the interaction overview diagram are interactions and interaction uses, lifelines, messages, control nodes and control flows.



# THE DEPLOYMENT DIAGRAM



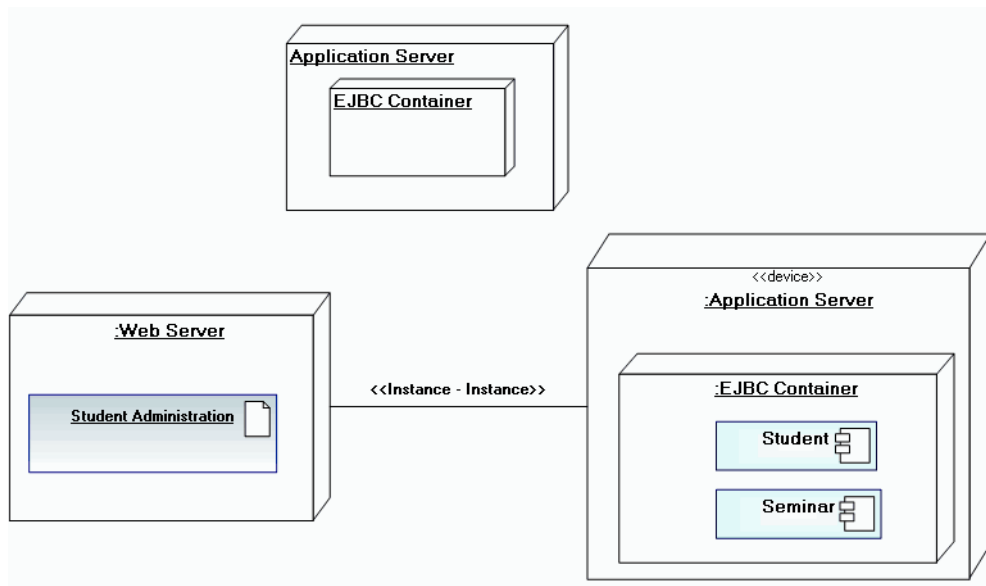
The deployment diagram complements the component diagram. It describes hardware resources (computer, router, etc.) in the system, and indicates distribution of components on these hardware resources.

It also describes connections between components or nodes.

This diagram also allows specification of interfaces required and implemented for sequencing of components.

It can be illustrated and supplemented by the addition of node, component or class instances.

## Example of a deployment diagram



## Creating a Deployment Diagram

In **HOPEX Application Design**, a deployment diagram is created from a package.

To create a deployment diagram:

1. Right-click the package in question.
2. In the pop-up menu that appears, select **New** > **Deployment Diagram**.

The new deployment diagram opens in the Edit window.

---

## Deployment Diagram Objects

### Node

A node is a physical object representing an IT resource, generally with a memory and often with calculation capabilities, on which components can be deployed.

Nodes can comprise other nodes or artifacts. To indicate that a component is assigned to a node, either place the component in the node, or connect the component to the node by a dependency link.

See ["Dependency links", page 205](#).

You can create a node in the deployment diagram using the **Node (UML)** button



in the insert toolbar.

### Communication path

Connections between nodes are represented by communication paths via which signals and messages are exchanged.


### Component

A component represents a modular part of a system that encapsulates its content, and which can be replaced in its environment. A component defines its behavior by means of interfaces that it provides and requires.


One component can be replaced by another if their interfaces conform.

A component can be a software package, program, code unit, etc.

### Artifact

An artifact  represents a physical information element used or produced by the software development process, or by the deployment or implementation of a system. Example: source files, scripts, executable binary files, development deliverables, word processing documents, electronic messages, etc.

### Manifestation


A manifestation  is the real physical restoration in an artifact of one or several modeling elements such as components or classes.

The source of a manifestation dependency is an artifact, the target a component or class.

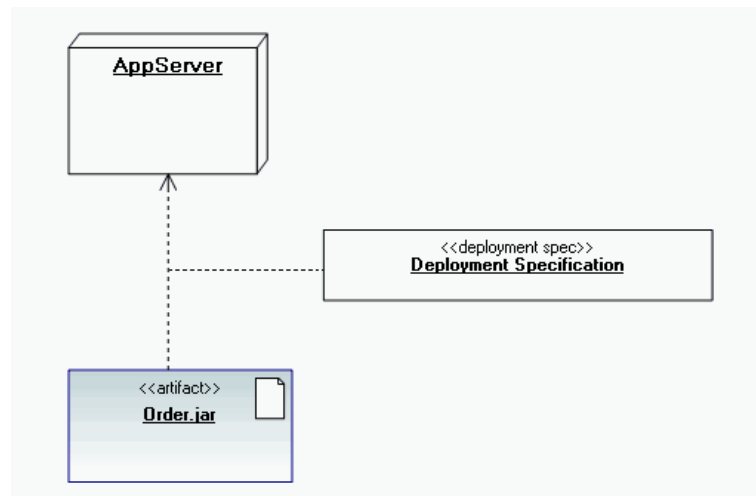
## Deployment specification

Deployment specification enables indication of the characteristics that determine execution parameters of an artifact or component deployed on a node.

## Configuration

The configuration button  enables creation of the link between a deployment specification and a deployment.

### Example





# APPENDIX: ATTRIBUTE TYPE



The following points are covered here:

- 6 ["Primitive Types", page 246](#)
- 6 ["Packages and Primitive Types", page 248](#)
- 6 ["Defining New Primitive Types", page 251](#)

## PRIMITIVE TYPES

A primitive type is used to group characteristics shared by several attributes. Primitive types are implemented as classes.

To access primitive types in **HOPEX Application Design**, the administrator must import the "Information Architecture" solution pack. See ["Prerequisite: Importing the Primitive Types", page 20](#).

---

### Defining a Primitive Type

Primitive types are defined in a class diagram.

These are classes for which the following is specified:

- They are of the "Primitive Type" stereotype.
- They are "Abstract" classes because they will not be instantiated.
- They are "Non-persistent" classes. They should not have a corresponding table in the database.

To specify types of class attributes:

1. Open the properties of the class and select the **Internal Characteristics** page.
2. Expand the **Attributes** section.
3. Click the **Type expression** field and select the attribute type using the arrow.

The following classes are in the standard list:

| <b>Alphanumeric types</b> |  | <b>Other Information</b> |
|---------------------------|--|--------------------------|
| M-Char                    | Alphanumeric string of fixed length    | Length                   |
| M-Varchar                 | Alphanumeric string of variable length |                          |
| <b>Numeric types</b>      |  |                          |
| M-Numeric                 | Number                                 | Length, decimal places   |
| M-Amount                  | Amount expressed as currency           | Length, decimal places   |
| <b>Date types</b>         |  |                          |
| M-Date                    | Date                                   |                          |

|                     |  |  |
|---------------------|--|--|
| M-Time              | Time   |  |
| M-Datetime          | Date and time  |  |
| <b>Binary types</b> |  |  |
| M-Timestamp         | Identification automatically generated from the date and time, expressed in thousandths of seconds since January 1, 1970 |  |
| M-Bool              | Boolean, equals 0 or 1   |  |
| M-Multimedia        | Binary string  |  |

## PACKAGES AND PRIMITIVE TYPES

---

### Packages

) *A package partitions the domain studied and the associated work. It enables grouping of various elements, in particular use cases and classes. A package can also contain other packages. Packages are interconnected through contractual reports defining their interface.*

The assignment of classes to packages imposes a rigid structure. As a class can belong to only one package, it is necessary to define client/supplier relationships so packages can use classes they do not own when they need to.

This is especially important for primitive type classes, because they will be used to define the attributes of other classes.

- *Rule: a class can belong to only one package.*

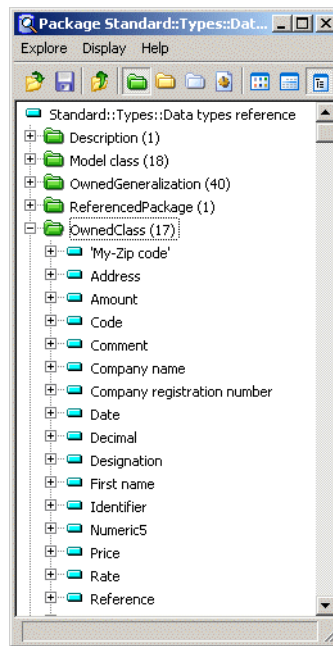
What primitive types are available for typing the class attributes depends on which package the class is in.

The type you can give to class attributes can only be primitive types defined for the package containing the class.

