# Hopex IT Architecture

## User Guide

Hopex Aquila

Bizzdesign

# CONTENTS

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# Contents

# INTRODUCTION

**Hopex IT Architecture** allows IT managers to formalize business needs in order to define the architecture of the information system that meets them, from the logical architecture to the technical infrastructure.

**Hopex IT Architecture** offers facilities for different analysis perspectives:

- ✓ **Information System management and upgrading**: a description of service and city planning architectures are two approaches that simplify IS upgrading by providing a frame of reference for planning your systems and analyzing your upgrading scenarios.
- ✓ **Application mapping**: a description of application architecture that offers a detailed view of information exchanges between applications, services, databases and organizational units.
- ✓ **Application deployment**: a description of the information system technical infrastructure to monitor application deployment on the different enterprise sites. The technical infrastructure takes account of the main hardware of your organization such as networks, servers, workstations, printers, firewalls and concentrators.
- ✓ **The representation of resource architectures**: a description of complex systems involving different types of IT resources.

In addition to **Hopex IT Architecture**, **Hopex IT Business Management** allows organizations to manage their information system transformation by offering possibilities to define steps and to manage assessments for each of the steps.

**Hopex IT Architecture** also offers a tool used to import configuration elements from CMDB (Configuration Management DataBase) and align them with modeling objects described in **Hopex IT Architecture**. For more information, see the "CMDB Import" documentation.

The purpose of this guide is therefore to present how to make best use of these functionalities for the successful evolution of your information system.

The following points are covered in **Hopex IT Architecture**:

- ✓ Modeling Applications and System Architectures.
- ✓ Modeling application architectures.
- ✓ Modeling technical architectures.
- ✓ Aligning IT and Business.
- ✓ Modeling IT Infrastructures.
- ✓ Accessing the Software Design.
- ✓ Describing information exchanges.
- ✓ About UML implementation.

# PRESENTATION OF HOPEX IT ARCHITECTURE

Combined with the products of the **Hopex** suite, **Hopex IT Architecture** supports a methodology and the tools used to describe, analyze and plan your information system transformation.

## The Scope Covered by Hopex IT Architecture

The modules offered in standard mode are used to follow a top-down approach, beginning with a review of the business capabilities of the enterprise and its strategy, and ending with a precise definition of the components of the existing or future information system.

Each module addresses specific user profiles. Standard reports are offered to simplify analysis of the subjects handled.

The method described in this guide is represented by the modules described below.

> ☛ *The order of use of these modules is given by way of information.*

***Describing and analyzing flows***: this step is based on scenario diagrams that represent the flows between the components of your information system.

> ☛ *For more details, see Describing flow scenarios.*

***Analyzing the functional coverage of the technical architecture***: during this step the reports proposed by **Hopex IT Architecture** are used to analyze the links between the components of the described application architecture and the expected functionalities.

> ☛ *For more details on modeling applications and services, see Describing Application Architecture and Describing Applications.*

***Describing the upgrade strategy of the information system and architecture***: this step consists in describing what the information system is able to deliver, through business capabilities, and how it plans to deliver them using the architectures.

> ☛ *For more details, see Analyzing the functional coverage of the architecture implemented and Building the Logical Architecture.*

***Describing the application environment***: this step consists in describing the deployment architecture and all the elements that compose it.

> ☛ *For more details, see Defining the Deployment Architecture of an Application.*

***Describing the technical infrastructure:*** this module allows to manage deployment constraints and to associate adapted solutions to them.

> ☛ *For more details, see Defining the technical infrastructure.*

***Using UML formalism***: furthermore, you can use the UML (Unified Modeling Language) modeling lnaguage to model your IS.

> ☛ *For more details, see About UML implementation.*

# Summary of Activities and Deliverables of Hopex IT Architecture

Activities are associated with each of the modules of the method we recommend for managing the evolution of your information system.

The **Hopex IT Architecture** solution offers the tools to carry out these activities, which are materialized by deliverables.

| Activities | Main deliverables |
|---|---|
| Defining the logical architecture | Logical architecture structure diagrams, see Describing Logical Application Architecture. |
| Building the application architecture | Application architecture structure diagrams and flow scenario diagrams, see Describing an Application with Hopex IT Architecture. |
| Analyzing the functional coverage of the application architecture | Assessing the functional coverage by software resources, see Describing the fulfillment of a Functionality. Assessing the coverage of technical functionalities by technical resources, see Describing the fulfillment of a Functionality. |
| Defining the deployment architecture | Description of the technical requirements for the application deployment, see Modeling technical architectures. |
| Defining the infrastructure | Description of the technical requirements for the application deployment, see Modeling IT Infrastructures. |
| Managing service catalogs | Description of service catalogs and recommended solutions, see Using service catalogs. |

*Presentation of the **Hopex IT Architecture** deliverables*

# Structure and positioning of the Hopex IT Architecture solution

**Hopex IT Architecture** can be used with other products in the **Hopex** suite.

### Hopex IT Business Management

**Hopex IT Business Management** Solution provides **Hopex IT Architecture** with method and tools for business transformation planning. Both solutions share the mapping functionality for business capabilities.

### Hopex Business Process Analysis

In addition to **Hopex IT Business Management**, **Hopex Business Process Analysis**solution provides the possibility to describe the organizations and processes that implement the business capabilities identified in **Hopex IT Architecture**;

# Hopex IT Architecture Profiles

In **Hopex IT Architecture**,there are profiles associated to specific activities.

Presentation of the solution interface depends on the profile selected by the user on connection to the application; the tree of menus and functions varies from one business role to another.

☛ *For more details on the Desktops connected to each of the profiles, see Hopex IT Architecture Desktop Presentation.*

| Profiles | Tasks |
|---|---|
| Solution Architecture Functional Administrator | In addition to the Solution Architect's functional rights, the Solution Architecture Functional Administrator has rights over all objects, methods, projects and workflows. He/she prepares the work environment and creates the elements required to manage the modeled elements. He/she manages the environment objects (application environments, infrastructures, reports, etc.), For more details, see Presenting the Solution Architecture Functional Administrator workspace menus. |
| Solution Architect | The Solution Architect has rights over all objects, methods, projects and assessments. The IT architect is responsible for building architecture models for the applications, IT technologies and IT infrastructures assigned to him/her. He/she can manage transformation projects. The Solution Architect is in charge of detailing the system specifications and designing UML diagrams. For more details, see Presentation of the Solution Architect workspace. |
| Application contributor | The Application contributor is in charge of validating the design of the applications assigned to them. For more details, see Presentation of the Application Designer workspace. |
| Application Viewer | The Application Viewer has read-only rights over the repository objects. For more details, see Presenting the Application Viewer workspace. |

# Business Roles of Hopex IT Architecture

In **Hopex IT Architecture**, there are, by default, business roles that can be assigned to certain users. These roles are:

- **Software Designer**: used to assign a user to software elements. The software designer is responsible for designing the software assigned to him/her.
- **Data Designer** , who is responsible for managing data.
- **Local Application Owner** used to assign a user to applications. The Local Application Owner is responsible for the following tasks:
  - Identifying risks
  - Responding to Questionnaires
  - Defining and implementing action plans,
  - Validating the modifications made by the architect in the context of object review workflows.
- The **Business Owner** specifies the characteristics for the software installations and applications they are in charge of at the business level.
- The **IT Owner** specifies the characteristics of the IT resources they are in charge of.

# THE HOPEX IT ARCHITECTURE METHOD

The method described in this paragraph is given by way of information. Depending on your work context, you can sequence differently the described steps.

## Describing Application Architecture

**Hopex IT Architecture** offers the means to represent different levels of application architectures: from the description of the application environment to the technical components to be implemented.

These representations make it possible to define the software and hardware components and to identify in a consistent way the data exchanged between them.

☛ *For more information about the use of an application architecture, see Modeling Applications and System Architectures.*

The description of *application systems* can be done according to a top-down approach, starting by describing the company's main application systems, or according to a unitary approach by describing only certain application systems.

📖 *An application system is an assembly of other application systems, applications and end users interacting with application components to implement one or several functions.*

### Application system environment description

If you use a unitary approach, you must describe the *application system environment* to provide a context to the used application system and its service interactions with external components.

📖 *An application system environment allows presenting the other application systems, applications or microservices with which this application system can interact.*

☛ *For more details on application system environments, see Describing an Application System Environment with Hopex IT Architecture.*

In addition to a precise description of the application architecture to be implemented, this step covers the following points:

- Identify precisely the exchanges between the different software and hardware components, and formalize them through *service interfaces*.

  📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

  ☛ *For further details about service interfaces, see Describing information exchanges.*

- Verify that the application architecture covers the functional requirements identified in the business capability maps.

  ☛ *For more details on the functional analysis, see Analyzing the functional coverage of the architecture implemented.*

## Describing application systems

In a top-down approach, the main application system structure diagram is the entry point for the description of the existing or planned application system.

☞ *For more details on application systems, see Describing an Application System.*

```
The following diagram describes the application system
corresponding to purchasing requests processing.
```
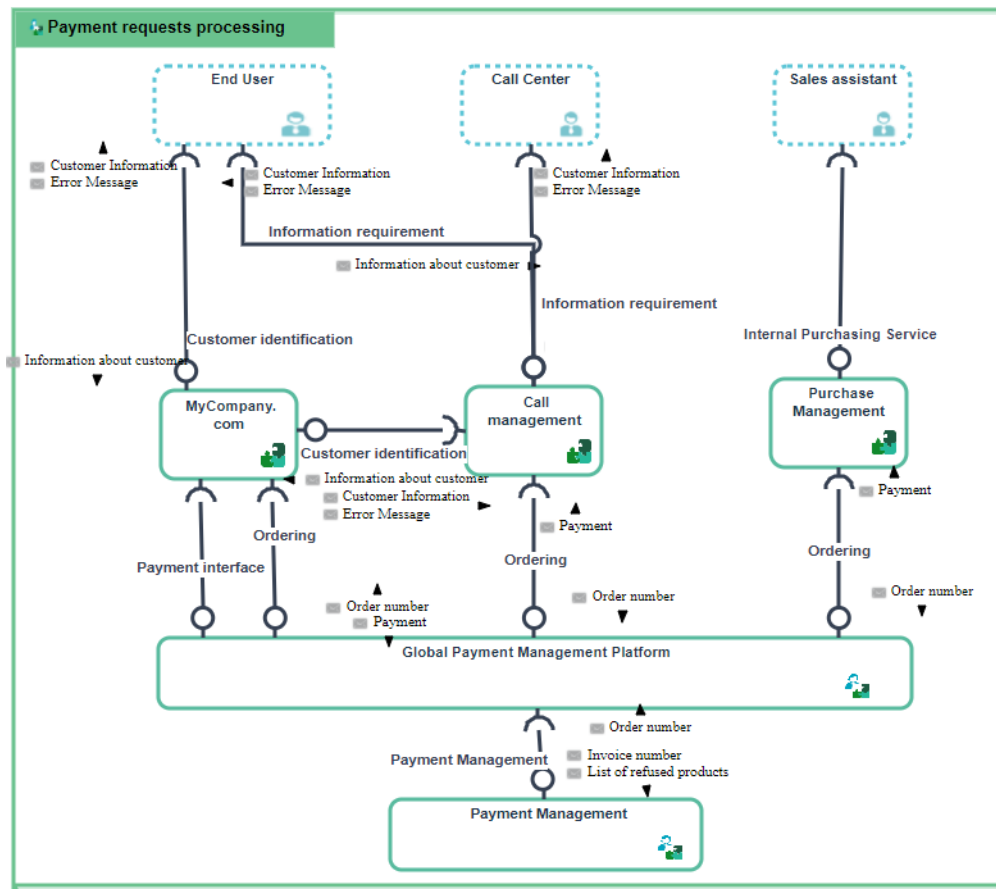


*The following diagram describes the application system corresponding to purchasing requests processing.*

```
Purchasing requests can be formulated by external customers
via an Internet purchasing application or indirectly via a
call center. Internal users have to call a "Sales assistant"
who uses the "Office Supplies Purchasing Management"
application.
```

The application subsystems can then be described hierarchically by showing at each level the points of exchange with the outside world.

The *data stores* are used to represent the data that will be stored in databases.

📖 *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.*

☛ *For more information on data stores, see  Managing Data.*

---

# Describing Applications

With **Hopex IT Architecture** an application is described by:

- The information flows it processes and transports, see Describing flow scenarios,
- The elements that provide the services associated with the functionalities it covers, see Describing the structure of an application and its services.

These complementary approaches help to draw up an exhaustive list of the components of an application (services and APIs) and the components of its environment interacting with it.

## Describing flow scenarios

**Hopex IT Architecture** offers flow scenario facilities to describe precisely data exchanged.

At each level of the application architecture, it is possible to define flow scenarios between system components in specific contexts.

The objective of flow scenarios is to verify that content is correctly conveyed between components.

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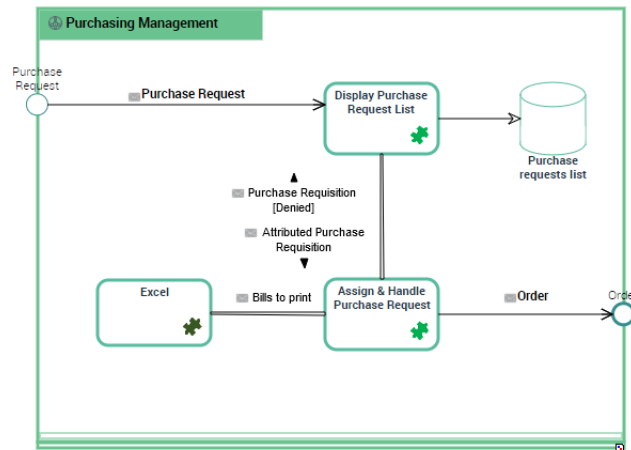


*Example of a Scenario of Application Flows for "Managing Purchase Orders".*

☛ *For more details on flow scenarios, see Creating a flow scenario sequence diagram.*

## Describing the structure of an application  and its services

Structure diagrams use service interactions to describe the data exchanged between the components.
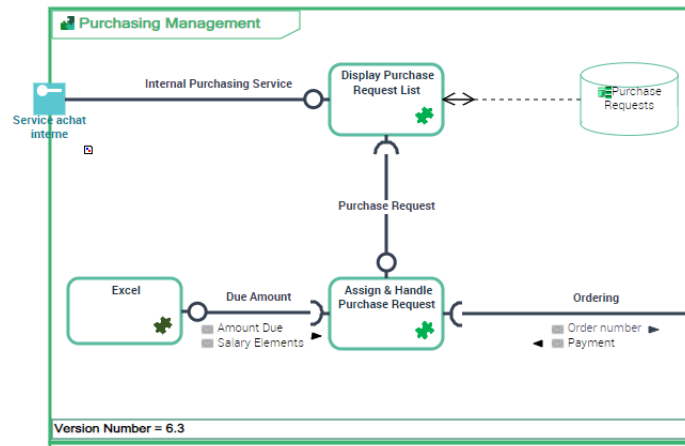


*"Purchasing Request Management" application structure diagram*

```
The "Purchase Request Management" application uses two IT
Services: "Display purchase request list" and "Assign and
handle purchase request". The IT Service "Assign and handle
DA" uses the Excel microservice.
```

# Defining the Deployment Architecture of an Application

An application deployment architecture allows to represent *Deployable Application Packages* and *Deployable Data Packages* as well as the *Technical Communication Lines* necessary for their exchanges.