The accessible primitive types are public classes with the "Primitive Type" stereotype, that are contained in or are used by the package or the packages of which it is the client.

You can define a reference package (or several reference packages) containing the primitive types used by the enterprise. All the other packages are declared as clients of the reference package of primitive types.

In the example below, the "Data types reference" package contains the classes "Address", "Code", "Date", etc.



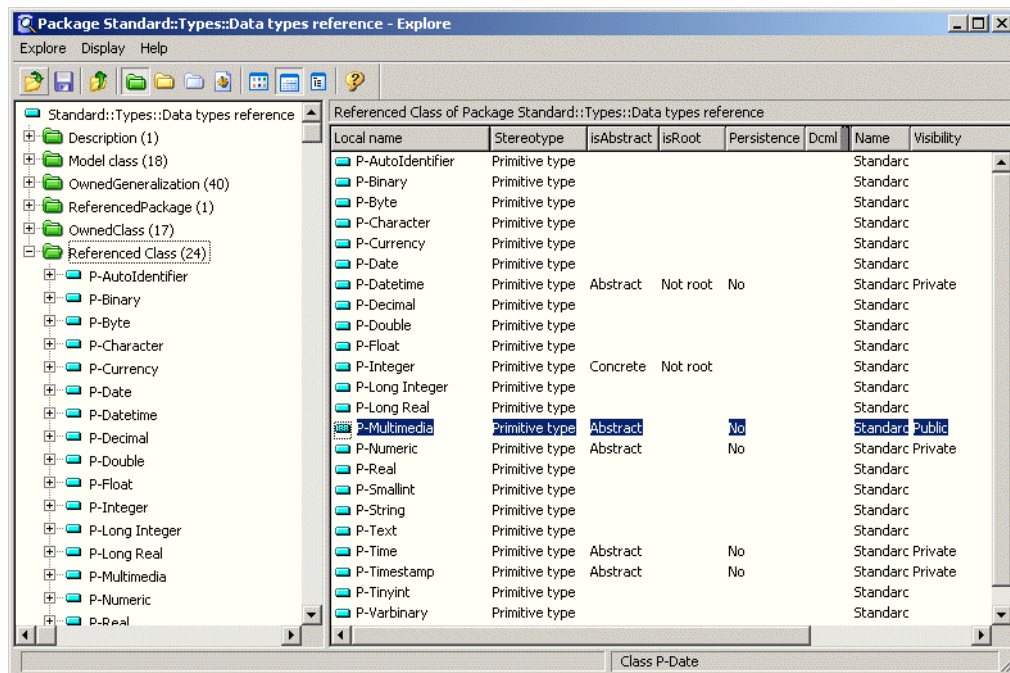
It is referenced by the packages "Library", "Order management", etc.

The class attributes for these packages can be typed using the types "Address", "Code", "Date", etc.

It is also possible to specify directly that a package uses a class contained in another package.

In the example below, the classes "P-Datetime", "P-Multimedia", "P-Numeric", etc. are used by the "Data Type Reference" package without being owned by that package.

Of these classes, only "M-Multimedia" is exported by the package for public use.



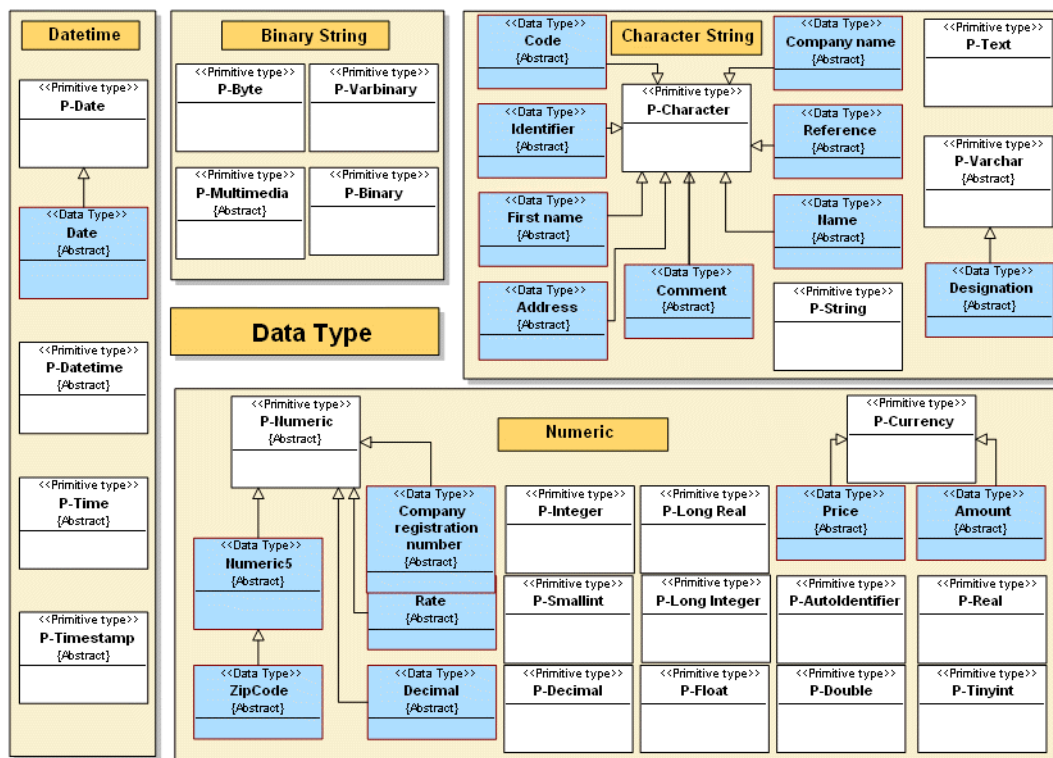
## DEFINING NEW PRIMITIVE TYPES

New primitive types can be defined using a class diagram.

Depending on whether classes have been organized into packages, the class diagram can describe:

- A reference database.
- The package of reference types.

You can define your own primitive types by declaring them as subclasses of the standard primitive types, as shown in the example below:

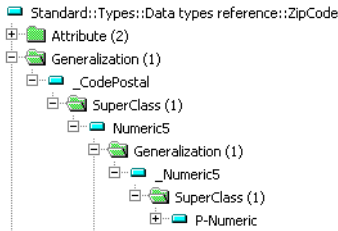


The primitive types defined as subclasses will automatically inherit the characteristics of their superclass. In particular, the datatype conversion rule for the superclass is applied to the subclass.

It is possible to specify a length and a number of decimal places for the subclass. These will be taken into account when generating the data types if they were not already defined for the superclass.

Inheritance can occur at several levels.

In the following example, the primitive type “ZipCode” is a specialization of the “Numeric5” type of length 5, which is itself a specialization of the standard type “P-Numeric”.

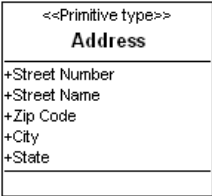


If the new primitive type is not defined directly or indirectly as a subclass of a standard primitive type, the conversion table that maps primitive types to column data types must be updated.

- A connection can also be directly defined between a type and the corresponding SQL datatype generated for each target DBMS without using the inheritance mechanism (see "Correspondences between pivots and datatypes" in the **HOPEX Database Builder** guide).

## Compound Primitive Type

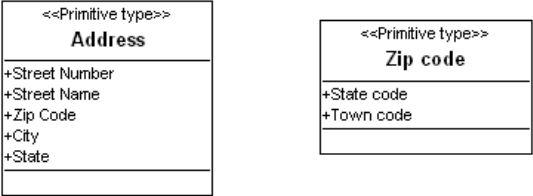
You can define a compound primitive type by assigning to it a list of attributes.



Here the Address type is composed of number, street, zip code, city, and country. The derivation of the Address attribute will produce these five columns.

It is possible to have several levels of compound types by assigning a compound type to an attribute of a compound type.

For example, the zip code can be broken down into the five main digits and the four-digit extension:



HOPEX XMI 2.1 Import for UML2

## XMI IMPORT OVERVIEW

---

The XML Metadata Interchange XMI is an OMG standard for exchanging UML Models between different UML products such as modeling tools and UML Design.

The XMI Import project aims at importing the content of .xmi and .uml files into HOPEX so that users can reproduce diagrams from other platforms. Only the data (the objects) are imported, not the drawings.

### Prerequisites

The XMI Import feature supports UML versions from 2.3 to 2.5. A file with a version lower than 2.3 or higher than 2.5 can be imported, without guarantee of full success.

However, any URI not corresponding to one of the ones provided by the OMG (see links down below) does not allow the import.

<http://schema.omg.org/spec/XMI/index.htm>

<http://schema.omg.org/spec/UML/index.htm>

### Scope of XMI Import

The purpose of the XMI Import tool is to import XMI data into HOPEX repository. The objects imported are those belonging to the Class Diagram, Use Case Diagram, Component Diagram, Composite Structure Diagram, Activity Diagram, Communication Diagram, Sequence Diagram, State Machine Diagram, Interaction Overview Diagram, Object Diagram, Deployment Diagram, etc.

Only objects are imported, not the drawings.

The exact list of the supported objects and their properties is detailed below.

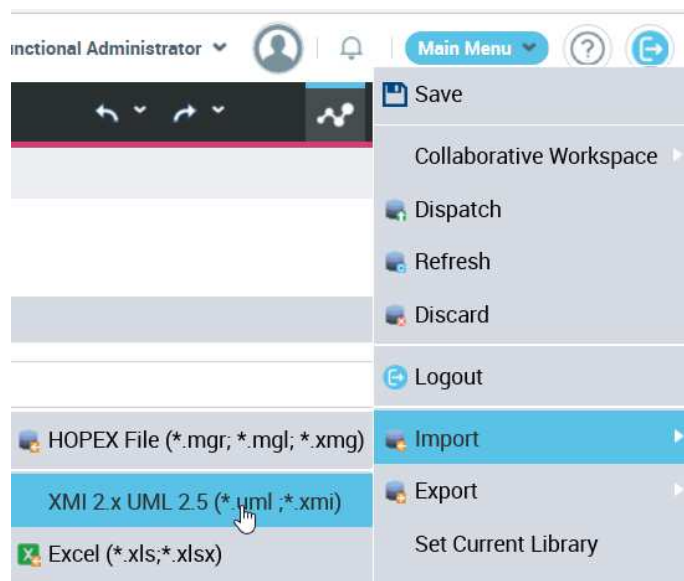
## IMPORTING XMI AND UML FILES

---

Depending on the source tool, the XMI import tool can import .xmi or .uml files.

To import a file:

1. In HOPEX, select **Main Menu > Import > XMI 2.x UML 2.5 (\*.uml; \*.xmi)**.



The import wizard appears.

2. In **File Location**, select the file to be imported.
3. Select the library in which you want to import the data (optional).
4. Click **Next**.

The wizard shows the import process progression.

Then it shows the report of imported data.

## HOPEX/XMI OBJECT MAPPING

---

The following paragraph indicates what kinds of objects are imported by UML2 diagram types. Only objects belonging to the selected package are imported.

All objects that do not belong to a package are attached to a package called “Default Package”.

A package “UML Primitive Types” is imported by default if not already.

### **Class Diagram**

Classes, attributes, associations, association ends, generalizations, generalization sets, operations, parameters, data types, primitive types, interfaces, enumerations, etc.

### **Use Case Diagram**

Use cases, actors, packages, constraints, extension points (text), participations, extensions (link), inclusions (link), generalization, dependencies, etc.

### **State Machine Diagram**

State machines, regions, states, pseudo states, transitions, constraints, etc.

### **Protocol State Machine Diagram**

Protocol state machines, regions, states, pseudo states, transitions, constraints, etc.

### **Activity UML Diagram**

Actions, control nodes, Input Pins, Output Pins, Exchange Pins, central buffer nodes, data store nodes, activity partitions, control flows, object flows, exception handlers, activities uml, activity parameter nodes, structured activity nodes, expansion regions, expansion nodes, interruptible activity regions, etc.

### **Component Diagram**

Components, ports, packages, interfaces, required interfaces, provided interfaces, classes, Connectors, realized elements, etc.

### **Composite Structure Diagram**

Collaborations UML, collaboration uses, parts, dependencies, connectors, interfaces, classes, provided interfaces, required interfaces, etc.

## Sequence Diagram

Life lines, combined fragments, interaction uses, gates, states invariant, UML messages, constraints, etc.

---

Messages include those exchanged directly between lifelines as well as messages exchanged through execution specification.

---

## Communication Diagram

Life lines, connectors, UML messages, etc.

## Deployment Diagram

Packages, components, artifacts UML, nodes UML, devices, execution environments, interfaces, deployment specifications, deployments, manifestations, deployment configurations, component instances, device instances, node instances, execution environment instances, communication paths, etc.

🔗\* Only objects owned by the selected package or its sub-packages are exported.

Objects that are linked to objects contained in the selected export package but owned by another package are also exported in order to ensure links. However, they will be owned by the exported package.

The following table indicates concepts managed by the export tool:

## Class Diagram

| MEGA Concepts | MetaAttribute     | MetaAssociation(End)                  |
|---------------|-------------------|---------------------------------------|
| Class         | Name              | Class Target Dependency               |
|               | xmi_id            | Realization Class                     |
|               | Visibility        | Nested Class                          |
|               | Comment           | Association                           |
|               | Abstract          | Connector                             |
|               | IsLeaf            | Association Class                     |
|               | IsActive          | Attribute                             |
|               | Client Dependency | Operation (UML)                       |
|               |                   | Generalization                        |
|               |                   | Required Interface                    |
|               |                   | Provided Interface                    |
|               |                   | Constraint                            |
|               |                   | Port                                  |
|               |                   | AssociationEnd                        |
|               |                   | Owned Part                            |
|               |                   | Behavior: State Machine               |
|               |                   | Behavior: Activity Uml                |
|               |                   | Behavior: Interaction Uml             |
|               |                   | Behavior: Collaboration Uml           |
|               |                   | Protocol: ProtocolStateMachineDiagram |
|               |                   | Source Dependency                     |
| Data Type     | Name              | Class Target Dependency               |

| MEGA Concepts | MetaAttribute  | MetaAssociation(End)  |
|---------------|--|---|
|               | xmi_id<br>Visibility<br>Abstract<br>IsLeaf<br>IsActive<br>Client Dependency        | Nested Class  |
| Interface     | xmi_id<br>Name<br>Visibility<br>Comment<br>Abstract<br>IsLeaf<br>Client Dependency | Class Target Dependency<br>Nested Class<br>Association<br>Attribute<br>Operation (UML)<br>Generalization<br>RequiredInterface<br>SpecificationInterface |
| Enumeration   | _Hexaidabs<br>Name<br>Visibility<br>Comment<br>Client Dependency                   | Class Target Dependency<br>Attribute<br>Operation (UML)<br>Literal Value<br>RequiredInterface<br>Specification Interface                                |
| LiteralValue  | _Hexaidabs<br>Name   | Value Slot  |

| MEGA Concepts     | MetaAttribute  | MetaAssociation(End)  |
|-------------------|--|---|
| Expression        | xmi_id<br>Name<br>Visibility<br>Comment<br>Client Dependency                   | Class Target Dependency<br>Specification Interface                        |
| Primitive Type    | Name<br>xmi_id<br>Visibility<br>Abstract<br>IsLeaf<br>IsActive                 | Class Target Dependency<br>Nested Class                                   |
| Association       | xmi_id<br>Name<br>Visibility<br>Comment<br>IsAssociationDerived<br>IsNavigable | Connection<br>Dependency (Target Association)<br>Class via AssociationEnd |
| Association End   | xmi_id<br>Name<br>Aggregation:<br>Composite/Shared                             | Association<br><br>Dependency   |
| Association Class | xmi_id   | Class Target Dependency   |

| MEGA Concepts   | MetaAttribute   | MetaAssociation(End)   |
|-----------------|---|--|
|                 | Name<br>Visibility<br>Comment<br>IsLeaf<br>Abstract<br>IsActive<br>IsAssociationDerived<br>IsNavigable  | Nested Class<br>AssociationEnd<br>Association<br>Association Class<br>Class via AssociationEnd<br>Attribute<br>Operation (UML) |
| Attribute       | xmi_id<br>Name<br>Visibility<br>Comment<br>IsLeaf<br>IsOrdred<br>Uniqueness<br>ReadOnly<br>IsDerived<br>InitialValue<br>Multiplicity: UpperValue,<br>LowerValue | Dependency (Target Attribute)<br>AttributType  |
| Operation (UML) | xmi_id<br>Name<br>Visibility<br>Comment   | Precondition<br>Postcondition<br>Parameter<br>ReturnType   |

| MEGA Concepts     | MetaAttribute | MetaAssociation(End)      |
|-------------------|---------------|---------------------------|
| Dependency        | Abstract      | Target Dependency         |
|                   | IsQuery       |                           |
| Dependency        | _Hexaidabs    | Class Source              |
|                   | Name          | Class Target              |
|                   | Visibility    | Stereotype                |
|                   | Comment       |                           |
| Generalization    | xmi_id        | Super Class               |
|                   | Name          | UML constraint            |
|                   | Comment       |                           |
| GeneralizationSet | xmi_id        | Generalization            |
|                   | Name          |                           |
|                   | Comment       |                           |
|                   | IsComplete    |                           |
|                   | IsDisjoint    |                           |
| Constraint        | xmi_id        | ConstrainedClass          |
|                   | Name          | ConstrainedGeneralization |
|                   | Comment       | ConstrainedElement        |
|                   | MaxInt        | Actor (UML)               |
|                   | MinInt        | Package                   |
|                   | Specification | UseCase                   |
|                   |               | UseCaseParticipation      |

| MEGA Concepts  | MetaAttribute                 | MetaAssociation(End) |
|----------------|-------------------------------|----------------------|
| Parameter      | _Hexaidabs<br>Name<br>Comment | Parameter Type       |
| Behavior (UML) | _Hexaidabs<br>Name            |                      |