📖 *A deployable application package is a split of application code according to technical criteria for hosting purpose. For example, it may be N tiers, Front End/Back End/… or GUI/Business Logic/Database etc... Each deployable application package is associated to required technologies (for running) and can host code for several IT services. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

📖 *A deployable data package represents a data part of an application deployment that must be hosted and accessed by application services (code) to run. Each deployable data package is associated to required technologies (for data hosting and access) and can host several data structures. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model). Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

📖 *A package connection represents a connection requirement between deployable packages. A client (package) requires a connection to a server (package). The connection is defined by a Connection Type and can be characterized with technical flow measures.*

Several viewpoints are proposed in **Hopex IT Architecture**:

- The *Application Deployment Environment* used to represent of the deployments of partner applications as well as microservices identified around the subject application, see Describing an Application Deployment Environment.
- The *Application System Deployment Architecture* used to represent the set of Application Deployment Architectures that must be coordinated to cover required dependencies between them, see Describing an Application System Deployment Architecture.
- The *Application Deployment Architecture* used to represent the deployment packages list and the module lines, see Describing an Application Deployment Architecture.
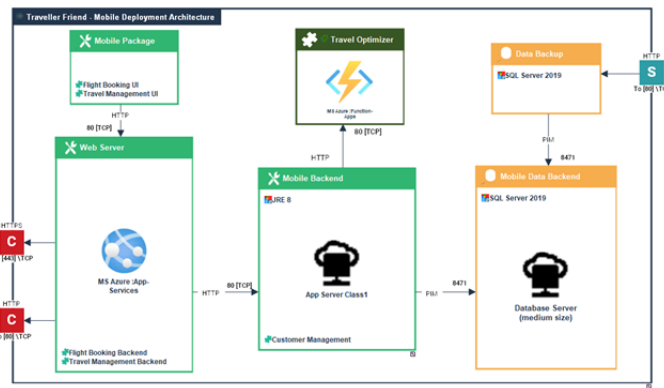


*Application Deployment Architecture Diagram*

To facilitate the creation of your application deployment architecture, **Hopex IT Architecture** provides standard deployment architecture diagrams.

☛ *For more details, see Deployment Architecture Templates.*

# Building the Logical Architecture

**Hopex IT Architecture** provides ways to define logical application architectures that represent ideal architectures. These representations make it possible to design logical structures for application architectures, to rationalize exchanges between these structures and to identify the data used. Logical application architectures can then be compared with the implemented architectures to detect gaps between the real and the ideal.

☛ *For more details on use of a logical application system, see Describing Logical Application Architecture.*

*Logical application systems* can be described using a top-down approach, starting with a description of the company's main application systems, or a unitary approach, describing only some logical application systems.

> 📖 *A logical application system is an assembly of other application architectures, logical applications and end users, interacting with application components to implement one or several functions.*

If you use a unitary approach, you must describe the *Application Logical System Environement* to provide a context to the use of the logical system and its service interactions with exeternal, logical components.

> 📖 *A logical application system environment presents a logical application system use context. It describes the service interactions between the logical application system and its external partners, which allows it to fulfill its mission and ensure the expected functionalities.*

At this level of the method, this step, which is not mandatory, covers the following points:

- Identify the exchanges between the logical components and formalize them through *service interfaces*.

  > 📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*
  >
  > ☛ *For further details about service interfaces, see Describing information exchanges.*

- Verify that the logical architecture covers the functional requirements identified in the business capability maps.

## Structure diagram of the logical application system

The *logical application system* components are described in a diagram featuring:

- the services offered or required;
- the processes handled, the components and their interactions service;
- the end users interacting with the application components.

```
The following diagram describes the structure of the
logical application system "Purchase request processing"
offered to customers.
```
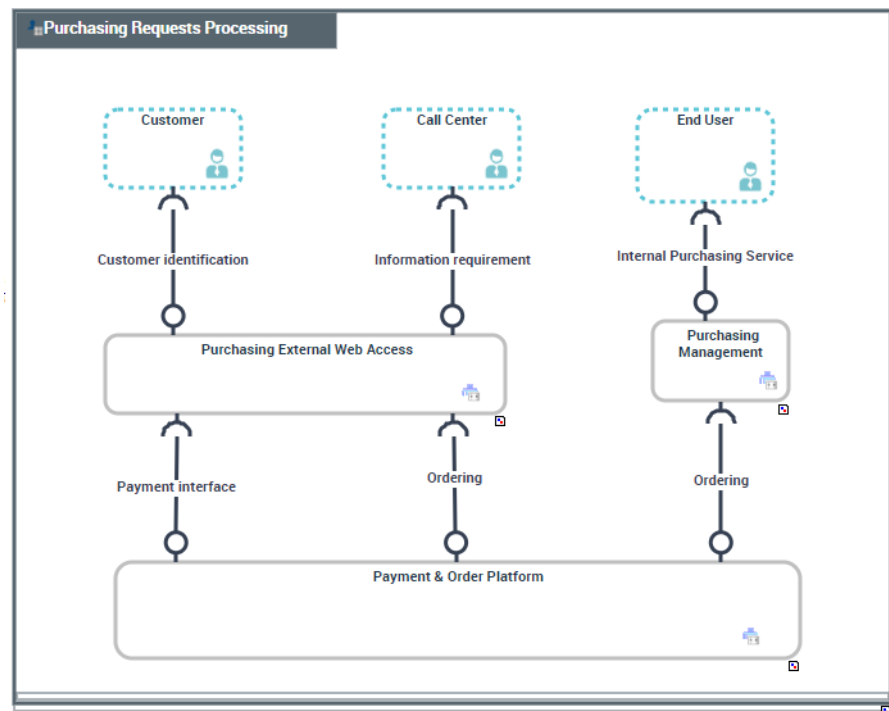


*" Purchase Request Processing" Logical application system structure diagram*

```
"Internet Purchase Requests" are offered to customers
either directly or through a "Call Center".
```

```
Requests made by customers are processed by a "Internet
Purchase Requests" logical application system.
```

```
The logical application system structure diagram, for
managing "Purchasing Requests", presents different logical
applications, access to a logical database as well as
```

```
service and request points for "Internal request service"
or "Order".
```
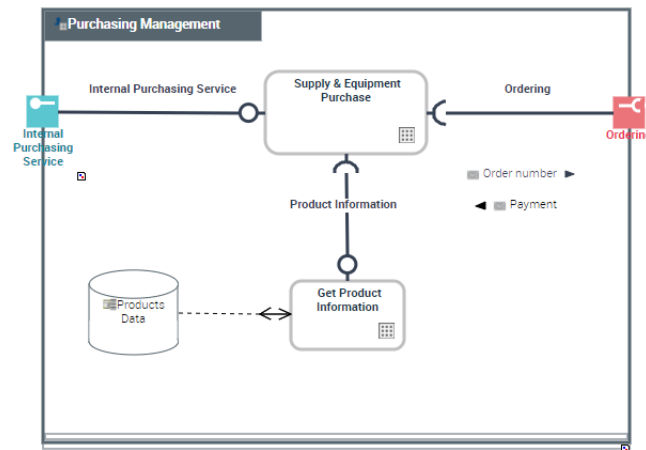


*" Purchasing request Management" Logical application system structure diagram*

☛ *For more details on use of a logical application system, see*
*Describing a Logical Application System with Hopex IT Architecture.*

## Logical application system environment diagram

📖 *A logical application system environment presents a logical application system use context. It describes the service interactions between the logical application system and its external partners, which allows it to fulfill its mission and ensure the expected functionalities.*

The components of a *logical application system environment* are presented in an application system environment diagram that describes the internal logical application systems as well as the partner logical application systems.

s
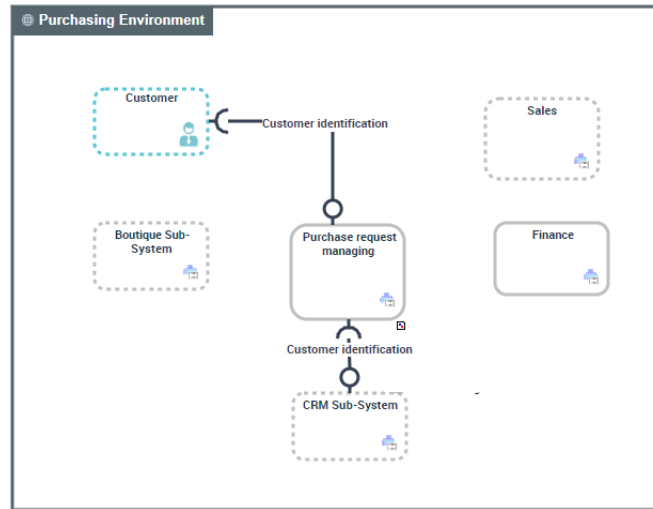


*Logical application system environment diagram*

```
Purchasing requests are formulated by users in conditions
specified by Sales service and also the Finance service
which are external to the described environment.
```

> ☛ *For more details on use of a logical application environment system, see Logical Application System Environment Description.*

---

# Analyzing the functional coverage of the architecture implemented

The goal of this step, on a strategic level, is to check the suitability between the business capabilities of the enterprise, the functionalities required and the applications that deliver them.

## Describing Business Capabilities

> ☛ *For more details on managing business capabilities with **Hopex IT Architecture**, see Describing Business Capabilities with Hopex IT Architecture.*

A *business capability* defines an expected skill.

> 📖 *A business capability represents a specific ability that an organization possesses or needs to develop to deliver a particular business outcome.*

A *business capability map* describes what the enterprise is capable of producing for its internal needs or for meeting the needs of its clients. It is thus based on the main business capabilities of its activity at a given moment.

📖 *A business capability map is a set of business capabilities with their dependencies which define a framework for an enterprise stage.*

```
For example, the standard ability to manage "Operational
Activities" is based on the business capabilities to
process "Supply", "Sales" and "Complaints", "Order
Management" and "Customer Management".
```
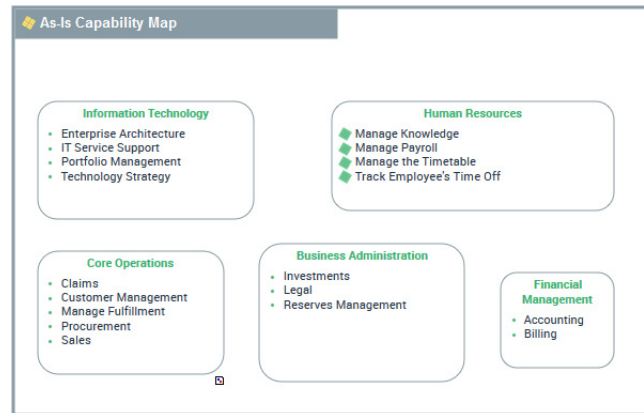


*Example of a business capability map*

☛ *For more details on managing a business capability map, see* *Describing Business Capabilities with Hopex IT Architecture.*

## Identifying the technological capabilities associated to business capabilities

The aim here is to connect *technological capabilities* – corresponding to what is expected to achieve a goal – to the actual means of production, i.e. *Applications* or *Application Systems* on a conceptual level.

📖 *An application system is an assembly of other application systems, applications and end users interacting with application components to implement one or several functions.*

By constructing the *functionality map* on the one hand and the *application system environment* on the other hand, you can check that the functionalities are implemented by application components.

☛ *For more details on the logical applications associated with business capabilities, see* *Describing the fulfillment of a Functionality.*

**Hopex IT Architecture** provides a report that presents the result of the implementation of business capabilities by *applications* or *application systems* physical or logical.

☛ *For more details on implementation of business capabilities, see* *Business Capability Breakdown Report.*

A *technological capability* defines the expected capacity of an equipment.

📖 *A technology capability is the ability to deliver a technology service which is required by a technology artifact or an application.*

A *functionality map* describes all the functionalities the enterprise is able to cover for its internal needs or for meeting the needs of its clients.

📖 *A technology capability map is a set of technology capabilities and their dependencies that, together, defines the scope of a hardware or software architecture.*
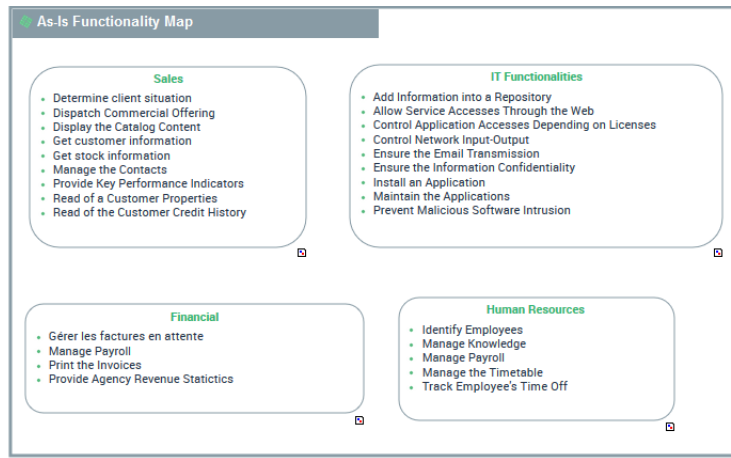


*Example of a functionality map*

☞ *For more details on managing a functionality map, see Describing a Functionality Map with Hopex IT Architecture.*

The description of *business capabilities* and *functionalities* is particularly interesting if business capabilities are associated with the functionalities that fulfill them.

☞ *For more details on this  analysis, see Describing Fulfillment of a Business Capability.*

### Identifying the   applications associated with functionalities

*Applications* cover *functionalities* associated with *business capabilities*. In **Hopex IT Architecture**, a report allows to check the functional coverage of your applications.

☞ *For more details on this functional coverage, see Describing Fulfillment of a Business Capability.*

# Defining the technical infrastructure

Describing the technical infrastructure helps to design deployable application and data packages to prepare their deployment.

Technical infrastructure elements are identified and characterized by technologies and hosted IT services.

With **Hopex IT Architecture** the infrastructure can be described in a bottom-up approach, from the most detailed to the most conceptual, or top-down, from the most conceptual to the most detailed. Presentation of these functionalities is based on the example of a call center.

## Resource Architecture Environment Diagram

*A business architecture environment represents the relationships of a business functional area with its partners.*

The following diagram describes the environment of a support center.
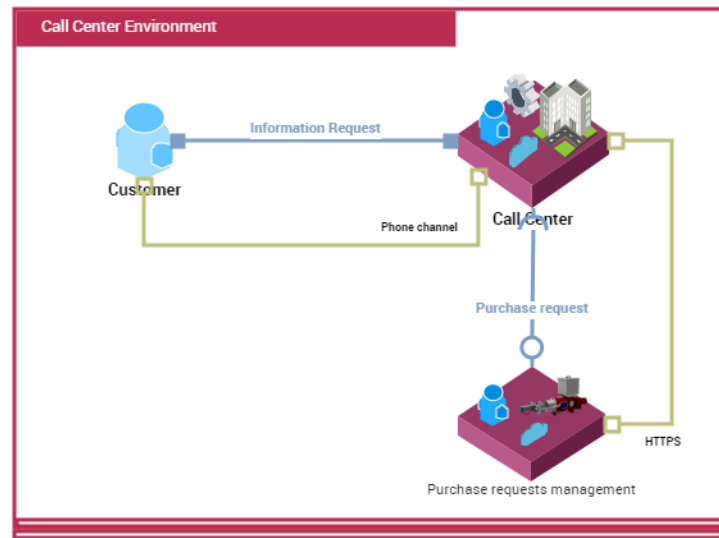


The call center responds to customer requests. It is based on an external service to fulfill any purchasing requests.