## Use Case Diagram

| MEGA Concepts | MetaAttribut                            | MetaAssociation(End)   |
|---------------|---|--|
| UseCase       | xmi_id<br>Name<br>Visibility<br>Comment | UsesUseCase<br>OwnedExtension<br>ExtensionPoint<br>Behavior: State Machine<br>Behavior: Protocol State Machine<br>Behavior: Interaction UML<br>Behavior: Activity UML<br>Behavior: Collaboration UML<br>Constraint<br>Generalization |
| Actor (UML)   | xmi_id<br>Name<br>Visibility<br>Comment | Participation<br>Constraint<br>Generalization  |
| Participation | xmi_id                                  | UseCase  |

|           |              |                    |
|-----------|--------------|--------------------|
|           | Name         | Actor (UML)        |
|           | Comment      | Constraint         |
| Extension | Multiplicity |                    |
|           | xmi_id       | Extended Use Case  |
|           | Name         | Extension Location |
|           | Comment      |                    |

## Composite Structure and Communication Diagram

| MEGA Concepts     | MetaAttribut      | MetaAssociation(End)                                   |
|-------------------|-------------------|--|
| Collaboration uml | xmi_id            | CollaborationRole                                      |
|                   | Name              | OwnedConnector   |
|                   | Comment           | OwnedCollaborationUse                                  |
|                   | IsAbstract        |  |
|                   | IsLeaf            |  |
| Collaboration use | xmi_id            | Type   |
|                   | Name              |  |
|                   | Comment           |  |
| Part              | xmi_id            | ConnectorEnd (of the LifeLine who represents the part) |
|                   | Name              | Dependency   |
|                   | Visibility        |  |
|                   | Client Dependency |  |
|                   | IsLeaf            |  |
|                   | IsUnique          |  |

|              |  |                   |
|--------------|--|-------------------|
|              | IsOrdered<br>Multiplicity<br>Aggregation:<br>Composite/Shared<br>Comment |                   |
| Connector    | xmi_id<br>Name<br>Connector Kind<br>IsLeaf                               | OwnedConnectorEnd |
| ConnectorEnd | xmi_id<br>Name<br>Multiplicity   | Connector         |

## State Machine

| MEGA Concepts | MetaAttribut                           | MetaAssociation(End)               |
|---------------|--|------------------------------------|
| State Machine | xmi_id<br>Name<br>Comment<br>Reentrant | DetailedState<br>Region            |
| Region        | xmi_id<br>Name<br>Comment              | State<br>PseudoState<br>Transition |
| State (UML)   | xmi_id                                 | Detailing Behavior                 |

|                        |  |   |
|------------------------|--|---|
|                        | Name<br>Comment                              | Outgoing<br>Incoming<br>OwnedRegion<br>DoActivity<br>ExitActivity<br>EntryActivity<br>ConnectionPoint (Entry Point/ Exit Point) |
| Pseudo State           | xmi_id<br>Name<br>Comment<br>PseudoStateKind | Outgoing Transition<br>Incoming Transition  |
| Transition (UML)       | xmi_id<br>Name<br>Comment                    | Source<br>Target<br>Source Pseudo State<br>Target Pseudo State<br>Trigger<br>Effect (Behavior)<br>Constraint                    |
| Event (UML)            | xmi_id<br>Name<br>Comment                    | EventKind   |
| Protocol State Machine | xmi_id<br>Name<br>Visibility                 | Region  |

|               |         |             |
|---------------|---------|-------------|
|               | Comment |             |
| Trigger (UML) | xmi_id  | Event (UML) |
|               | Name    |             |
|               | Comment |             |

### Sequence, Communication and Interaction Overview Diagram

| MEGA Concepts           | MetaAttribut              | MetaAssociation(End)   |
|-------------------------|---------------------------|--|
| Interaction UML         | xmi_id<br>Name<br>Comment | Gate<br>Fragment (Combined Fragment, State Invariant, ...)<br>LifeLine<br>Message<br>Action<br>Parameter<br>Operation (UML)<br>OwnedInteraction (UML)<br>OwnedInteractionOperand |
| Interaction Operand     | xmi_id<br>Name<br>Comment | Fragment (Combined Fragment, State Invariant, ...)<br>OwnedInteraction (UML)<br>OwnedInteractionOperand  |
| OccurrenceSpecification | xmi_id<br>Name<br>Comment | Event (UML)<br>Message   |
| ExecutionSpecification  | xmi_id<br>Name            | start<br>finish  |

|                   |  |   |
|-------------------|--|---|
|                   | Comment  |   |
| LifeLine          | xmi_id<br>Name<br>Comment                                | ElementRepresentedByALifeline<br>ElementCoveringLifeline<br>OwnedSelector |
| Combined Fragment | xmi_id<br>Name<br><br>InteractionOperatorKind<br>Comment | InteractionOperand<br>CoveredLifeLine                                     |
| Interaction Use   | xmi_id<br>Name<br>Comment                                | RefersTo<br>CoveredLifeLine   |
| State Invariant   | xmi_id<br>Name<br>Comment                                | InvariantConstraint<br>CoveredLifeLine                                    |
| Gate              | xmi_id<br>Name<br>Comment                                |   |
| Message UML       | xmi_id<br>Name<br>MessageKind<br>MessageSort<br>Comment  | Receiver Sender Connector   |

## Activity and Interaction Overview Diagram

| MEGA Concepts | MetaAttribut   | MetaAssociation(End)   |
|---------------|--|--|
| Action        | xmi_id<br>Name<br>IsLeaf<br>ActionKind<br>Comment      | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow<br>OwnerGroup<br>RequestOperation<br>CalledBehavior<br>StructuralFeatureElementManagedByAnAction<br>RequestSignal<br>Association<br>ClassManagedByAnAction<br>Variable<br>InputPin<br>OutputPin<br>ProtectingExceptionHandler<br>Trigger<br>LocalPostCondition<br>LocalPreCondition<br>Constraint |
| Control Node  | xmi_id<br>Name<br>IsLeaf<br>Comment<br>ControlNodeType | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow<br>OwnerGroup   |

|                     |   |  |
|---------------------|---|--|
| Input Pin           | xmi_id<br>Name<br>IsLeaf<br>InputPinKind<br>ControlType<br>OrderingKind<br>Comment  | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow<br>Constraint |
| Output Pin          | xmi_id<br>Name<br>IsLeaf<br>OutputPinKind<br>ControlType<br>OrderingKind<br>Comment | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow<br>Constraint |
| Exchange Pin        | xmi_id<br>Name<br>IsLeaf<br>ControlType<br>OrderingKind<br>Comment                  | OutgoingObjectFlow<br>IncomingObjectFlow<br>OwnerGroup   |
| Central Buffer Node | xmi_id<br>Name<br>IsLeaf  | OutgoingObjectFlow<br>IncomingObjectFlow<br>OwnerGroup   |

|                    |  |  |
|--------------------|--|--|
|                    | ControlType<br>OrderingKind<br>Comment                             |  |
| Data Store Node    | xmi_id<br>Name<br>IsLeaf<br>ControlType<br>OrderingKind<br>Comment | OutgoingObjectFlow<br>IncomingObjectFlow<br>OwnerGroup<br>Constraint |
| Activity Partition | xmi_id<br>Name<br>IsDimension<br>IsExternal<br>Comment             | ContainedElement<br>Constraint                                       |
| Object Flow        | xmi_id<br>Name<br>IsLeaf<br>Comment                                | Guard<br>Weight<br>SourceElement<br>TargetElement<br>Constraint      |
| Control Flow       | xmi_id<br>Name<br>IsLeaf<br>Comment                                | Guard<br>Weight<br>SourceElement<br>TargetElement                    |

|                          |                 |                              |
|--------------------------|-----------------|------------------------------|
| Exception Handler        | xmi_id          | Constraint                   |
|                          | Name            | ProtectedNode                |
|                          | Comment         | ExceptionInput<br>Constraint |
| Activity UML             | xmi_id          | ElementOwnedByAnActivityUML  |
|                          | Name            | Constraint                   |
|                          | Reentrant       |                              |
|                          | SingleExecution |                              |
|                          | IsLeaf          |                              |
|                          | Comment         |                              |
| Activity Parameter Node  | xmi_id          | OutgoingObjectFlow           |
|                          | Name            | IncomingObjectFlow           |
|                          | IsLeaf          | Constraint                   |
|                          | ControlType     |                              |
|                          | OrderingKind    |                              |
|                          | Comment         |                              |
| Structured Activity Node | xmi_id          | OutgoingControlFlow          |
|                          | Name            | OutgoingObjectFlow           |
|                          | IsLeaf          | IncomingControlFlow          |
|                          | MustIsolate     | IncomingObjectFlow           |
|                          | Comment         | OwnerGroup                   |
|                          |                 | ContainedElement<br>InputPin |

|                               |   |  |
|-------------------------------|---|--|
|                               |   | OutputPin<br>Constraint  |
| Expansion Node                | xmi_id<br>Name<br>IsLeaf<br>ControlType<br>OrderingKind<br>Comment  | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow<br>Region<br>Constraint   |
| Expansion Region              | xmi_id<br>Name<br>IsLeaf<br>MustIsolate<br>ExpansionKind<br>Comment | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow<br>OwnerGroup<br>ContainedElement<br>ExpansionNode<br>InputElement<br>OutputElement |
| Interruptible Activity Region | xmi_id<br>Name<br>Comment   | OwnerGroup<br>ContainedElement<br>Constraint   |
| Loop Node                     | xmi_id<br>Name<br>IsLeaf  | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow   |

|                  |  |   |
|------------------|--|---|
|                  | MustIsolate<br>TestedFirst<br>Comment  | IncomingObjectFlow<br>OwnerGroup<br>ContainedElement<br>InputPin<br>OutputPin<br>Constraint<br>Test   |
| Conditional Node | xmi_id<br>Name<br>IsLeaf<br>MustIsolate<br>Assured<br>Determinate<br>Comment | OwnerGroup<br>ContainedElement<br>InputPin<br>OutputPin<br>Constraint<br>OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow |
| Sequence Node    | xmi_id<br>Name<br>IsLeaf<br>MustIsolate<br>Comment                           | OutgoingControlFlow<br>OutgoingObjectFlow<br>IncomingControlFlow<br>IncomingObjectFlow<br>OwnerGroup<br>ContainedElement<br>InputPin<br>OutputPin<br>Constraint |

## Package Diagram

| MEGA Concepts | MetaAttribut                            | MetaAssociation(End)  |
|---------------|---|---|
| Package       | Name<br>xmi_id<br>Visibility<br>Comment | Client Dependency<br>Package Target Dependency<br>Owned Class<br>Owned Package<br>Owned Association<br>Association Class<br>Owned Dependency<br>Owned Element (UML): Generalization<br>Owned Use Case<br>Owned Actor (UML)<br>Constraint<br>Behavior: State Machine<br>Behavior: Protocol State Machine<br>Behavior: Activity Uml<br>Behavior: Interaction Uml<br>Behavior: Collaboration Uml<br>Owned Component<br>Owned Event |

## Component Diagram

| MEGA Concepts | MetaAttribut              | MetaAssociation(End)                   |
|---------------|---------------------------|--|
| Component     | xmi_id<br>Name<br>Comment | required Interface<br>provided<br>Port |

|           |  |  |
|-----------|--|--|
|           | Isleaf<br>Visibility<br>Client Dependency  | OwnedPart  |
| Port      | xmi_id<br>Name<br>Comment<br>Client Dependency                                     |  |
| Interface | xmi_id<br>Name<br>Visibility<br>Comment<br>Abstract<br>IsLeaf<br>Client Dependency | Class Target Dependency<br>Nested Class<br>Association<br>Association Class<br>Attribute<br>Operation (UML)<br>Generalization<br>RequiredInterface<br>SpecificationInterface |

## Deployment Diagram

| MEGA Concepts | MetaAttribut                                   | MetaAssociation(End)  |
|---------------|--|---|
| Artifact UML  | xmi_id<br>Name<br>Comment<br>Client Dependency | Target Dependency<br>OwnedAttribute<br>OwnedOperation<br>NestedArtifact |
| Node UML      | xmi_id   | Deployment  |

|   |  |   |
|---|--|---|
|   | Name<br>Comment<br>Client Dependency           |   |
| Device  | xmi_id<br>Name<br>Comment<br>Client Dependency | Deployment                                |
| Execution Environment   | xmi_id<br>Name<br>Comment<br>Client Dependency | Deployment                                |
| Deployment Specification  | xmi_id<br>Name<br>Comment<br>Client Dependency | Target Dependency                         |
| Instance ( Node<br>UML/Device/Execution<br>Specification/Component) | xmi_id<br>Name<br>Comment<br>Client Dependency | Instantiated Element<br>Target Dependency |
| Communication Path  | xmi_id<br>Name<br>Comment                      | Communication Path End                    |

|               |                           |  |
|---------------|---------------------------|--|
| Deployment    | xmi_id<br>Name<br>Comment | Deployed Element<br>Deployment Configuration                 |
| Manifestation | xmi_id<br>Name<br>Comment | Multiplicity<br>Deployed Element<br>Deployment Configuration |

## Object Diagram

| MEGA Concepts | MetaAttribut              | MetaAssociation(End) |
|---------------|---------------------------|----------------------|
| Instance      | xmi_id<br>Name<br>Comment |                      |
| Link          | xmi_id<br>Name<br>Comment | LinkEnd              |
| LinkEnd       | xmi_id<br>Name<br>Comment | Instance             |

HOPEX XMI 2.1 Export for UML2

## **XMI EXPORT OVERVIEW**

---

The XML Metadata Interchange XMI is an OMG standard for exchanging UML Models between different UML products such as modeling tools and UML Design.

The XMI 2.1 Export project aims at exporting the content of HOPEX Diagrams as .xmi files so that models modeled in HOPEX can be imported by UML tools such as Eclipse EMF.

### **Prerequisites**

The XMI 2.1 export feature is available with HOPEX UML, and supports XMI version 2.1 with UML 2.3.

### **Scope of XMI Export**

The purpose of XMI export is to translate the specification of HOPEX Class Diagrams, Use Case Diagrams, Component Diagram, Composite Structure Diagram, Activity Diagram, Communication Diagram, Sequence Diagram and State Machine Diagrams into XMI. Diagrams and diagrams drawings are not considered except with UML2 plugin for Eclipse.

The tool handles translation of the concepts of the above HOPEX diagrams that have a correspondence in UML 2.0. The list of supported mappings is detailed below.

## EXPORTING XMI FILES

---

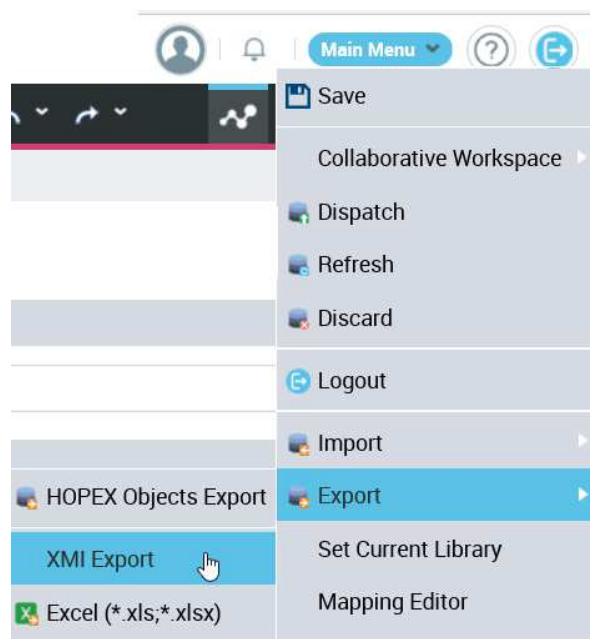
Depending on the destination tool, the XMI export tool produces two types of file. For Eclipse with UML plugin, the export will produce one .uml file for data and some .umlclass, .umlusc and/or .umlstm files for diagrams (each file represents an HOPEX diagram).