☛ *For more details, see* *Describing IT Infrastructures.*

## Describing Resource Architectures

The Resource Architecture Assembly Diagram describes the hardware and organizational resources required for handling service requests.

☛ *For more details, see* *Describing Resource Architectures.*

In the call center example, we consider only the operator and the IT infrastructure that represents its equipment.
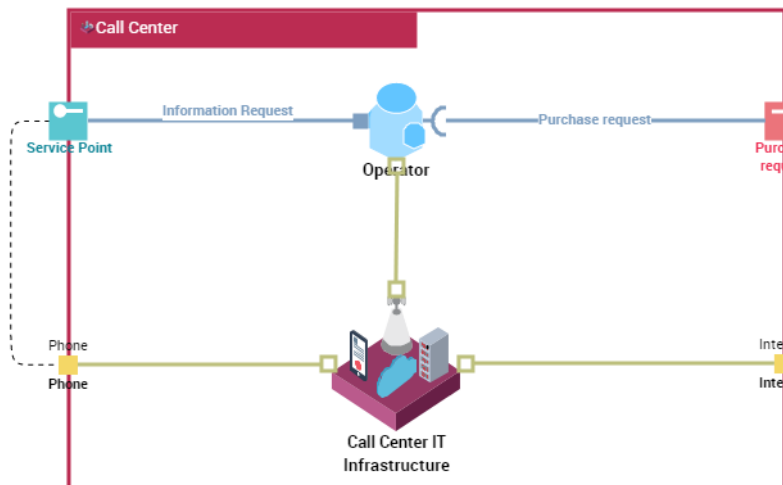


A team of operators handles all requests, whatever their nature, by telephone or by e-mail.

The operator identifies the caller, records the request, applies a first filter (in case of error) and if necessary records a purchasing request via request points.

This diagram contains a *Request Point* from which the operators make purchasing requests.

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

## IT infrastructure assembly structure diagram

This diagram presents an IT infrastructure. It contains Infrastructure IT component such as: computers or IT equipments.

☛ *For more details, see Describing IT Infrastructures.*
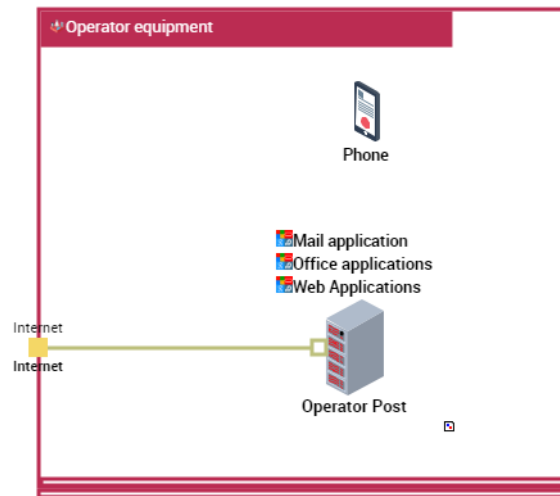
The basic hardware architecture of a call center includes
two link points to the outside: a telephone link, a link to
a private network that enables the HTTP link for the
purchasing request.

Note that the used *communication protocols* are mentioned on the network
channels.

> 📖 *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.*

## Computing Device Assembly Diagram

The computing device assembly diagram presented below describes the software technologies, the deployable application packages and the IT devices installed on a standard PC.

☞ *For more details, see* *Describing the Computing Devices.*
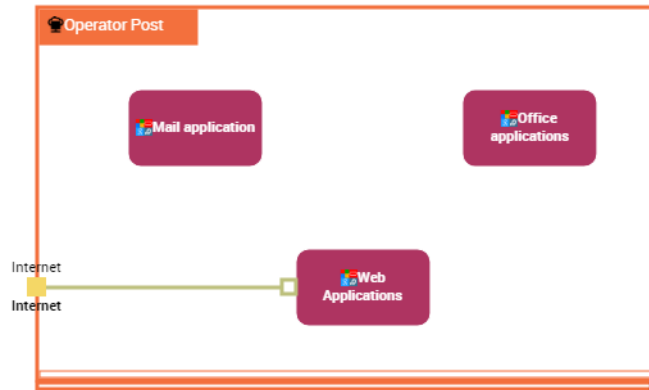


*Computing device assembly diagram of a standard PC.*

```
A standard PC is equipped with office system applications
and electronic mail applications.

A standard PC also has an HTTP connection to access Web
applications to manage purchase requests.
```

# Designing applications

**Hopex IT Architecture** offers the tools to assist architects in specifying updates to their IT system.

☞ *To access this UML feature, you must sign-in as an* **IT Application Designer** *or a* **Solution Architecture Functional Administrator***.*

## Using UML formalism

**Hopex IT Architecture** provides the tools required to model logical data via class diagrams and data models.

☞ *For more details on UML main concepts, see* *UML modeling of data.*

## Describing batch processing

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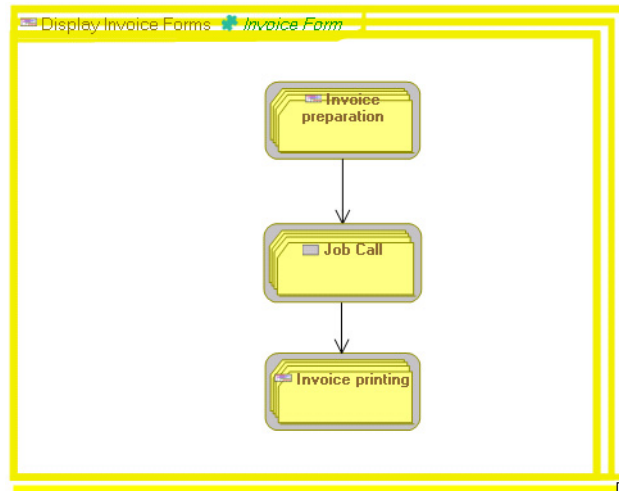


☛ *For more details on describing batch processing, see* *Describing Batch Processing.*

## 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.
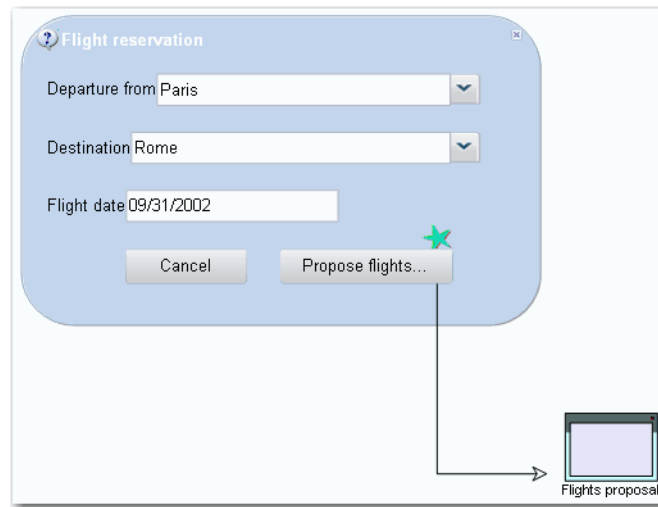


☛ *For more details on describing interfaces, see Defining User Interfaces.*

## Describing application processes

**Hopex IT Architecture** offers the possibility to check that the processes performed by the application system are correctly covered by describing *Application processes*.

📖 *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 system processes, see Describing System Processes.*
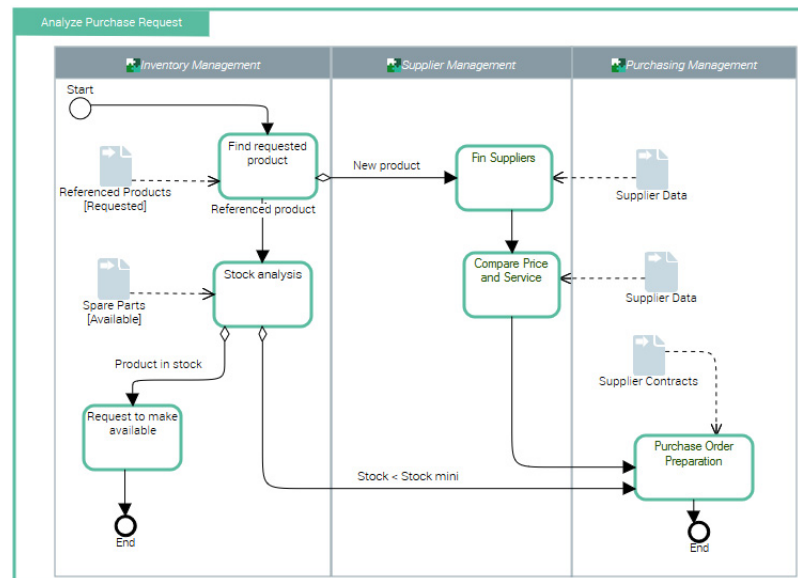
*Application process diagram*

# Managing service catalogs

📖 *A service catalog contains a list of key service offers for which solutions are recommended.*

A *service catalog* describes the list of services provided to an *Enterprise*. These services are defined by their functional coverage (the *capacity* they provide), the solutions that can be implemented to deliver the service (the available agents) and the status of each agent within the catalog (this solution is norm, accepted, tolerated, prohibited, etc.).

Defining a service catalog requires defining the functional scope covered via a reference capability map.

Capabilities (or functionalities) associated to services are implemented by one or several technical elements that are considered as solutions. The solution status helps the user to select the better solution for his context.



*Presentation of a Business service catalog*

```
This presentation of a business service catalog enables the
identification of the business capabilities associated with
the services, the technical solutions that implement the
capabilities, and the status of these solutions in the
context of the catalog's use
```

**Hopex IT Architecture** offers the following service catalogs:

- *business service catalogs*,

     📖 *A business service catalog provides a centralized information source for the business services offered by the service provider organization. It contains a customer-oriented view of the services associated to business capabilities, how they are supposed to be used, the processes that they support as well as the expected service quality level. The business service catalog presents the list of functionalities mentioned as well as implementation recommendations.*

- *Cloud services catalogs*,

     📖 *An information service catalog provides a centralized information source for the information services offered by the service provider organization. It contains a customer-oriented view of the information services used, how they are supposed to be used, the processes that they support as well as the expected service quality level. The information service catalog presents the list of functionalities mentioned as well as implementation recommendations.*

- *Technology service catalogs*,

     📖 *A technology service catalog provides a centralized information source for the technology services offered by the service provider organization. It contains a customer-facing view of the technology services in use, how they are intended to be used, the process they enable, and the levels and quality of service the customer can expect from each service. The technology service catalog provides the list of*

> reference technology capabilities and their recommended implementation.

- *information service catalogs*,

  📖 *An information service catalog provides a centralized information source for the information services offered by the service provider organization. It contains a customer-oriented view of the information services used, how they are supposed to be used, the processes that they support as well as the expected service quality level. The information service catalog presents the list of functionalities mentioned as well as implementation recommendations.*

- *hardware service catalogs*.

  📖 *A hardware service catalog provides a centralized information source for the hardware services offered by the service provider organization. It contains a customer-oriented view of the hardware used, how they are supposed to be used, the processes that they support as well as the expected service quality level. The hardware service catalog presents the list of hardware functionalities mentioned as well as implementation recommendations.*

  ☛ *For more details on the use of service catalogs, see Using service catalogs.*

# HOPEX IT ARCHITECTURE DESKTOP PRESENTATION

☛ *Hopex IT Architecture is mainly intended for web users. Desktops described in this guide are accessible only to Web desktop users.*

## Connecting to the solution

To connect to **Hopex IT Architecture**, see **Hopex Common Features**, "Hopex Desktop".

## Hopex IT Architecture Desktop Presentation

The menus and commands available in **Hopex IT Architecture** depend on the product licenses that you have and on the profile with which you are connected.

☛ *For more details on the use of the HOPEX interface, see the chapter "Interface Presentation" of the **Hopex Common Features** guide.*

All users signed-in to the **Hopex IT Architecture** Solution dispose of the same desktop containing several navigation menus.

## Presentation of the Solution Architect workspace

Users connected to the **Solution Architect** profile can use several navigation menus giving access to all the features provided in the **Hopex IT Architecture** Solution.
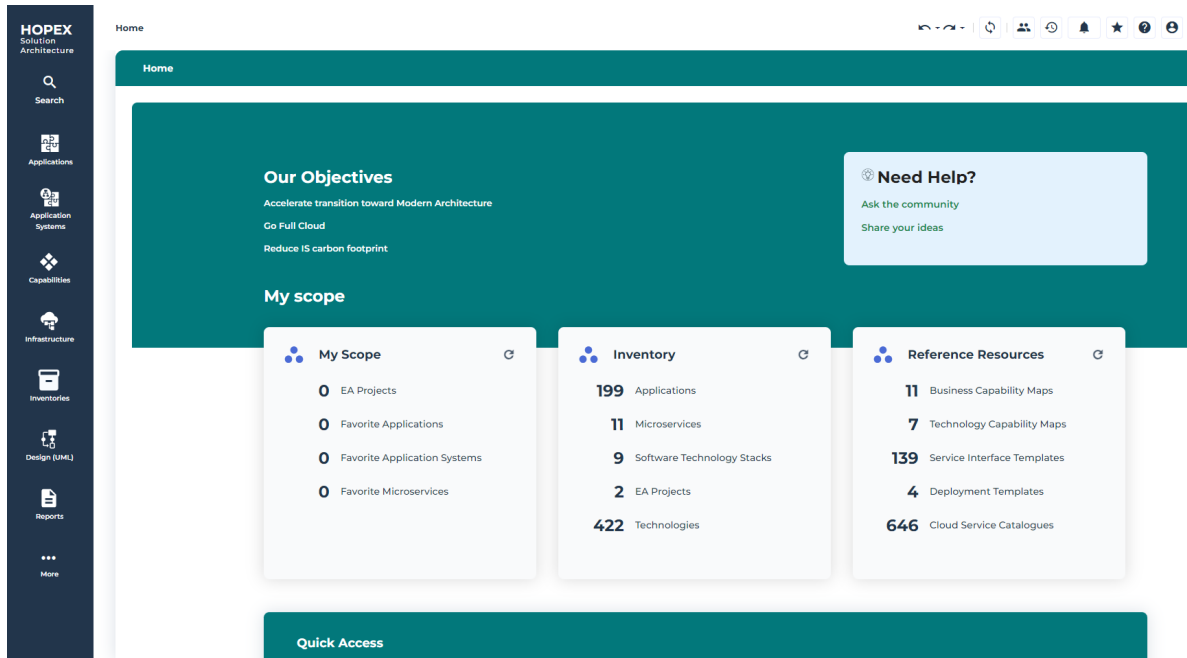


### *The Applications menu*

The **Applications** menu gives access to all the repository applications.

☛ *For more information about the description of applications, see Describing an Application with Hopex IT Architecture.*

### *The Application Systems menu*

The **Application Systems** menu gives access to all the repository application systems.

☛ *For more information about the description of applications, see Describing System architecture.*

### The Capabilities menu

The **Capabilities** menu gives access to the following subjects.

- **Business capability**, to describe business capabilities and business capability maps.

  ☞ *For more details on logical architectures, see Describing Business Capabilities with Hopex IT Architecture.*

- **Functionalities**, to describe the  functionality maps of the information system.

  ☞ *For more details on functionalities, see Using Functionalities with Hopex IT Architecture.*

### The Infrastructure menu

The **Infrastructure** menu gives access to subjects related to infrastructures.