For other modeling tools, as Enterprise Architect or MagicDraw UML, the export will produce one .xmi file for data but no diagram description files will be generated.

### Export for Eclipse with UML2 plugin

To export HOPEX data for Eclipse with UML2 plugin:

1. In HOPEX, select **Main Menu > Export > XMI Export**.



The export dialog box appears.

2. Select the **Package** to be exported.
3. Specify the name and path of the file to be exported.
4. Under **Options**, check the **Export for Eclipse** parameter.
5. Click **Next**.

The window shows the export process progression.

Then the window showing the report of all exported data appears.

## Export for other tools

Because many modeling tools do not support the UML Diagram Interchange Specification, the MEGA XMI 2.1 Export feature exports .xmi file for data but no diagram description files for other tools than Eclipse.

To export HOPEX data for modeling tools such as Enterprise Architect or MagicDraw UML:

1. In HOPEX, select **Main Menu > Export > XMI Export**.

The export dialog box appears.

2. Select the **Package** to be exported.
3. Specify the name and path of the file to be exported.
4. **Uncheck the Export for Eclipse** parameter.
5. Click **Next >**.

The window that appears shows the export process progression.

Then the window showing the report of all exported data appears.

## HOPEX/XMI OBJECT MAPPING

---

The XMI export feature translates a class diagram or use case diagram or state machine diagram or even protocol state machine diagram specified in HOPEX into an XMI compliant output file.

The following paragraph indicates what kinds of objects are exported by UML2 diagram types. Only objects belonging to the selected package are exported.

### **Class Diagram**

Packages, classes, interfaces, enumerations, literal strings (expression text), associations, association roles, generalizations, constraints, required interfaces (link), provided interfaces (supported interface link), data types (class stereotype), primitive types (class stereotype), attributes, operations.

### **Use Case Diagram**

Use cases, actors, packages, constraints, extension points (text), participations, extensions (link), inclusions (link), generalization, dependencies.

### **State Machine Diagram**

State machines, regions, states, pseudo states, transitions, constraints

### **Protocol State Machine Diagram**

Protocol state machines, regions, states, pseudo states, transitions, constraints

### **Activity UML Diagram**

Actions, control nodes, Input Pins, Output Pins, Exchange Pins, central buffer nodes, data store nodes, activity partitions, control flows, object flows, exception handlers, activities uml, activity parameter nodes, structured activity nodes, expansion regions, expansion nodes, interruptible activity regions.

### **Component Diagram**

Components, ports, packages, interfaces, required interfaces, provided interfaces, classes, Connectors, realized elements.

### **Composite Structure Diagram**

Collaborations UML, collaboration uses, parts, dependencies, connectors, interfaces, classes, provided interfaces, required interfaces.

## Sequence Diagram

Life lines, combined fragments, interaction uses, gates, states invariant, messages UML, constraints.

---

Messages include those exchanged directly between lifelines as well as messages exchanged through execution specification.

---

## Communication Diagram

Life lines, connectors, messages UML.





















## Deployment Diagram























Packages, components, artifacts UML, nodes UML, devices, execution environments, interfaces, deployment specifications, deployments, manifestations, deployment configurations, component instances, device instances, node instances, execution environment instances, communication paths.























🔗\* Only objects owned by the selected package or its sub-packages are exported.























Objects that are linked to objects contained in the selected export package but owned by another package are also exported in order to ensure links. However, they will be owned by the exported package.

The following table indicates concepts managed by the export tool:

| HOPEX Concepts  |  |
|---|--|
|    | Package                                |
|    | Name                                   |
|    | _Hexaidabs                             |
|    | Visibility                             |
|    | Comment                                |
|    | Client Dependency                      |
|    | Package Target Dependency              |
|  | Owned Class                            |
|  | Owned Package                          |
|  | Owned Association                      |
|  | Association Class                      |
|  | Owned Dependency                       |
|  | Owned Element (UML): GeneralizationSet |
|  | Owned Use Case                         |
|  | Owned Actor (UML)                      |
|  | Constraint                             |
|  | Behavior: State Machine                |
|  | Behavior: Protocol State Machine       |
|  | Behavior: Activity Uml                 |
|  | Behavior: Interaction Uml              |























| HOPEX Concepts  |
|---|
| <ul style="list-style-type: none"> <li> Behavior: Collaboration Uml</li> <li> Owned Component</li> <li> Owned Event</li> </ul>   |
| <div data-bbox="240 611 347 649"> <b>Class</b></div> <ul style="list-style-type: none"> <li> Name</li> <li> _Hexaidabs</li> <li> Visibility</li> <li> Comment</li> <li> Abstract</li> <li> IsLeaf</li> <li> IsActive</li> <li> Client Dependency</li> <li> Class Target Dependency</li> <li> Realization Class</li> <li> Nested Class</li> <li> Association</li> <li> Connector</li> <li> Association Class</li> <li> Attribute</li> <li> Operation (UML)</li> <li> Generalization</li> <li> Required Interface</li> </ul> |










| HOPEX Concepts  |                             |
|---|-----------------------------|
|    | Provided Interface          |
|    | Constraint                  |
|    | Method                      |
|    | Port                        |
|    | AssociationEnd              |
|    | Owned Part                  |
|    | Behavior: State Machine     |
|    | Behavior: Activity Uml      |
|   | Behavior: Interaction Uml   |
|  | Behavior: Collaboration Uml |
|  | <b>Data Type</b>            |
|  | Name                        |
|  | _Hexaidabs                  |
|  | Visibility                  |
|  | Abstract                    |
|  | IsLeaf                      |
|  | IsActive                    |
|  | Client Dependency           |
|  | Class Target Dependency     |
|  | Nested Class                |
|  | <b>Interface</b>            |
|  | _Hexaidabs                  |























| HOPEX Concepts  |                         |
|---|-------------------------|
|    | Name                    |
|    | Visibility              |
|    | Comment                 |
|    | Abstract                |
|    | IsLeaf                  |
|    | Client Dependency       |
|    | Class Target Dependency |
|    | Nested Class            |
|   | Association             |
|  | Association Class       |
|  | Attribute               |
|  | Operation (UML)         |
|  | Generalization          |
|  | RequiredInterface       |
|  | SpecificationInterface  |
|  | <b>Enumeration</b>      |
|  | _Hexaidabs              |
|  | Name                    |
|  | Visibility              |
|  | Comment                 |
|  | Client Dependency       |
|  | Class Target Dependency |























| HOPEX Concepts  |  |
|---|--|
| <ul style="list-style-type: none"> <li>Attribute</li> <li>Operation (UML)</li> <li>Literal Value</li> <li>RequiredInterface</li> <li>Specification Interface</li> </ul>   |  |
| <b>Expression</b> <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>_Hexaidabs</li> <li>Name</li> <li>Visibility</li> <li>Comment</li> <li>Client Dependency</li> </ul> </li> <li>Class Target Dependency</li> <li>Specification Interface</li> </ul>  |  |
| <b>Primitive Type</b> <ul style="list-style-type: none"> <li> <ul style="list-style-type: none"> <li>Name</li> <li>_Hexaidabs</li> <li>Visibility</li> <li>Abstract</li> <li>IsLeaf</li> <li>IsActive</li> </ul> </li> <li>Class Target Dependency</li> <li>Nested Class</li> </ul> |  |























| HOPEX Concepts   |  |
|--|--|
| <div> <div> <div></div> <div>Association</div> </div> <div> <div></div> <div>_Hexaidabs</div> </div> <div> <div></div> <div>Name</div> </div> <div> <div></div> <div>Visibility</div> </div> <div> <div></div> <div>Comment</div> </div> <div> <div></div> <div>IsAssociationDerived</div> </div> <div> <div></div> <div>IsNavigable</div> </div> <div> <div></div> <div>Connection</div> </div> <div> <div></div> <div>Dependency (Target Association)</div> </div> <div> <div></div> <div>Class via AssociationEnd</div> </div> </div>   |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| <div> <div> <div></div> <div>Association Class</div> </div> <div> <div></div> <div>_Hexaidabs</div> </div> <div> <div></div> <div>Name</div> </div> <div> <div></div> <div>Visibility</div> </div> <div> <div></div> <div>Comment</div> </div> <div> <div></div> <div>IsLeaf</div> </div> <div> <div></div> <div>Abstract</div> </div> <div> <div></div> <div>IsActive</div> </div> <div> <div></div> <div>IsAssociationDerived</div> </div> <div> <div></div> <div>IsNavigable</div> </div> <div> <div></div> <div>Class Target Dependency</div> </div> <div> <div></div> <div>Nested Class</div> </div> </div> |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