☞ *For more details on infrastructures, see Modeling IT Infrastructures.*

- **IT infrastructure**, to describe the IT infrastructure elements.

  ☞ *For more details on infrastructures elements, see Describing IT Infrastructures.*

- **Resource architecture**, to describe the elements constituting a resource architecture.

  ☞ *For more details on resource architectures, see Describing Resource Architectures.*

- **Resource configuration**, to describe the server and network technical elements.

  ☞ *For more details on resource configuration, see Describing a resource configuration.*

### The Inventories menu

The **Inventories** menu gives access to the following subjects.

- **Software** theme, giving access to the following elements:
  - **Application services**,

    📖 *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.*

    ☞ *For more details on applications services, see Describing an IT Service with Hopex IT Architecture.*

  - **Microservices**,

    📖 *A microservice 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 microservices, see Describing a microservice with Hopex IT Architecture.*

  - **System processes**,

    📖 *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 system processes, see Describing System Processes.*

- **Business Service Catalogs** to describe the services provided by the businesses to the users.

  ☛ *For more details on service catalogs, see Using service catalogs.*

- **Communication systems**

  📖 *A communication system helps to identify and describe the main integration processes using several Software Communication Chains as well as communication services.*

  ☛ *For more details on communication systems, see Using communication systems.*

- **Logical software architecture**, to describe the elements contained in the information system logical architecture.

  ☛ *For more details on logical architecture, see Describing Logical Application Architecture.*

- **Deployment** to describe the elements linked to the deployment of an information system.

  - **Technology Capabilities**

    📖 *A technology capability is the ability to deliver a technology service which is required by a technology artifact or an application.*

    ☛ *For more information on the use of technology capabilities, see Describing a technology capability.*

  - **Software Technologies** to describe the technical elements of the information system.

    ☛ *For more details on software technologies, see Describing Software Technologies.*

  - **Cloud service catalogues**

    ☛ *For more details on Cloud service catalogues, see Using Cloud Services.*

  - **Technology service catalogues**

    ☛ *For more details on technology services catalogs, see Using Cloud Services.*

- **Infrastructure** theme, giving access to the following themes:

  - **Facilities**,

    📖 *A facility is a model of site of interest for the enterprise. Examples: Data Center, Factory or Outlet*

    ☛ *For more details on facilities, see Describing a Facility.*

  - **Computing Networks**,

    📖 *An IT network is set of IT equipment components (e.g.: routers, switches, firewalls) that allow remote communications between*

computing devices (e.g.: IT server). An IT network can be broken down into sub-networks.

☛ *for more details on computing networks, see Describing an IT network.*

- **Computing Devices**

  📖 *An IT network is set of IT equipment components (e.g.: routers, switches, firewalls) that allow remote communications between computing devices (e.g.: IT server). An IT network can be broken down into sub-networks.*

  ☛ *For more details on computing devices, see Describing a Computing Device.*

- **Network Devices**,

  📖 *An IT network is set of IT equipment components (e.g.: routers, switches, firewalls) that allow remote communications between computing devices (e.g.: IT server). An IT network can be broken down into sub-networks.*

  ☛ *for more details on computing networks, see Describing a Computer Network Device.*

- **Data**, to describe the business data.

  ☛ *For more details on business data management, see Managing Data.*

  - **Business dictionaries**

    📖 *A business dictionary collects and structures a set of concepts that expresses the knowledge of a particular area.*

    ☛ *For more details on business dictionaries, see the **Hopex Data Governance** guide.*

  - **Data Dictionaries**

    📖 *A data dictionary describe all the elements defining your logical data architecture.*

    ☛ *For more details on data dictionaries, see the **Hopex Data Governance** guide.*

  - **Information Service Catalogs** to describe the user services offered by the information system.

    ☛ *For more details on service catalogs, see Using service catalogs.*

- **Hardware** to describe all hardware elements of the information system.
  - **Hardware Capabilities**, to access the functionalities related to hardware elements.

    📖 *A hardware capability is the ability to deliver a physical outcome which is required by an organizational resource in order to perform its work. This hardware capability is generally necessary within a computing process in order to execute a specific operation.*

    ☛ *For more information on the use of technology capabilities, see Describing a hardware capability.*

  - **Hardware**,

    📖 *Non-IT Hardware can embed computers. Together with their embedded computers, they provide information and IS services. Examples: Connected Truck with Delivery Calendar Application and connected Drone with Online Payment Application. Hardware device can also provide hardware functionalities. Example: Connected fridge*

*providing ordering functionalities and of course a freezing hardware functionality and connected drones fly and provide Online Payment.*

☛ *for more details on computing networks, see* Describing an Hardware*.*

- **Hardware services Catalogs**.

☛ *For more details on hardware services catalogs, see* Using Cloud Services*.*

- **All Sketches**, to access all the skectches of your repository.

☛ *For more details on the use of sketches with* **Hopex IT Architecture***, see* Creating a Sketching diagram with Hopex IT Architecture*.*

## *The Design (UML) menu*

The **Design (UML)** menu gives access to the following submenus:

- **OO Implementation (UML)**, to design you IS using the UML formalism.

☛ *For more details on the use of UML concepts, see* About UML implementation*.*

Depending on the options selected, two submenus are also available:

☛ *To see these submenus, open the* **Options** *window and check that* **IT Architecture > User Interface and Batch Features (ADES)** *is selected.*

- Submenu **Batch and Program Implementation**,

☛ *For more details, see* Describing Batch Processing*.*

- Submenu **User Interfaces**.

☛ *For more details on describing user interfaces, see the* Defining User Interfaces*.*

## *The Reports menu*

The **Reports** menu gives access to all the reports contained in each Solution.

☛

☛ *For more details on the use of these reports, see "Generating Reports" chapter in* **Hopex Common Features** *guide.*

☛ *For more information on* **Hopex IT Architecture** *reports, see* HOPEX IT Architecture Reports*.*

### The Governance menu

**Governance** gives access to the following submenus.

- **Policy framework**, to access the frameworks that define the company policy.

  ☛ *For more details on Policy Frameworks, see Define a Policy Framework with Hopex IT Architecture*

- **EA Projects**, to access the project management features.

  ☛ *For more details on project management, see "The Enterprise Architecture (EA) Projects in Hopex" in the **Hopex Common Features** guide.*

- **Action Plans**, to describe and manage the action plans linked to the transformation of the information system.

  ☛ *For more details on managing action plans, see "Using Action plans" in **Hopex Common Features** guide.*

### The Environment menu

The **Environment** menu gives access to the following submenus:

- **Containers**, to access the Libraries and Enterprises management features.

  ☛ *For more details on **Containers** and **Organization**, see Defining the Work Environment.*

- **Organization**, to access the main objects handled in the **Hopex IT Architecture** Solution.

  - **Business Lines**,

    📖 *A business line is a high level classification of main enterprise activities. It corresponds for example to major product segments or to distribution channels. It enables classification of enterprise processes, organizational units or applications that serve a specific product and/or specific market.*

  - **Process Categories**,

    📖 *A process category defines a group of processes. It is linked to a Process Map or higher level Process Category. It regroups several processes and/or other categorized elements (e.g. Value Streams, Applications). It serves as an intermediate categorization level in the process hierarchy, so as to provide a guided and progressive access to finer grained processes.*

  - **Processes**

    📖 *A process is a set of operations performed by org-units within a company or organization, to produce a result. It is depicted as a sequence of operations, controlled by events and conditions. In the BPMN notation, the process represents a sub-process from the organizational point of view.*

  - **Sites**

    📖 *A site is a geographical location of an enterprise. Examples: Boston subsidiary, Seattle plant, and more generally the headquarters, subsidiaries, plants, warehouses, etc.*

  - **Org-Units**.

    📖 *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level*

*depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

☛ *For more details on the use of **Org-units**, see Using Org-units.*

- **My RFC's**
- **Ideas**, to access the features of library and environment management. If you own the associated license, **Ideas** provides access to idea management facilities.

    ☛ *For more details ideas management, see «Submitting and evaluating ideas» chapter in the **Hopex Common Features** guide .*

    ☛

    ☛

- **Common**, to access to following objects:
  - **Tags**

      📖 *A tag is a classifying description used to characterize objects.*

      ☛ *For more details on the use of tags, see Platform - Common Features > Collaboration Tools > Communicating in HOPEX.*

  - **Report DataSets**

      📖 *A Report DataSet is a set of data extracted from the HOPEX repository and used as a data source in reports.*

      ☛ *For more information, see Platform - Common Features > Documentation > Generating Documentation > Managing Report DataSets.*

  - **All Sketches**, to access all the skectches of your repository.

      📖 *A sketching diagram is a drawing that enables you to exchange with your coworkers without an issue of methodology or formalism.*

      ☛ *For more details on the use of sketches with **Hopex IT Architecture**, see Creating a Sketching diagram with Hopex IT Architecture.*

### Administration Menu

The **Administration** menu gives access to the following submenus:

- **Templates** giving access to templates associated with the following components:
  - **Service Interfaces**, see Using a Service Interface Template.
  - **Service Operations**, see Using a Service Interface Template.
  - **Contents**, see Using a Service Interface Template
  - **Deployment Architecture**, see Deployment Architecture Templates.
- **Categorization Schemas**, see Defining Data Categories.
- **Methodological Domains**, see Defining Methodological Domains.

### Presenting the Solution Architecture Functional Administrator workspace menus

The activities offered only to users connected with the **Solution Architecture Functional Administrator** profile are:

- Administration, via the **Administration** menu,
- the creation of objects from the **Environement** menu, see The Environment menu

### Presentation of the Application Designer workspace

The menus offered when using the **Application Designer** profile are similar to the ones of the **Solution Architect** profile.

However, the users signed-in as an **Application Designer** can modify object properties but can not modify diagrams.

### Presenting the Application Viewer workspace

The navigations menus available for the users signed-in as **Application Viewer** are identical to the menus available in the **Solution Architect** profile.

However, users signed-in as **Application Viewer** can only access objects in read-only mode.

---

## Switching between Profiles

Using the **Hopex IT Architecture** desktop, you can access to any Hopex solution desktop, without logging out, just by switching to another profile.

For example, you can switch to a specific profile:

1. Select **Main Menu > Switch Profile**.
2. Select the profile with which you want to connect.
3. (If you made modifications in your private workspace) Click:
   - **Yes**, to save your modifications in the repository.
   - **No**, if you do not want to save in the repository the modifications you made since your last dispatch. Modifications to your desktop are also lost.

   The desktop associated with the selected profile is displayed.

   ☛ *Click **Cancel** to stay in your private workspace.*

# BEFORE STARTING WITH HOPEX IT ARCHITECTURE

## Defining the Work Environment

In the context of the **Hopex IT Architecture** solution, a library can hold all the elements of your project: processes and org-units, for example.

    📖 *Libraries are collections of objects used to split repository content into several independent parts. They allow creation of virtual partitions of the repository. In particular, two objects owned by different libraries can have the same name.*

Using *Enterprise* enables preparation of a transformation project.

    📖 *An Enterprise is a purposeful undertaking, conducted by one or more organizations, aiming at delivering goods and services, in accordance with the enterprise mission in its changing environment. During its development over time, an enterprise has to adapt to its environment and sets up transformation goals and objectives along with course of action to achieve these objectives. The design and realization of the resulting transformation stages may transcend organizational boundaries and consequently require an integrated team working under the direction of a governing body to involve stakeholders in transformation initiatives. This requires the implementation of an integrated team, under the responsibility of a governing body, to involve the stakeholders in the transformation.*

    ☛ *For more details on managing containers, see the "Enterprises and Libraries" chapter in the* **Hopex Common Features** *guide.*

### Accessing the list of libraries with Hopex IT Architecture

To access the list of libraries from the **Environment** navigation menu:

- ☐ Select **Containers > Libraries**.
  The library tree appears.

### Accessing the list of enterprises with Hopex IT Architecture

To access the list of enterprises from the **Environment** navigation menu:

- ☐ Select **Containers > Enterprises**.
  The list of enterprises is displayed.

In the context of **Hopex IT Architecture** solution, Enterprise characteristics are simplified as related to **Hopex IT Business Management** solution.

    ☛ *For more details on using Enterprise in a transformation context, see "The strategic elements of a transformation phase" chapter of* **Hopex IT Business Management** *guide.*

# Using Org-units

*Org-Units* are used in several diagrams. The main points concerning this object type are reminded in this section.

> 📖 *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

## Creating an org-unit

To create an org-unit from the **Environement** navigation menu:

1.  Select **Organizations > Org-Units**.
    The list of org-units displays in alphabetical order.
2.  Select the **Org-unit** file and click **New**.
3.  In the wizard window, select **Org-units**.
4.  In the **Creation of an Org-Unit** window, enter the name of the org-unit you want to create.
5.  Click **OK**.
    The org-unit appears in the list.

## Internal org-unit/external entity

During creation, org-units are considered as elements internal to the company.

To specify that an org-unit is not part of the company, you must modify the org-unit properties and enter the "External" status.

To assign the "External" characteristic to the org-unit:

1.  Right-click the org-unit and select **Properties**.
2.  Open the **Characteristics** property page of the org-unit.
3.  In the **Identification** section, select the **External org-unit** field.
    This characteristic is represented graphically and is automatically displayed in the diagrams.

# Using IT architecture diagrams

With **Hopex IT Architecture**, an application object may be described by different diagrams. Each diagram type corresponds to a specific view of the object: internal architecture, deployment architecture, flows exchanged inside the object and flows

exchanged outside depending on the context of use of the object. Depending on the described object, each representation is associated to a diagram type.

- **External data flows** are represented in an *Environment Diagram*. This diagram type contains the described application object and the application flows exchanged with partners (other application systems, applications, data stores, org-units or position type).

- The **internal architecture** is described by a structure diagram representing the object components their exchanges. A structure diagram can be designed for an application, an application environment, an application system, an application system environment, a logical application system, a logical application, an application service or a microservice.

- The **deployment architecture** is described by a diagram representing the *deployable application packages*, the *microservices* and the *deployable data package* used as well as the required communication techniques.

- The **Internal Data Flows** are represented in a *scenario of flow diagram* describing the messages exchanged between the object components. With **Hopex IT Architecture**, you can design two types of *scenario of flow diagram*:

  - The *Scenario of flows diagrams* that describe the flows exchanged in different use scenarios of the object described.
  - *Scenario of sequence diagrams* that describe the chronology of the flows exchanged in different use scenarios of the object described.

    ☛ *To use Scenario of Application Flows Diagrams, open the* **Options** *window and check that* **IT Architecture > Activate Flow Scenario Sequence Diagrams** *option is activated.*

- The **external interactions** are represented in a *Scenario of Flow Diagram* describing the external service interactions of an application object in a specific environment. This diagram contains the described object and the service interactions with partners (other systems).

| Representation | List of involved diagrams |
|---|---|
| Internal Architecture | <ul><li>Application structure diagram</li><li>Architecture</li><li>Application System Structure Diagram</li><li>IT Service Structure Diagram</li><li>Microservice structure diagram</li><li>Application System Structure Diagram</li><li>Logical application structure diagram</li><li>Structure diagram of the logical application system</li><li>Deployable Application Package Diagram</li><li>Resource Architecture Assembly Diagram</li><li>Hardware Assembly Diagram</li><li>IoT Device Assembly Diagram</li></ul> |
| Internal Data Flows | <ul><li>Scenario  of Application Flow Diagram</li><li>Scenario of Application System Flows Diagram</li><li>Scenario of IT Service Flow Diagram</li><li>Scenario of Microsystem Flows Diagram</li><li>Scenario of Logical Application System Flows Diagram</li><li>Scenario of Application Flows Sequence Diagram (UML)</li><li>Scenario of Application System Flows Sequence Diagram (UML)</li></ul> |
| External Data Flows | <ul><li>Scenario of Application Environment Flows Diagram</li><li>Scenario of Application System Environment Flows Diagram</li><li>Scenario of Logical Application System Environment Flows Diagram</li><li>Scenario of Application Environment Flows Sequence Diagram (UML)</li><li>Scenario of Application System Flows Sequence Diagram (UML)</li></ul> |
| External Interactions | <ul><li>Application Environment Diagram</li><li>Application system environment diagram</li><li>Logical application system environment diagram</li><li>Resource Architecture Environment Diagram</li></ul> |
| Deployment Architecture | <ul><li>Application Deployment Architecture Diagram</li><li>Application System Deployment Architecture Diagram</li><li>Microservice Deployment Architecture Diagram</li></ul> |

## Creating a structure diagram

To create a, *Application system structure diagram*, for example:

1. Open the **Diagrams** property page of the application system and click **Create a diagram**.

**2.** In the choice window, select **Structured diagram > Internal Architecture**.
The diagram opens in the edit area. You are now in the **Hopex** graphic editor. The frame of the described object appears in the diagram.

***Example of an application system structured diagram***
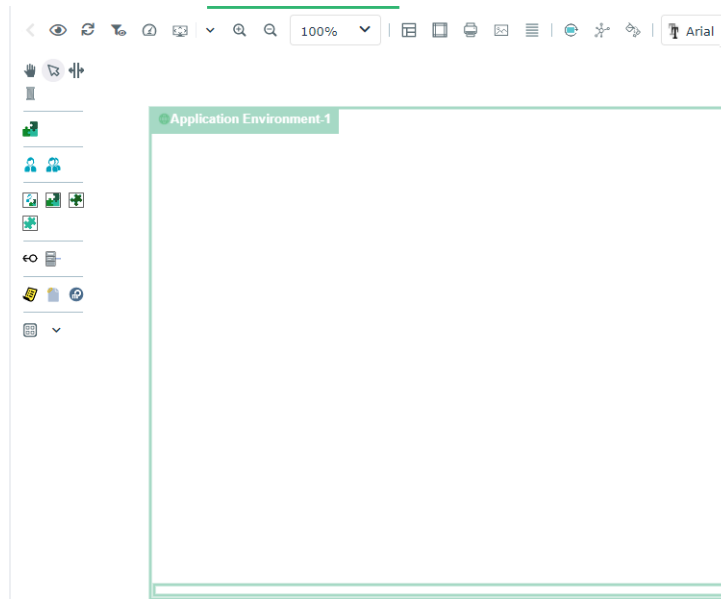


By default, the diagram is initialized with the described object, represented by a frame; the components of the described object are positioned at the top of the diagram.

If the described object is represented in a higher level diagram, the new diagram is initialized taking into account participants and flows that are represented in the higher level diagram.

## Diagram commands with Hopex IT Architecture

Depending on their type, **Hopex IT Architecture** diagrams propose specific commands.

| Icon | Description |
|------|-------------|
|  | **Refresh channels**<br>Allows to update the content of the channels described in a scenario of flows. See Creating an application flow channel. |
|  | **Reinitialize components**<br>Add, in the diagram, the components of the first level of the described object. |
|  | **Auto Layout**<br>Enables to organize automatically the described object components in the diagram. See Auto Layout in architecture diagrams. |
|  | **Add Items**<br>Enables to complete the current diagram with the elements defined in other diagrams. See Environment diagram initialization.<br>Available only for application environment and application system environment diagrams. |

## Auto Layout in architecture diagrams

If the environment contains components and interactions between components, each new diagram is initialized with these components displayed at the top left of the frame of the described environment.

The **Auto Layout** button allows you to reorganize the diagram elements taking into account the exchanged flows.

The **Diagram compression/dilatation coefficient** enables the specification of the distance to be expected between elements.

When you use the **Auto Layout** function, the previous presentation of your diagram is lost.

The auto layout facility is proposed for the following diagrams:

- Application Environment
  - Application Environment Diagram
  - Scenario of Application Environment Flows Diagram
- Application
  - Scenario of Application Flows Diagram
  - Application Structure Diagram
  - Application Deployment Environment Diagram
- Application System Environment
  - Scenario of Application System Environment Flows
  - Application System Environment Diagram
- Application System
  - Scenario of Application System Flows
  - Application System Structure Diagram
- IT Service
  - Scenario of IT Service Flow Diagram
  - IT Service Structure Diagram
- Logical Application
  - Scenario of Logical Application Flows Diagram
  - Logical application structure diagramme
  - Logical Application Deployment Environment Diagram
- Logical Application System Environment
  - Logical Application System Environment Diagram
  - Scenario of Application System Environment
- Logical application system
  - Structure diagram of the logical application system
  - Scenario of Logical Application System
- Resource Architecture Environment
  - Resource Architecture Environment Diagram

## Environment diagram initialization

An environment diagram represents a use context of an application or an application system.

> 📖 *An application environment is used to represent a use context of an application. An application environment allows presenting the other application systems, applications, microservices or actors with which this application can interact.*

> 📖 *An application system environment allows presenting the other application systems, applications or microservices with which this application system can interact.*

In order to simplify the description of a specific use context of an application system, for example, **Add Items** button provides the list of components with which the application system interacts and helps you to select the objects you wish to add in your diagram.
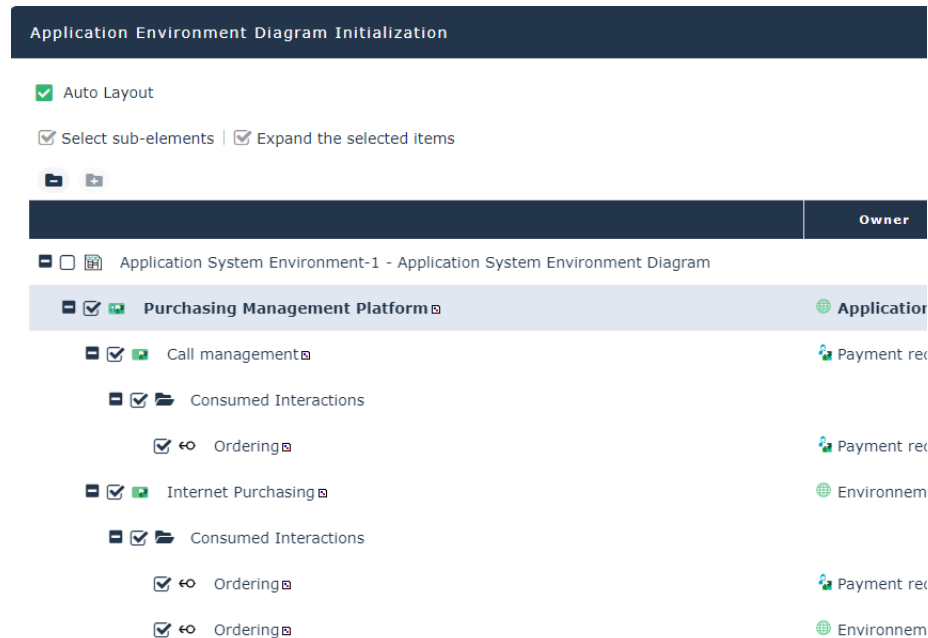
The **Sub-Elements selection** and **Expand selected elements** buttons help you in your selection.

## Creating a Sketching diagram with Hopex IT Architecture

A sketching diagram is a drawing that enables you to exchange with your coworkers without an issue of methodology or formalism.

Sketching diagrams can then be reworked and transformed into diagrams recognized by an **Hopex** Solution.

> ☞ *For more details ont the use of skecthing diagrams, see "Use a sketching diagram" in the **Hopex Common Features** guide.*

To create a *sketching diagram* for an application, for example, with **Hopex IT Architecture**:

1. From the **Applications** navigation menu, select the application of interest to you and click **Create a diagram**.
2. In the wizard window, select **Sketching diagram**.
   The diagram opens in the edit area. You are now in the **Hopex** graphic editor.

### Creating an ArchiMate@ diagram with Hopex IT Architecture

**Hopex for the ArchiMate® Framework** product provides facilities to use the set of concepts defined by the Open Group for ArchiMate®. ArchiMate® concepts are mapped with **Hopex** Enterprise Architecture building blocks so as to manage compatibility and continuity with other models.

You can associate a diagram based on ArchiMate@ formalism to an application, for example.

☛ *For more details, see "Using ArchiMate Diagrams in an Enterprise Architecture solution" chapter of the **Hopex IT Business Management** guide.*

### Using diagram comparison

The comparison of diagrams of an application system or architecture of an application system deployment enables to compare different versions of the same object.

☛ *For more details on the use of a diagram comparison, see "Comparing diagrams" chapter in the **Hopex Common Features** guide.*

## Hopex IT Architecture properties pages content

**Hopex IT Architecture** provides properties pages available for each object type.

☛

☛ *For more details on the access to property pages , see the chapter "Handling an object properties" in the **Hopex Common Features** guide.*

☛ *Using the facilities described in the **Hopex Power Studio** guide, you can customizing the properties pages of your solution.*

The pages below are common to main **Hopex IT Architecture** objects.

- the **Components** page gives access to the list of the described object components defined in the different diagram types.
  - the **Data Store** section gives access to the described object specific data stores.
    - ☛ *For more information on data stores, see  Managing Data.*
  - the **Internal components** section gives access to the list of object components defined in the workflow scenario.
  - the **Boundary Components** section gives access to the list of components in the structure diagrams of the described object.
    - ☛ *For more details an application object components, see Application structure diagram.*
- the **Scenario of flows** page displays the representations of the described object internal flows.
  - ☛ *For more details on flow scenarios, see Describing data flows.*
- the **Environments** page displays the described object contexts of usage.
  - ☛ *For more details on environments, see  Describing an Application Environment with Hopex IT Architecture.*
- The **Deployment Architecture** property page provides access to the list of components required for the architecture deployment.
  - ☛ *For more information on the components of a deployment architecture diagram, see Using an application deployment architecture diagram.*
- the **Usage** page displays information related to the described object context of usage in the different taypes of diagrams.
  - the **Scenario** section displays the list of elements containing the described object in the scenarios of flows.
    - ☛ *For more information on a scenario of flow, see Using a Scenario of Application Flows Diagram.*
  - the **Structure** section displays the list of elements having the described object in their environment.
    - ☛ *For more information on the components of an application structure diagram, see Application structure diagram.*
- the **Qualification** page provides access to the category of the described object.
  - ☛ *For more details on categories, see  Defining Data Categories.*
- the **Executed process** page provides access to the application processes executed by the described object.
  - ☛ *For more details on system processes, see Describing System Processes.*
- The **Gouvernance** page enabling to enter the architecture *Decisions* specific the application.
  - ☛
  - ☛ *For more information on Decisions, see "Drawing up an Application Inventory > Recording Architecture Decisions" chapter of the **Hopex IT Portfolio Management** guide.*

The page also displays the list of *Enterprise policies* the described object must comply to.

☛ *For more details on business policies management with Hopex IT Architecture, see Define a Policy Framework with Hopex IT Architecture.*

- the **Reports** page provides access to the reports available for the described object.

  ☛ *For more information on **Hopex IT Architecture** reports, see HOPEX IT Architecture Reports.*

- the **Diagrams** page displays the described object diagrams and enables to create new diagrams.

  ☛ *For more information on diagram types proposed by **Hopex IT Architecture**,see Using IT architecture diagrams.*

## Using duplication with Hopex IT Architecture

**Hopex IT Architecture** solution provides facilities for specific building blocks duplication such as applications, application systems or deployment architectures.

To duplicate an application system, for example:

1. Right-click the application system you want to duplicate to open its pop-up menu.
2. Select **Manage > Duplicate**.
   A dialog box opens to display the list of elements connected to the object to be duplicate.



*Hopex IT Architecture duplication dialog box*

3. Select the new building block **owner**.

4. For each component, the **Duplication Mode** column proposes the following options:
   - Create a **New copy** of the selected component. In this case, the name of the new component appears in the **New name for created object**.
   - **Keep** the component that will be owned by the source and the target building blocks.
   - **Ignore** the component that will not be duplicated or referenced.
5. Click **Launch Duplication** to validate your choices.
   The new building block can be accessed from its owner.

## Using duplication with Hopex IT Architecture in batch mode

A building block and the connected objects defined in the MetaModel, can be duplicated in batch mode.

The **SmartDuplicate** function provide the reference to the created building block.

The parameters of the **SmartDuplicate** function are defined in the *sOptions* string as follows:

- The function **SmartDuplicate**(ByVal *sOptions* As String) As MegaObject
- *sOptions* is definied with the following format "K1=V1,K2=V2, ...", the proposed value are:
  - *Root*=[NewCopy] - Default: NewCopy
  **ORoot** relates to the duplicated building block

  - *Components*=[Keep|NewCopy|Ignore] - Default: Keep
  Components relate to the components of the building block which is duplicated.*Components* relate to the components of the building block which is duplicated.

  - *Boundaries*=[Keep|Ignore] - Default: Keep
  Boundaries relate to the components of the building block which is duplicated.*Boundaries* relate to the components of the building block which is duplicated.

  - *Scenarios*=[Keep|Ignore] - Default: Keep
  Scenarios relate to the components of the building block which is duplicated.*Scenarios* relate to the components of the building block which is duplicated.

  - *Environments*=[Keep|Ignore] - Default: Keep
  Environments relate to the components of the building block which is duplicated.*Environments* relate to the components of the building block which is duplicated.

### *Example of use:*

```
Set newObject=
myApplication.SmartDuplicate("Root=NewCopy,Components=Keep,
Environments=Ignore")
```

# Using service catalogs

## Implementation of service catalogs

In **Hopex IT Architecture**, a service catalog is made of service catalog item. For example, a *hardware services catalog* is made up of several *hardware service catalog items*.

> 📖 *A hardware service catalog provides a centralized information source for the hardware services offered by the service provider organization. It contains a customer-oriented view of the hardware used, how they are supposed to be used, the processes that they support as well as the expected service quality level. The hardware service catalog presents the list of hardware functionalities mentioned as well as implementation recommendations.*

> 📖 *A hardware service catalog item defines which hardware functionality is in the catalog and which hardware artifacts provide the hardware functionality.*

A *Service catalog item* represents a service which is associated with a *capability* or a *functionality*, depending on the type of catalog. The capabilities associated to the services of a service catalog are connected to a capability map (or a functionality map) which is associated to the service catalog.

With **Hopex IT Architecture**, the technical solution providing a service is represented by an *Implementation*.