| HOPEX Concepts  |                                       |
|---|---------------------------------------|
|    | AssociationEnd                        |
|    | Association                           |
|    | Association Class                     |
|    | Class via AssociationEnd              |
|    | Attribute                             |
|    | Operation (UML)                       |
|    | <b>Attribute</b>                      |
|    | _Hexaidabs                            |
|    | Name                                  |
|  | Visibility                            |
|  | Comment                               |
|  | IsOrdred                              |
|  | Uniqueness                            |
|  | ReadOnly                              |
|  | IsDerived                             |
|  | InitialValue                          |
|  | Multiplicity : UpperValue, LowerValue |
|  | Dependency (Target Attribute)         |
|  | AttributType                          |
|  | OverloadedAttribute                   |
|  | <b>Operation (UML)</b>                |
|  | _Hexaidabs                            |























| HOPEX Concepts  |                       |
|---|-----------------------|
|    | Name                  |
|    | Visibility            |
|    | Comment               |
|    | Abstract              |
|    | IsQuery               |
|    | Precondition          |
|    | Postcondition         |
|    | Method                |
|    | Parameter             |
|  | ReturnType            |
|  | Target Dependency     |
|  | <b>Dependency</b>     |
|  | _Hexaidabs            |
|  | Name                  |
|  | Visibility            |
|  | Comment               |
|  | Class Source          |
|  | Class Target          |
|  | <b>Generalization</b> |
|  | _Hexaidabs            |
|  | Name                  |
|  | Comment               |






















| HOPEX Concepts  |                           |
|---|---------------------------|
|    | Super Class               |
|    | UML constraint            |
|    | <b>GeneralizationSet</b>  |
|    | _Hexaidabs                |
|    | Name                      |
|    | Comment                   |
|    | IsComplete                |
|    | IsDisjoint                |
|    | Target Dependency         |
|  | Generalization            |
|  | <b>Constraint</b>         |
|  | _Hexaidabs                |
|  | Name                      |
|  | Comment                   |
|  | MaxInt                    |
|  | MinInt                    |
|  | ConstrainedClass          |
|  | ConstrainedGeneralization |
|  | ConstrainedElement        |
|  | Actor (UML)               |
|  | Package                   |
|  | UseCase                   |

| HOPEX Concepts  |                                  |
|---|----------------------------------|
|    | UseCaseParticipation             |
|    | <b>Parameter</b>                 |
|    | _Hexaidabs                       |
|    | Name                             |
|    | Comment                          |
|    | Parameter Type                   |
|    | <b>Behavior (UML)</b>            |
|    | _Hexaidabs                       |
|   | Name                             |
|  | Specification                    |
|  | <b>UseCase</b>                   |
|  | _Hexaidabs                       |
|  | Name                             |
|  | Visibility                       |
|  | Comment                          |
|  | UsesUseCase                      |
|  | OwnedExtension                   |
|  | ExtensionPoint                   |
|  | Behavior: State Machine          |
|  | Behavior: Protocol State Machine |
|  | Behavior: Interaction UML        |
|  | Behavior: Activity UML           |






| HOPEX Concepts   |  |
|--|--|
|  Constraint<br><br> Generalization   |  |
|  |  |
|  <b>Actor (UML)</b>   |  |
|  _Hexaidabs<br><br> Name<br><br> Visibility<br><br> Comment<br><br> Participation<br><br> OwnedExtension<br><br> Constraint<br><br> Generalization |  |
|  <b>Participation</b>   |  |
|  _Hexaidabs<br><br> Name<br><br> Comment<br><br> Multiplicity<br><br> UseCase<br><br> Actor (UML)<br><br> Constraint  |  |
|  <b>Extension</b>   |  |
|  _Hexaidabs<br><br> Name   |  |






















| HOPEX Concepts   |
|--|
| <ul style="list-style-type: none"> <li>Comment</li> <li>Extended Use Case</li> <li>Extension Location</li> </ul>   |
| <b>State Machine</b> <ul style="list-style-type: none"> <li>_Hexaidabs</li> <li>Name</li> <li>Comment</li> <li>Reentrant</li> <li>DetailedState</li> <li>Region</li> </ul> |
| <b>Region</b> <ul style="list-style-type: none"> <li>_Hexaidabs</li> <li>Name</li> <li>Comment</li> <li>State</li> <li>PseudoState</li> <li>Transition</li> </ul>          |
| <b>State (UML)</b> <ul style="list-style-type: none"> <li>_Hexaidabs</li> <li>Name</li> <li>Comment</li> <li>Detailing Behavior</li> </ul>                                 |























| HOPEX Concepts  |                         |
|---|-------------------------|
|    | Outgoing                |
|    | Incoming                |
|    | OwnedRegion             |
|    | OwnedRegion             |
|    | DoActivity              |
|    | ExitActivity            |
|    | EntryActivity           |
|    | <b>Pseudo State</b>     |
|   | _Hexaidabs              |
|  | Name                    |
|  | Comment                 |
|  | PseudoStateKind         |
|  | Outgoing Transition     |
|  | Incoming Transition     |
|  | <b>Transition (UML)</b> |
|  | _Hexaidabs              |
|  | Name                    |
|  | Comment                 |
|  | Source                  |
|  | Target                  |
|  | Source Pseudo State     |
|  | Target Pseudo State     |























| HOPEX Concepts   |  |
|--|--|
|  Trigger<br> Effect (Behavior)<br> Constraint   |  |
|  |  |
|  |  |
|  <b>Event (UML)</b>   |  |
|  _Hexaidabs<br> Name<br> Comment  |  |
|  <b>Protocol State Machine</b>   |  |
|  _Hexaidabs<br> Name<br> Reentrant<br> Comment<br> Region   |  |
|  <b>Action</b>  |  |
|  _Hexaidabs<br> Name<br> IsLeaf<br> ActionKind<br> Comment<br> OutgoingControlFlow<br> OutgoingObjectFlow |  |























| HOPEX Concepts  |   |
|---|---|
|    | IncomingControlFlow                       |
|    | IncomingObjectFlow                        |
|    | OwnerGroup                                |
|    | RequestOperation                          |
|    | CalledBehavior                            |
|    | StructuralFeatureElementManagedByAnAction |
|    | RequestSignal                             |
|    | Association                               |
|    | ClassManagedByAnAction                    |
|  | Variable                                  |
|  | InputPin                                  |
|  | OutputPin                                 |
|  | ProtectingExceptionHandler                |
|  | Trigger                                   |
|  | LocalPostCondition                        |
|  | LocalPreCondition                         |
|  | <b>Control Node</b>                       |
|  | _Hexaidabs                                |
|  | Name                                      |
|  | IsLeaf                                    |
|  | Comment                                   |
|  | OutgoingControlFlow                       |









| HOPEX Concepts   |
|--|
| <ul style="list-style-type: none"> <li> OutgoingObjectFlow</li> <li> IncomingControlFlow</li> <li> IncomingObjectFlow</li> <li> OwnerGroup</li> </ul>  |
| <div data-bbox="245 680 395 719"> <b>Input Pin</b></div> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> IsLeaf</li> <li> ControlType</li> <li> OrderingKind</li> <li> Comment</li> <li> OutgoingControlFlow</li> <li> OutgoingObjectFlow</li> <li> IncomingControlFlow</li> <li> IncomingObjectFlow</li> </ul> |
| <div data-bbox="245 1453 416 1491"> <b>Output Pin</b></div> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> IsLeaf</li> <li> ControlType</li> <li> OrderingKind</li> <li> Comment</li> </ul>  |

| HOPEX Concepts  |
|---|
| <ul style="list-style-type: none"> <li> OutgoingControlFlow</li> <li> OutgoingObjectFlow</li> <li> IncomingControlFlow</li> <li> IncomingObjectFlow</li> </ul>  |
| <div data-bbox="240 678 448 719"> <b>Exchange Pin</b></div> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> IsLeaf</li> <li> ControlType</li> <li> OrderingKind</li> <li> Comment</li> <li> OutgoingControlFlow</li> <li> OutgoingObjectFlow</li> <li> IncomingControlFlow</li> <li> IncomingObjectFlow</li> <li> OwnerGroup</li> </ul> |
| <div data-bbox="240 1523 531 1563"> <b>Central Buffer Node</b></div> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> IsLeaf</li> <li> ControlType</li> <li> OrderingKind</li> </ul>   |