> 📖 *An implementation describes the relationship between a logical entity and a physical entity that implements it. The physical entity gives the list of logical entities that it implements.*

The table below draws up the summary of objects that implement the service catalog items according to their category.

| Type of service catalog | Type of service elements | Type of service solutions |
|---|---|---|
| Business function | Business capability | All types of technical and functional objects that implement a business capability with **Hopex IT Architecture**. For more details, see Describing Fulfillment of a Business Capability. |
| Cloud | Technology Capability Map | Cloud Services, see Using Cloud Services. |
| Technical | Technology Capabilities | Software technologies, see Describing a Software Technology. |
| Hardware | Hardware capabilities | Hardware and IoT Device, see Using Cloud Services. |

## Defining a service catalog

The management principle of a service catalog is identical for all types of service catalogs. The types of service catalogs offered are:

- *business service catalogs*,

  📖 *A business service catalog provides a centralized information source for the business services offered by the service provider organization. It contains a customer-oriented view of the services associated to business capabilities, how they are supposed to be used, the processes that they support as well as the expected service quality level. The business service catalog presents the list of functionalities mentioned as well as implementation recommendations.*

- *Cloud services catalogs*,

  📖 *An information service catalog provides a centralized information source for the information services offered by the service provider organization. It contains a customer-oriented view of the information services used, how they are supposed to be used, the processes that they support as well as the expected service quality level. The information service catalog presents the list of functionalities mentioned as well as implementation recommendations.*

- *Technology service catalogs*,

  📖 *A technology service catalog provides a centralized information source for the technology services offered by the service provider organization. It contains a customer-facing view of the technology services in use, how they are intended to be used, the process they enable, and the levels and quality of service the customer can expect from each service. The technology service catalog provides the list of reference technology capabilities and their recommended implementation.*

- *hardware service catalogs*.

  📖 *A hardware service catalog provides a centralized information source for the hardware services offered by the service provider organization. It contains a customer-oriented view of the hardware used, how they are supposed to be used, the processes that they support as well as the expected service quality level. The hardware service catalog presents the list of hardware functionalities mentioned as well as implementation recommendations.*

This chapter is based on the example of an *technology service catalog*.

## Creating a technology services catalog

To create a *Technology service catalog*:

1. From the **Inventories** navigation menu, select **Deployment > Technology Service Catalogs**.
   The service catalogs access page opens.
2. Click on the arrow to the left of the **Business service catalog** field and select **New**.
   A creation dialog box opens.
3. Enter the **Name** of your catalog and its **Owner**.
4. Select the **Technology capability map** associated to the service catalog and click **OK**.
   A new service catalog appears in the edit area.

   ☛ *Likewise, you can create a business service catalog or a hardware service catalog.*

## Adding a service catalog item

The **Characteristics** page of a service catalog provides access to:

- its **Name**,
- its **Owner**, by default, during creation of the logical application system, the current library.
- To access the list of service catalog items owned:

To add a *Business service catalog item*:

1. From the **Inventories** navigation menu, select **Software > Business Service Catalogs**.
   The service catalogs access page opens.
2. From the drop-down list of the **Business Service Catalog** field, select the Business Service Catalog that interests you.
   The list of service catalog items is displayed in the form of a tree that presents the capabilities associated to the services already declared as well as technical objects that represent the implemented solutions.
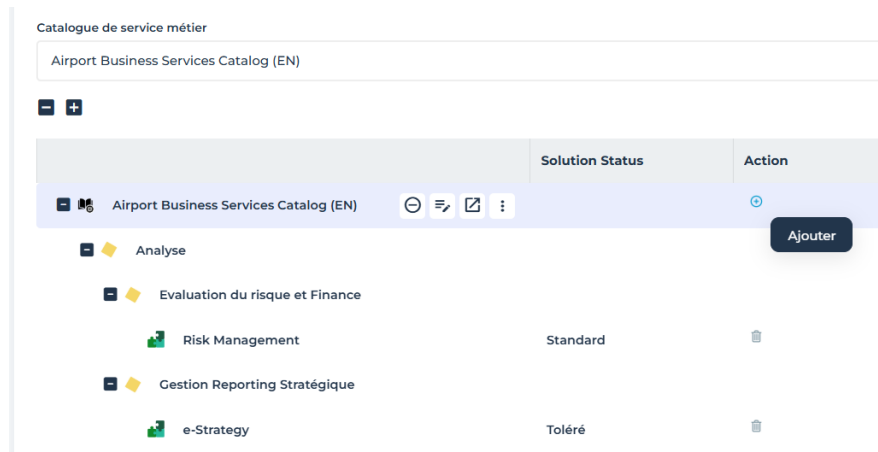


3. In the **New** column of a business service catalog, click the **Add** button.
   A  business service catalog item creation window opens.

4. Expand the tree branches to find the business capability that interests you.
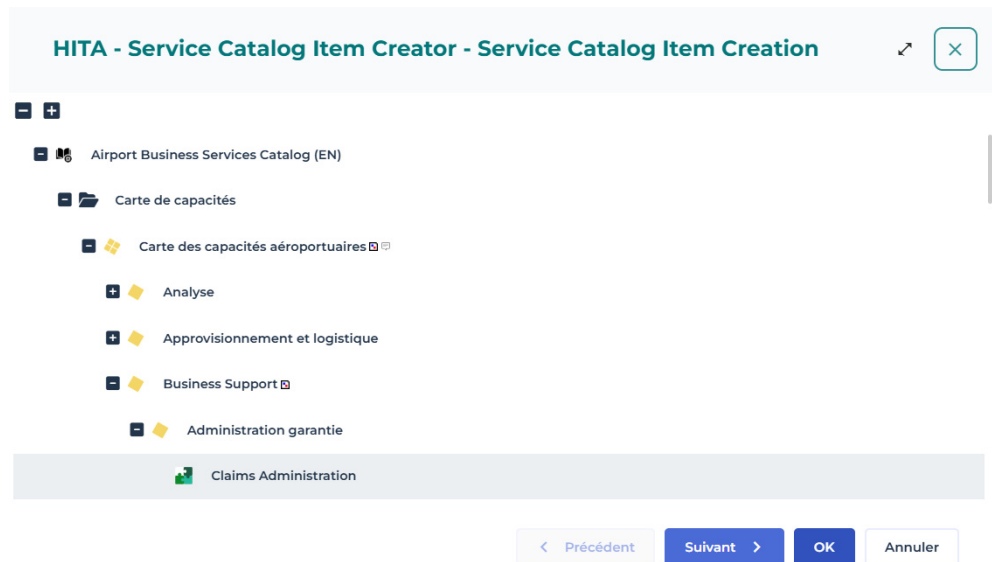


5. Select the object that implement the capability in the context of your business catalog.

☛ *For more details on implementation creation, see Using fulfillment mechanisms.*

**6.** Click **Next**.
A new page opens to specify the status of the object that represents the solution in the context of the service catalog.

- **Norm**: first choice, this choice is mandatory.
- **Accepted**: other choice accepted when it is not possible to use the Norm (please justify).
- **Tolerated**: tolerated for existing installations but for new uses.
- **Prohibited**: no installation should use this solution (existing or future).
- **Emergent**: for a solution currently under study.
- **To be defined**: for a solution to be defined.



**7.** Select a status and click **OK**.
The catalog item pops up in the list with the associated technology capability.

## Service catalog reports

The **Services Coverage Matrix** is a report presenting a list of service catalogs, the list of capabilities (or functionalities) covered, and for each of them, the implementation means.

### *Report parameters*

This consists of defining report input data.

| Settings | Parameter type | Constraints |
|---|---|---|
| List of service catalogs | All types of service catalogs:<br>- Business,<br>- information,<br>- technical,<br>- hardware. | A mandatory catalog. |

### *Accessing a service catalog report*

To access a service catalog report:

1. Open the **Reports** page of a service catalog.
2. Select the **Services coverage matrices** report.
3. In the **Parameters** section, select the service catalogs that you want to present in your report.
4. Click **Refresh the Report**.
   The new report appears.

## Using Workflows

With **Hopex IT Architecture** you can use standard workflows to manage:

- Validation requests;

  ☛ *For more details on validation requests workflows, see "Using validation requests" in* **Hopex Common Features** *guide.*

- Review requests.

  ☛ *For more details on the review process, see Process Validation Workflow.*

## Define a Policy Framework with Hopex IT Architecture

📖 *A business policy is a directive whose purpose is to govern or to guide the enterprise.*

📖 *A policy framework consists of a number of business policies. It is composed of sections and sub-sections that represent categories of business policies. Under these sections you can define the business policies, the assets constrained by the policies in question and their implementation.*

☛ *For more details on Policy Frameworks, see* **Hopex Data Governance** *guide.*

You can import in your **Hopex** repository some *Policy Frameworks*.

☛ *The Policy Frameworks are provided by your administrator in a package. For more details, see "Importing a module in* **Hopex**" *chapter in the* **HOPEX Administration** *guide.*

## Defining a Business Policy with Hopex IT Architecture

📖 *A business policy is a directive whose purpose is to govern or to guide the enterprise.*

### Accessing Business Policies with Hopex IT Architecture

To access the list of *Business Policies*:

1. From the **Governance** navigation menu, select **Policy Framework**. The *Policy Framework* tree appears.
2. Expand the folders to see *Business Policies*.

### Creating a business policy

To create a *Business Policy*:

1. From the **Governance** navigation menu, select **Policy Framework**. The *Policy Framework* tree appears.
2. Expand the folder of the *Policy Frameworks* that interests you.
3. Select the *Policy Framework Section* that interests you and click **New > Business Policy**. The new *Business Policy* appears in the tree.

### Connect a business policy to an application

To identify the *Business Policies* that your applications needs to comply with:

1. Open the **Governance** property page of the application that interests you.
2. In the **Policy to comply with** section, click **Connect**. A window opens providing a tree of policy frameworks and the associated business policies.
3. Select the business policies that interest you and click **Connect**. The selected business policies appear in the **Policy to comply** section.

### Assessing an application compliance with a business policy

The last compliance assessment result appears at the top of the application **Governance** property page.

Possible results are the following:

- **Full Compliance**: if all the business policies connected to the application are compliant.
- **No Compliance**: if all the business policies connected to the application are not compliant.
- **Partial Compliance**: if all the business policies connected to the application are assessed at different compliance levels.

To assess the compliance level of an application with a *business policy*:

1. Open the **Governance** property page of the application that interests you.
2. In the **Policy to comply with** section, select business policies you want assess.

3. In the **Policy Compliance Assessment** section, click **Evaluate**.
   A **Governance Assessment** window opens providing a tree of policy frameworks and the associated business policies to be assessed.
4. Specify the **Measure Date**.
5. In the **Compliance** column, select the compliance level you wish to assign to the application.
6. Click **OK**.
   The compliance level appears at the top of the property page.

## Defining an Architecture Principle

    📖 *An Architecture Principle is a general guideline that informs, supports and constrains the way in which an organization will design and construct architectures.*

### Accessing to Architecture Principles

To access the list of *Architecture Principles*:
1. From the **Governance** navigation menu, select **Policy Framework**.
   The *Policy Framework* tree appears.
2. Expand the folders to see the list of *Architecture Principle Categories*.

    📖 *An Architecture Principle Category enables a grouping of Architecture Principles for ease of management.*

### Creating an Architecture Principle

An *Architecture Principle* is owned by an *Architecture Principle Category*.

To create an *Architecture Principle*:
1. From the **Governance** navigation menu, select **Policy Framework**.
   The *Policy Framework* tree appears.
2. Expand the folder of the *Policy Frameworks* that interests you.
3. Select the *Architecture Principle Category* that interests you and click **New > Architecture Principle**.
   The new *Architecture Principle* appears in the tree.

### Defining an architecture principle scope

To define the scope of an *Architecture Principle*:
1. Open the **Characteristics** property page of the architecture principle that interests you.
2. In the **Scope** section, click **New**.
   A **Creation of Regulatory Requirement** window opens.
3. Select the **Object Type** that interests you, for example **Application Deployment Architecture**.
4. Select the **Element Subject to Regulation** and click **Add**.
   A new fulfillment of architecture principle element is created.

# Defining Data Categories

**Hopex IT Architecture** Solution enables data classification using *Data Categories*.

> ☞ *For more information on Data Categories see the **Hopex Data Governance** guide.*

To access the list of *data categories* from the **Administration** navigation menu:

☐ Select **Categorization Schemas > Data Categories**.
The list of Categorization Schemas appears.

> ☞

> ☞ *For more information on Categorization Schemas, see "Defining Categorization Schemas" chapter in the **Hopex IT Business Management** guide.*

# Defining Methodological Domains

Each **Hopex** Solution offers its own methodological domains defining all Solution users' common goals.

The methodological domains available in **Hopex IT Architecture** concern:

- The use of Cloud Services,
- Transition to a modern architecture,
- Reducing the IT carbon print.

# Importing components with Hopex IT Architecture

**Hopex IT Architecture** uses Excel data exchange wizards to export import and export existing architecture components.

> ☞ *For more details on Excel data exchange wizards, see the "Exchanging Data with Excel" chapter in the **Hopex Common Features** guide.*

Two Excel templates are proposed:

- **Hardware_Functionalities_Template.xlsx** for infrastructure elements import/export : computing devices connected to the breakdown of hardware functionalities they implement.
- **Technical_Functionalities_Template.xlsx** for technology elements import/export : technologies connected on one side to vendors, and on the other side to the breakdown of technical functionalities they implement.

# Structure of the import/export Excel templates of Hopex IT Architecture

**Hopex IT Architecture** Exel templates that enable import of harware or technical elements are dentically presented.

- At the level hardware, the elements are as follows: *Hardware*, *IoT Device*, *Server* and *It Device*.

  📖 *An IoT device is both a hardware device and a computing device which provides cvombined hardware and information services to the users using it directly. As a hardware device, it embeds sensors - e.g. accelerometer - which provide data to the embedded computing device. As a computing device, it can host data stores or run applications. Examples: smartwatch with GPS tracker, on-line surveillance video camera with live IP video feed, connected weighting scale with weight history management*

  ☞ *For more details on hardware elements, see Describing a Computing Device.*

- At the level of technologies, the elements are as follows: *Vendor (Org-Unit)* and*Software technology*.

  📖 *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.*

  ☞ *For more details on technologies, see Describing a Software Technology.*

- At the level of hardware of technical Capabilities, the elements are as follows:

  - *Hardware* or *technical Capabilities*

    📖 *A hardware capability is the ability to deliver a physical outcome which is required by an organizational resource in order to perform its work. This hardware capability is generally necessary within a computing process in order to execute a specific operation.*

  - *Hardware* or *Technical Capabilities maps*,

    📖 *A technology capability map is a set of technology capabilities and their dependencies that, together, defines the scope of a hardware or software architecture.*

  - *Sub-functionalities*, which define the link between a functionality and the functionality map (or the functionality) in which it is referenced.

    ☞ *For more details on functionalities, see Using Functionalities with Hopex IT Architecture.*

  - *Functionality fulfillments*, which define the link between a functionality and the hardware or technical object that implements it.

The list of information provided for in the Excel template delivered with **Hopex IT Architecture** is presented in the following order:

- For elements of type: *Hardware* or *technical Capability map*, *Computing device (IoT device*, *Server*, *It device)*:
  - **Short Name** : name of the object concerned.
  - **Comment** : object comment.
- For elements of type *Technology*:
  - **Short Name** : name of the object concerned.
  - **Technology Code**.
  - **Comment** : object comment.
  - **Vendor**.
- For elements of type *Org Unit* :
  - **Short Name** : name of the object concerned.
  - **Internal/External**.
  - **Org-Unit Type**.
  - **Comment** : object comment.
- For each element of *Functionality Composition* type:
  - Name of the composite object: functionality map or functionality,
  - Name of the owned functionality.
- For each element of *Functionality Implementation* type:
  - **Fulfilled Hardware/Technical Functionality**: name of the implemented functionality.
  - Name of the object (computing device or technology) that implements the functionality.