| HOPEX Concepts  |                           |
|---|---------------------------|
|    | Comment                   |
|    | OutgoingObjectFlow        |
|    | IncomingObjectFlow        |
|    | OwnerGroup                |
|    | <b>Data Store Node</b>    |
|    | _Hexaidabs                |
|    | Name                      |
|    | IsLeaf                    |
|    | ControlType               |
|  | OrderingKind              |
|  | Comment                   |
|  | OutgoingObjectFlow        |
|  | IncomingObjectFlow        |
|  | OwnerGroup                |
|  | <b>Activity Partition</b> |
|  | _Hexaidabs                |
|  | Name                      |
|  | IsDimension               |
|  | IsExternal                |
|  | Comment                   |
|  | ContainedElement          |
|  | <b>Object Flow</b>        |














| HOPEX Concepts   |               |
|--|---------------|
|                             | _Hexaidabs    |
|                             | Name          |
|                             | IsLeaf        |
|                             | Comment       |
|                             | Guard         |
|                             | Weight        |
|                             | SourceElement |
|                             | TargetElement |
|  <b>Control Flow</b>        |               |
|                           | _Hexaidabs    |
|                           | Name          |
|                           | IsLeaf        |
|                           | Comment       |
|                           | Guard         |
|                           | Weight        |
|                           | SourceElement |
|                           | TargetElement |
|  <b>Exception Handler</b> |               |
|                           | _Hexaidabs    |
|                           | Name          |
|                           | Comment       |
|                           | ProtectedNode |























| HOPEX Concepts  |
|---|
|  ExceptionInput  |
|  <b>Activity UML</b> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> Reentrant</li> <li> SingleExecution</li> <li> IsLeaf</li> <li> Comment</li> </ul>  ElementOwnedByAnActivityUML   |
|  <b>Activity Parameter Node</b> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> IsLeaf</li> <li> ControlType</li> <li> OrderingKind</li> <li> Comment</li> </ul>  OutgoingObjectFlow  IncomingObjectFlow |
|  <b>Structured Activity Node</b> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> IsLeaf</li> </ul>  |























| HOPEX Concepts  |
|---|
| <ul style="list-style-type: none"> <li> MustIsolate</li> <li> Comment</li> <li> OutgoingControlFlow</li> <li> OutgoingObjectFlow</li> <li> IncomingControlFlow</li> <li> IncomingObjectFlow</li> <li> OwnerGroup</li> <li> ContainedElement</li> </ul>  |
| <p> <b>Expansion Node</b></p> <ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> IsLeaf</li> <li> ControlType</li> <li> OrderingKind</li> <li> Comment</li> <li> OutgoingControlFlow</li> <li> OutgoingObjectFlow</li> <li> IncomingControlFlow</li> <li> IncomingObjectFlow</li> <li> Region</li> </ul> |
| <p> <b>Expansion Region</b></p> <ul style="list-style-type: none"> <li> _Hexaidabs</li> </ul>   |

| HOPEX Concepts  |                                      |
|---|--------------------------------------|
|    | Name                                 |
|    | IsLeaf                               |
|    | MustIsolate                          |
|    | ExpansionKind                        |
|    | Comment                              |
|    | OutgoingControlFlow                  |
|    | OutgoingObjectFlow                   |
|    | IncomingControlFlow                  |
|    | IncomingObjectFlow                   |
|  | OwnerGroup                           |
|  | ContainedElement                     |
|  | ExpansionNode                        |
|  | <b>Interruptible Activity Region</b> |
|  | _Hexaidabs                           |
|  | Name                                 |
|  | Comment                              |
|  | ContainedElement                     |
|  | <b>Collaboration uml</b>             |
|  | _Hexaidabs                           |
|  | Name                                 |
|  | Comment                              |
|  | IsAbstract                           |

| HOPEX Concepts  |
|---|
| <ul style="list-style-type: none"> <li> IsLeaf</li> <li> CollaborationRole</li> <li> OwnedConnector</li> <li> OwnedCollaborationUse</li> </ul>  |
| <ul style="list-style-type: none"> <li> <b>Collaboration use</b></li> <li><ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> Comment</li> </ul> </li> <li> Type</li> <li> Dependency</li> </ul>  |
| <ul style="list-style-type: none"> <li> <b>Part</b></li> <li><ul style="list-style-type: none"> <li> _Hexaidabs</li> <li> Name</li> <li> Visibility</li> <li> Client Dependency</li> <li> IsUnique</li> <li> IsOrdered</li> <li> Multiplicity</li> <li> Comment</li> </ul> </li> <li> ConnectorEnd (of the LifeLine who represents the part)</li> <li> Dependency</li> </ul> |
| <ul style="list-style-type: none"> <li> <b>Connector</b></li> </ul>  |

| HOPEX Concepts  |                               |
|---|-------------------------------|
|    | _Hexaidabs                    |
|    | Name                          |
|    | Connector Kind                |
|    | IsLeaf                        |
|    | Comment                       |
|    | OwnedConnectorEnd             |
|    | <b>LifeLine</b>               |
|    | _Hexaidabs                    |
|    | Name                          |
|  | Comment                       |
|  | ElementRepresentedByALifeline |
|  | ElementCoveringLifeline       |
|  | OwnedSelector                 |
|  | <b>Combined Fragment</b>      |
|  | _Hexaidabs                    |
|  | Name                          |
|  | InteractionOperatorKind       |
|  | Comment                       |
|  | InteractionOperand            |
|  | CoveredLifeLine               |
|  | <b>Interaction Use</b>        |
|  | _Hexaidabs                    |

| HOPEX Concepts  |                         |
|---|-------------------------|
|    | Name                    |
|    | InteractionOperatorKind |
|    | Comment                 |
|    | RefersTo                |
|    | CoveredLifeLine         |
|    | <b>State Invariant</b>  |
|    | _Hexaidabs              |
|    | Name                    |
|    | Comment                 |
|  | InvariantConstraint     |
|  | CoveredLifeLine         |
|  | <b>Gate</b>             |
|  | _Hexaidabs              |
|  | Name                    |
|  | Comment                 |
|  | <b>Message UML</b>      |
|  | _Hexaidabs              |
|  | Name                    |
|  | MessageKind             |
|  | Comment                 |
|  | Receiver                |
|  | Sender                  |

| HOPEX Concepts  |                              |
|---|------------------------------|
|    | Connector                    |
|    | <b>Artifact UML</b>          |
|    | _Hexaidabs                   |
|    | Name                         |
|    | Comment                      |
|    | Client Dependency            |
|    | Target Dependency            |
|    | <b>Node UML</b>              |
|   | _Hexaidabs                   |
|  | Name                         |
|  | Comment                      |
|  | Client Dependency            |
|  | Deployment                   |
|  | <b>Device</b>                |
|  | _Hexaidabs                   |
|  | Name                         |
|  | Comment                      |
|  | Client Dependency            |
|  | Deployment                   |
|  | <b>Execution Environment</b> |
|  | _Hexaidabs                   |
|  | Name                         |

| HOPEX Concepts   |
|--|
| <ul style="list-style-type: none"> <li>Comment</li> <li>Client Dependency</li> <li>Deployment</li> </ul>   |
| <b>Deployment Specification</b> <ul style="list-style-type: none"> <li>_Hexaidabs</li> <li>Name</li> <li>Comment</li> <li>Client Dependency</li> <li>Target Dependency</li> </ul>  |
| <b>Instance ( Node UML/Device/Execution Specification/Component)</b> <ul style="list-style-type: none"> <li>_Hexaidabs</li> <li>Name</li> <li>Comment</li> <li>Client Dependency</li> <li>Instantiated Element</li> <li>Target Dependency</li> </ul> |
| <b>Communication Path</b> <ul style="list-style-type: none"> <li>_Hexaidabs</li> <li>Name</li> <li>Comment</li> <li>Communication Path End</li> <li>Target Dependency</li> </ul>   |

| HOPEX Concepts                                     |   |
|--|---|
| <div> <div></div> <b>Communication Path</b> </div> | <div> <div></div> _Hexaidabs </div>               |
|  | <div> <div></div> Name </div>                     |
|  | <div> <div></div> Comment </div>                  |
|  | <div> <div></div> Multiplicity </div>             |
|  | <div> <div></div> Deployment Target </div>        |
|  |   |
| <div> <div></div> <b>Deployment</b> </div>         | <div> <div></div> _Hexaidabs </div>               |
|  | <div> <div></div> Name </div>                     |
|  | <div> <div></div> Comment </div>                  |
|  | <div> <div></div> Multiplicity </div>             |
|  | <div> <div></div> Deployed Element </div>         |
|  | <div> <div></div> Deployment Configuration </div> |