## Importing computing devices or technologies with Excel

☛ *For more information on the structure of the Excel template, see Importing components with Hopex IT Architecture.*

If you want to export computing devices or technologies or functionality maps that exist in another repository than your current one, for example, you can use the Excel template of **Hopex IT Architecture**.

Several steps must be followed in order for the Excel import of a business capability breakdown to be performed correctly:

1. (optional) Specifying the current library,
2. Exporting data from your repository with Hopex IT Architecture,
3. Completing the import file for Hopex IT Architecture,
4. Import your new file into your repository.

☛ *For more details on Excel data exchange wizards, see the "Exchanging Data with Excel" chapter in the **Hopex Common Features** guide.*

### *Specifying the current library*

This optional stage enables to connect imported objects to the current library.

A *library* and an *enterprise* are used to represent a unique work context.

📖 *Libraries are collections of objects used to split repository content into several independent parts. They allow creation of virtual partitions of the repository. In particular, two objects owned by different libraries can have the same name.*

In order for the data you import with Excel to be linked to a specific container, you must specify the current library.

☛ *For more details, see Defining the Work Environment.*

### Exporting data from your repository with Hopex IT Architecture

If you want to export technical or hardware  functionality maps that exist in another repository than your current one, for example, you can use the Excel template of **Hopex IT Architecture**.

☛ *For more details on Excel data exchange wizards, see the "Exchanging Data with Excel" chapter in the **Hopex Common Features** guide.*

When the Excel file is filled with the names of the objects you want to import, you must complete the necessary information for import into **Hopex IT Architecture**.

☛ *For more details, see Completing the import file for Hopex IT Architecture.*

### Completing the import file for Hopex IT Architecture

To get a correct import/export file, you must have specified the following elements:

- For each element of type *hardware* or *technical capability map*, *hardware* or *technical functionality map*, *Vendor*, *Technology*, *(IT, Server, connected device) Computing device*, you must enter the name of each object.
- For each breakdown (**Technical Function_Composion** or **Functionality Composition** Excel sheet), you must indicate:
  - The name of the composite object: functionality map or functionality,
  - The name of the owned functionality.
- To specify that a technology implements a functionality for example, you must indicate in the **Technical Function_Fulfillment** sheet:
  - the name of the functionality implemented in the **Fulfilled Hardware/Technical Functionality** column.
  - Name of the object (computing device or technology) that implements the functionality.

💣 **The first two lines of each Excel worksheet are reserved for file configuration; ensure that the first two lines of the imported file remain identical to those obtained after an export.**

☛ *For more information on the structure of the Excel template, see Structure of the import/export Excel templates of Hopex IT Architecture.*

## Using Tools of Conversion towards Hopex Aquila

If you use **Hopex Aquila** and if your administrator had carried out the conversion of **Business Person** instances coming from a repository previous to Hopex V5 into **Person (System)** instances, you must make sure that assignments have been created between the new **Person (System)** instances and the objects that was under  **Business Person** instances responsibility.

☛ *For more information on repository conversion, consult the technical note **How to migrate to HOPEX** .*

The conversion tool "Mega Repository - Convert Person and ARC Responsibility into Person (System) and Assignment" must be activated by your administrator using the **Environment automatic update** facility.

The principle of this **Hopex IT Architecture** conversion tool  is as follows:

1. A new **Person (System)** instance must be created for each **Business Person** instance with a **Responsibility** link with an **Application**, a **Database**, a **Network**, a **Node**, a **Server**, a **IT Service** or a **Workstation**.

2. For each **Responsibility** link between an asset and a previous **Business Person** instance, an assignment is created between the asset and the new **Person (System)** instance for each case described in the table of responsibilities to be replaced by assignments.

☛ *The Business Roles table used for assignments(below) specify the assignment to be created between a new **Person (System)** instance and an asset depending on the type of **Responsibility** link betwwen the converted **Business Person** instance and the asset type. If no business role is specified, no assignment is created and the **Business Person** instance is not converted.*

| Asset Type | IT Manager | Process manager | Quality Manager | Risk Manager or Provider |
|---|---|---|---|---|
| Application | Local Application Owner | Business User | - | - |
| Database | Data owner | - | Data Quality Manager | - |
| Network | - | - | - | - |
| Node | - | - | - | - |
| Server | - | - | - | - |
| IT service | Local Application Owner | - | - | - |
| Workstation | - | - | - | - |

*Table of business roles used for created assignments*

# ABOUT THIS GUIDE

This guide explains how to make best use of **Hopex IT Architecture** to ensure efficient management of IT Architecture.

## Guide Structure

The first part of the **Hopex IT Architecture** guide is composed of following chapters:

- Modeling Applications and System Architectures ; presents the functionalities offered by Hopex IT Architecture to describe the IT components of your enterprise.
- **Aligning IT and Business** ; explains how Hopex IT Architecture helps you in analyzing your Logical Architecture.
- Modeling technical architectures; explains how to  prepare the deployment of your IS components.
- Modeling IT Infrastructures; describes the fonctionalities proposed by **Hopex IT Architecture** to take into account systems using resources other than software.

The second part of the  **Hopex IT Architecture** guide comprises the chapters dedicated to UML.

## 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 Business Process Analysis** guide, which describes the functionalities proposed to manage processes;
- The  **Hopex IT Portfolio Management** guide, which describes functions proposed to manage all your applications;
- The **Hopex IT Business Management** guide, which describes functionalities proposed to manage your architecture transformation projects;
- the **Hopex Power Supervisor** administration guide.

# Conventions used in the guide

☛ *Remark on the preceding points.*

📖 *Definition of terms used.*

☺ *A tip that may simplify things.*

🦕 *Compatibility with previous versions.*

💣 **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**.
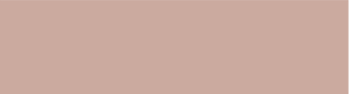
# Architecture Specification

# MODELING APPLICATIONS AND SYSTEM ARCHITECTURES

**Hopex IT Architecture** enables representation and documentation of IT architectures according to a service-oriented architecture.

Modeling an architecture according to a service-oriented approach facilitates the analysis of communications between architectures. Thus, the description of architectures is based on concepts that enable a more generic use of the tool.

Les points traités ici sont les suivants :

✓ Hopex IT Architecture Concepts Overview ;
✓ Describing an Application with Hopex IT Architecture ;
✓ Describing System architecture.

# HOPEX IT ARCHITECTURE CONCEPTS OVERVIEW

The information system can be broken down according to two levels of detail: the application and the application system.

## Application

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. An application meets:

- business requirements

    ```
    Examples: billing, accounting, equipment management, load/
    capacity calculation.
    ```

- technical requirements

    ```
    Examples: specific communication interface, access control.
    ```

- transverse requirements

    ```
    Examples: electronic mail, directories, office system
    applications.
    ```

For the creation of applications, see Describing an Application with Hopex IT Architecture.

## Application System

An application system is an assembly of applications responding to a coherent set of functionalities, implemented by the applications making up the system.

```
An application system can comprise a suite of applications
grouped for commercial reasons (integrated management
software packages such as SAP, Oracle Applications,
Siebel…).

An application system can also correspond to a group of
applications that have the same functional objectives
(accounts and financial management system integrating all
accounting applications: general, suppliers, analyses, as
well as financial and budgetary analysis modules, human
resources management systems integrating salaries, time
management, career management, etc.).
```

The application system, like an application, can be the subject of specific developments (carried out internally or bought-in/sub-contracted) or they can be proprietary market products (software packages).

The logical organization and structure of application systems and applications, together with description of their exchanges, constitutes the foundations of the application architecture. Thus, the representation of flows in an application system

enables identification of the impact of the  retirement of an application on the entire system.

For the creation of an application system, see Describing System architecture.

# DESCRIBING AN APPLICATION WITH HOPEX IT ARCHITECTURE

A project for describing the functional architecture of an information system is used to inventory the existing *applications* and their interactions.

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

## Creating an Application with Hopex IT Architecture

To create an *application*:

1. Click the **Applications** navigation menu.
   The list of applications appears in the edit area.
2. Click **New.**
3. The **Creation of Application** dialog box appears.
4. Enter the **Name** of your application.
5. (Optional) Select the **Owner**.
6. Click **OK**.
   The new application appears in the list.

# The properties of an application with Hopex IT Architecture

The **Characteristics** property page of an application provides access to different sections.

- The **Identification** section provides access to the following information:
  - the **Name**
  - its **Owner**, by default during creation of the application, the current library.
  - the text of its **Description**.
  - the internal **Code**.
  - the **Version number**.
  - the **Application Type**.
  - the **Cloud Computing**.
  - **Description**.
- the **Service Level Agreement** provides the **Maximum Tolerable Downtime (MTD)** as welle as the **SLA Level** of the application, from the following informations:
  - **Recovery Point Objective (RPO)**,
  - **Recovery Time Objective (RTO)**,
  - **Work Recovery Time (WRT)**.
- the **Functional scope** section of the application, see  Defining Application Functional Scope.
- **Specifications** section, see  Creating an application Use Case Diagram.
- the **Responsibility**: it relates to the person(s)  responsible for the application.
  - Local Application Owner,
  - IT Owner,
  - Business Owner.

    ☛ *For more details on these roles, see Business Roles of Hopex IT Architecture..*

- the **Technologies** section provides access to the list of *Software Technology* and the list of *Software Technology Stacks* used by the application.

    📖 *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.

   📖 *A software technology stack is a set of software technologies.*

   ☛ *For more details on software technologies, see Describing Software Technologies.*

- the **Exchange** section describes the application flows emitted and received by the application. See Using a Scenario of Application Flows Diagram.
  - for more details on **Data** section, see Managing Data.
  - the **Risks** section presents the risks associated with the application, see Describing an Application Environment with Hopex IT Architecture.
  - associated **Attachments**.

   ☛ *For more details on other property pages proposed by **Hopex IT Architecture**, see Hopex IT Architecture properties pages content.*

With **Hopex IT Architecture** an application is described by other property pages, see Hopex IT Architecture properties pages content.

# Defining Application Functional Scope

To indicate the objects that define application functional coverage:
1. Open the **Characteristics** property page of the application.
2. Expand the **Functional Scope** section.
   The types of data that define functional coverage of the application are:

- *Process Categories* using the application

   📖 *A process category defines a group of processes. It is linked to a Process Map or higher level Process Category. It regroups several processes and/or other categorized elements (e.g. Value Streams, Applications). It serves as an intermediate categorization level in the process hierarchy, so as to provide a guided and progressive access to finer grained processes.*

- *Business capabilities* covered by the application

   📖 *A business capability represents a specific ability that an organization possesses or needs to develop to deliver a particular business outcome.*

   ☛ *For more details on business capabilities, see Describing Business Capabilities with Hopex IT Architecture.*

   ☛ *A report covers distribution of applications in business capabilities, see Reports on the Architecture Functional Coverage .*

- The *Implemented Functionalities* fulfilled by the application.

   📖 *A functionality is a service required to perform a work. This functionality is generally necessary within an activity in order to execute a specific operation. If it is a software functionality, it can be provided by an application.*

   ☛ *For more details on functionalities, see Describing a Functionality Map with Hopex IT Architecture.*

   ☛ *For more details on fulfillments, see Using fulfillment mechanisms.*

- *Logical Fulfillments* of the application are the logicial application or logical application systems fulfilled by the application.

   ☛ *For more details on fulfillments, see Using fulfillment mechanisms.*

# Describing structure and services of an application

At first, an application can be described from a logical point of view, see Describing Logical Application Architecture.

However, and from a concrete point of view, an *application* is described by several types of diagram;

- an *application structure diagram* is used to represent the service interactions between the application components using service interfaces.

  ☛ *For more details, see Application structure diagram.*

- A *Scenario of application flows* describes the flows exchanged between the IT services or the microservices used by the 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.*

- *Scenario of sequence of flows* presents the agents necessary for the scenario (IT services, microservices, data stores) and exchanged sequenced application flows.

  ☛ *For more details, see Creating a flow scenario sequence diagram.*

- an *application deployment architecture* used to represent technical elements that support the application.

  ☛ *For more details, see Describing an Application Deployment Architecture.*

- A *Use Case Diagram* used to represent the exchanges between the application and actors, according an UML approach.

  ☛ *For more details, see Creating an application Use Case Diagram.*

For more details on modeling applications with **Hopex IT Architecture**, see Modeling application architectures.

# Describing an Application Environment with Hopex IT Architecture

     📖 *An application environment is used to represent a use context of an application. An application environment allows presenting the other application systems, applications, microservices or actors with which this application can interact.*

## Describing an Application Environment

     📖 *An application environment is used to represent a use context of an application. An application environment allows presenting the other application systems, applications, microservices or actors with which this application can interact.*

An *application environment* is described by several types of diagrams:

- An *Application Environment diagram* describes the exchanges between the subject application and its partners in a specific context.

  ☛ *For more details, see Application Environment Diagram presentation.*

- a *scenario of application environment flows* describes the flows exchanged between the described application and its partners: applications, application systems,  IT services or microservices used by the described application in a specific context.

  ☛ *For more details, see Using a Scenario of Application Flows Diagram.*

- a *scenario of sequences of flows* presents the agents necessary for the scenario (application, IT services, microservices, data stores) and sequence application flows exchanged.

  ☛ *For more details, see Using a flow scenario sequence diagram.*

## Accessing the List of Application Environments

To access the list of application environments from the **Applications** navigation menu:

❱ Open the **Environments** page from the application of you interest.
The list of application environments appears in the edit area.

## Creating an application environment

To create an *Application environment:*:

1. Open the **Environments** page from the application of you interest.
   The list of application environments appears in the edit area.
2. Click **New**.
   The new application environment appears in the list under the name "Environment" followed by the name of the application.

## Application environment properties

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

- its **Owner**, by default during creation of an application system environment, the current library.
- its **Name**,
- the text of its **Description**.

  ☛ *For more details on other property pages of the application environment, see Hopex IT Architecture properties pages content.*

The **Components** property page of the application environment provides access to partners elements:

- Applications
- Microservice,
- IT Services,
- System users.

☛ *For more information on the components of an application environment diagram, see Application Environment Diagram presentation.*

## Application Environment Diagram presentation

With **Hopex IT Architecture**, an *application environment* is entirely described by a an application environment diagram that is used to describe the service interactions between the environment applications described, its users and the external applications.

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 *microservice partners* 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 a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

- *Service interactions* between components.

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

- request and service points

## Specifying the Risks associated with an Application

**Hopex IT Architecture** is used to identify the risks associated with an application, and to retrieve the evaluations defined in the **Hopex Enterprise Risk Management** solution. You can define a new risk using the application or connect a previously defined risk.

To connect a risk to an application:
1. Open the **Characteristics** property pages of the application.
2. Expand the **Risk** section.
3. Click **Connect**.
   The query dialog box appears.
4. Find and select the risk required and click **OK**.

For more details on risks and their evaluation, see **Hopex Enterprise Risk Management**.

# DESCRIBING SYSTEM ARCHITECTURE

## Describing an Application System

A project for describing the functional architecture of an information system is also used to inventory the existing *application systems* and their interactions.

> 📖 *An application system is an assembly of other application systems, applications and end users interacting with application components to implement one or several functions.*

An *application system* is described by several types of diagrams:

- An *application structure diagram* is used to represent the service interactions between the application components using service interfaces.

  > ☛ *For more details, see Creating an application system structure diagram.*

- An *Application System Deployment Architecture*. used to represent the technical architecture chosen for the deployment of each component that support the application system as well as the techniques used for their communications .

  > ☛ *For more details, see Describing an Application System Deployment Architecture.*

- A *scenario of application system flows* presents the flows exchanged between the application systems, the applications or the microservices used by this application system. A scenario can represent a particular use case of the application system or more globally all the flows exchanged within this application system.

  > ☛ *For more details, see Using a Scenario of Application System Flows.*

- *Scenario of sequence of flows* presents the agents necessary for the scenario (IT services, microservices, data stores) and exchanged sequenced application flows.

  > ☛ *For more details, see Using a flow scenario sequence diagram.*

## Creating an Application System

To create an *application system*:

1. Click **Application Systems** navigation menu.
   The list of application systems appears.
2. Click **New**.
   The **Creation of Application System** dialog box appears.
3. Enter the **Name** of your application system and click **OK**.
   The new application system appears in the list.

# Application System Properties

The **Characteristics** property page for an application system provides access to several sections.

- The **Identification** section provides access to the following information:
  - the **Name**,
  - its **Owner**, by default during creation of the application system, the current library.
  - the text of its **Description**
  - the internal **Code**,
  - the **Version number**,
  - **Description**.
- About the **Functional Scope** section of the application system, see Defining Application Functional Scope.
- the **Use Cases** section, see  Creating an application Use Case Diagram.
- The **Responsibility** section relates to the person(s) responsible for the application system.
  - Software Designer
  - Local Application Owner

    ☛ *For more details on these roles, see Business Roles of Hopex IT Architecture..*

- The **Attachments** section is limited to associated attachments.

    ☛ *For more details on other property pages proposed by **Hopex IT Architecture**, see Hopex IT Architecture properties pages content.*

With **Hopex IT Architecture** an application system is described by other property pages. See Hopex IT Architecture properties pages content.

# Creating an application system structure diagram

This diagram describes the internal structure of an application system:

- services offered or required,
- the application components and their interactions; these are application systems service, applications and microservices,
- the end users interacting with the application components.

```
The following diagram describes the application system
corresponding to purchasing requests processing.
```
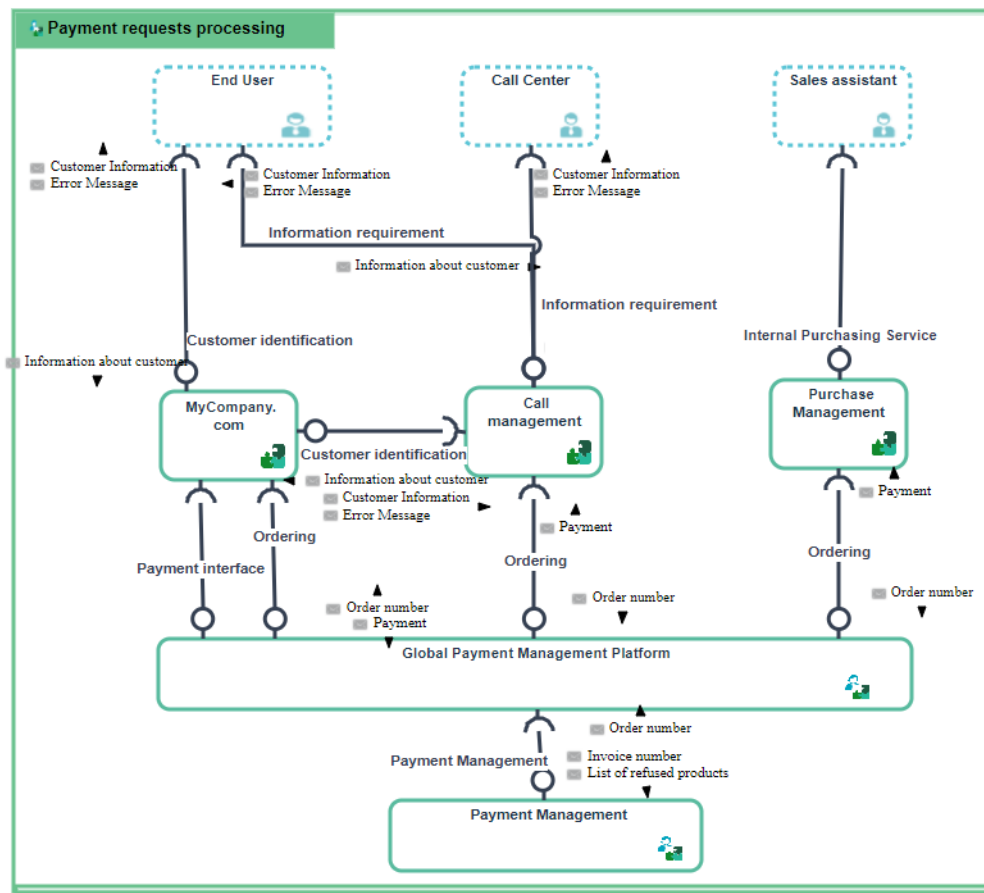


*The following diagram describes the application system corresponding to purchasing requests processing.*

To create an application system structure diagram:

1. Open the **Diagrams** page of the application system of your choice and click **Create a diagram**.
2. Select **Structured diagram > Internal Architecture**.
   The **application system structure diagram** appears.

### Adding an application system to an application system structure diagram

To describe an application system that implements another application system, you can add an *application system* of the application system structure diagram.

> For example, the purchasing requests processing system uses the "Purchasing Management Platform" and "Payment Management" application system services.

To add an **Application System**:

1. In the objects toolbar of the application system structure diagram, click **Application System**.
2. Click in the frame of the described application system.
   An addition window box prompts you to choose the **application system** implemented (for example "Payment management").
3. Select an application system.
4. Click **OK**.
   The application system appears in the diagram.

### Adding an end user to an application system structure diagram

To specify that an application system, such as purchasing request processing, is activated by internal or external org-units, you will add an associated *end user*.

> 📖 *The end user represents an organizational unit interacting at the boundaries of an application system or a logical application system.*

To add an *end user*:

1. In the application system structure diagram objects toolbar, click **End User** and click in the frame of the diagram.
   An addition window prompts you to choose the **Object Type** that you wish to use: *Org-Unit* or *Position type*.
2. For example, select the **Org-unit** object type.

   > 📖 *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

   > 📖 *A position type represents a status assigned to an individual or a group of individuals with the aim of defining an organization or a hierarchy.*

3. Select the org-unit that interests you and click **OK**.
   The actor appears in the diagram.

# Using a Scenario of Application System Flows

☛ *For more details on the use of a scenario of flows, see* *Using a Scenario of Application Flows Diagram.*

A scenario of application system flows represents the flows exchanged between certain elements of the application system in a given context. The elements represented are:

- application systems,
- applications,
- Microservices
- organization org-units,
- internal or external local application data stores,
- input or output application ports.

The interactions offered between these elements:

- flows that carry a content,
- 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.

```
The scenario of application system flows below describes
the interactions between a client and the eCommerce
application.
```
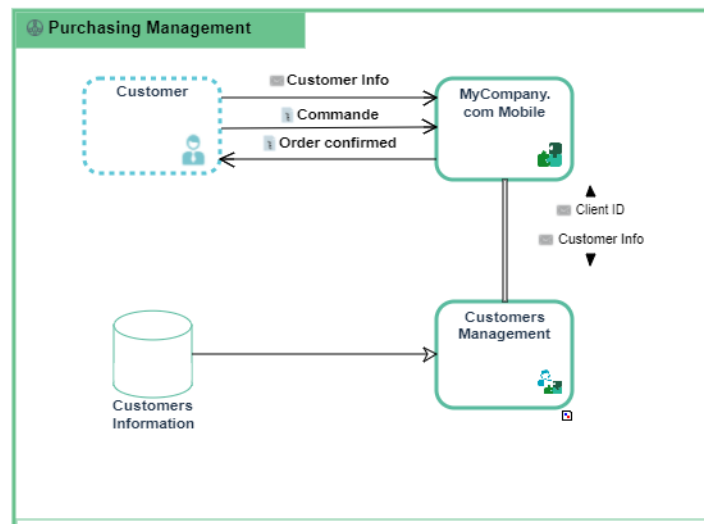


*Example of scenario of application system flows for "Purchasing Requests Processing".*

To create a scenario of application system flows diagram:

1. Open the **Diagrams** page of the application system of your choice and click **Create a diagram**.
2. Select **Structured Diagram > External Data Flows**.
   The **Scenario of application system flows diagram** appears.

### *Adding an org-unit to the Scenario of Application System Flows*

An org-unit is represented by an *Org-Unit* or by a *Position type*.

    📖 *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

    📖 *A position type represents a status assigned to an individual or a group of individuals with the aim of defining an organization or a hierarchy.*

To add an organization unit:
1. In the scenario of application system flows object toolbar, **Org Unit**.
2. Click in the frame of the described application system.
   An addition window prompts you to choose the org-unit name you wish to use:
3. Select the org-unit that interests you and click **OK**.
   The actor appears in the diagram.

    ☛ *To create a new org-unit, enter his name and click **OK**.*

### *Adding a scenario of application system flows*

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

To add an **application**:
1. In the scenario of application system flows object toolbar, click **Application**.
2. Click in the frame of the described application system.
   An addition dialog box prompts you to choose the **application** that you want to use (for example "eCommerce purchase").
3. Select the application and click **OK**.
   The application appears in the diagram.

In the same way you can add:

- an application system

    📖 *An application system is an assembly of other application systems, applications and end users interacting with application components to implement one or several functions.*

- a microservice.

    📖 *A microservice 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.*

If the component we have added in the scenario of application system flows is already described by a scenario of flows, a new section is created in the **Characteristics** property page of the component.

For more details, see Reinitializing components in a scenario of flows.

# Describing an Application System Environment with Hopex IT Architecture

> 📖 *An application system environment allows presenting the other application systems, applications or microservices with which this application system can interact.*

## Accessing the list of application system environments

To access the list of application system environments from System the **Application Systems** navigation menu:

❱ Open the **Environments** page of the application system of your choice. The list of application system environments appears in the edit area.

## Creating an application system environment

To create an *Application System Environment* from the **Application Systems** navigation menu:

1. Open the **Environments** page of the application system of your choice. The list of application system environments appears in the edit area.

2. Click **New**.
   The new application system environment appears in the list, it has the name of the application system followed by "Environment".

## Application system environment properties

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

- its **Owner**, by default during creation of an application system environment, the current library.
- its **Name**,
- the text of its **Description**.

With **Hopex IT Architecture** an application system environment is described by other property pages. See Hopex IT Architecture properties pages content.

## Application system environment diagrams

An *application system environment* is described by several types of diagram:

- an *Application System Environment* diagram describes the exchanges between the subject application system and its partners in a specific context.

  ☛ *For more details, see Describing an application system environment diagram.*

- a *scenario of application system environment flows* presents the flows exchanged between the application services or the microservices used by the described application system in a specific context.

  ☛ *For more details, see Describing a Scenario of Application System Environment Flows.*

### *Describing an application system environment diagram*

An application system environment is described by an **application system environment diagram** that describes the service interactions between the internal application systems, its users and the partner application systems.

☛ *For more details on use of a structure diagram, see Application structure diagram*
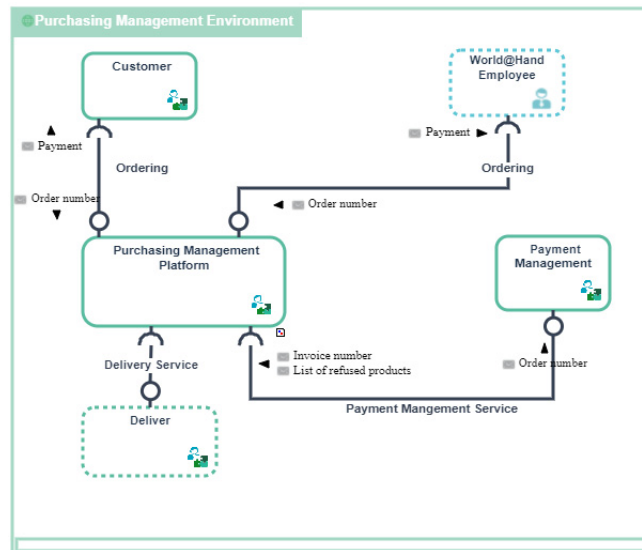


*Application system environment diagram for the Purchasing Requests*

```
Purchase requests are formulated by clients or employed
using the "Purchasing Management Platform".

The "Purchasing Management Platform" application system
uses an internal application system for the "Payment
management" and a partner application system for the
"Delivery".
```

The elements of an application system environment diagram are:

- the main *application system* principal described by the environment.

  📖 *An application system is an assembly of other application systems, applications and end users interacting with application components to implement one or several functions.*

- *partner application systems* that represent the other application system with which the main application system described by the environment interacts.

  ```
  In this example, this concerns two loan services offered to
  individuals and companies.
  ```

  📖 *A partner application system is an application system external to the environment of the described application service. The partner*

> *application system can be a service supplier or a service consumer with respect to application system users.*

- The categories of users of services provided by the environment are represented either by an *Org-Unit* or by a *Position Type*.

  > &#128214; *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

  > &#128214; *A position type represents a status assigned to an individual or a group of individuals with the aim of defining an organization or a hierarchy.*

  ```
  This concerns two user categories: individuals and
  companies.
  ```

- *Service interactions* between components

  > &#128214; *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

### Describing a Scenario of Application System Environment Flows

A scenario of application system environment represents the flows exchanged between the components of the application system environment.

> &#9758; *For more details on use of a scenario of flows, see Using a Scenario of Application Flows Diagram*

The elements of a scenario of application system environment are:

- the main *application system* principal described by the environment.

  > &#128214; *An application system is an assembly of other application systems, applications and end users interacting with application components to implement one or several functions.*

- *partner application systems* that represent the other application system with which the main application system described by the environment interacts.

  > &#128214; *A partner application system is an application system external to the environment of the described application service. The partner application system can be a service supplier or a service consumer with respect to application system users.*

- *End User Participants* that represent the categories of users of application system provided by the environment.
- The categories of users of services provided by the environment are represented either by an *Org-Unit* or by a *Position Type*.

  > &#128214; *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external*

org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.

   📖 *A position type represents a status assigned to an individual or a group of individuals with the aim of defining an organization or a hierarchy.*

- *Service interactions* between components

   📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

# MODELING APPLICATION  ARCHITECTURES

With **Hopex IT Architecture**, an application is described by the application flows processed, the components (services and API) providing the functionalities expected by the business and the environment components interacting with it.

After describing the functionalities requested by an application to meet business requirements, this chapter describes how to describe the flows and the structure of applications.

The following points are covered here:

- ✓ Describing data flows.
- ✓ Describing the structure and services of an application;
- ✓ Describing System Processes;
- ✓ Managing Data.

# DESCRIBING DATA FLOWS

## Defining a data flow and its usages

> 📖 *A flow represents the circulation of information between two agents in the Information System (for example, two applications, an application and an actor, etc.). This flow is defined by a sender, a receiver, and exchanged content, symbolizing the data being transported. A flow is defined in absolute terms.*

> 📖 *An application flow represents the use of a flow between two agents (e.g., applications) in a usage context (represented by a scenario of flows). An application flow is based on a flow, which represents the reference flow in context.*

A scenario of flows represents a context in which an agent is used: an application or an IT system, for example. An *application flow* describes the exchange of data in a unique usage context. However, the same exchange of data can be described in another usage context by another application flow.

An *application flow* is owned by a single flow scenario and refers to a single *flow*.

A flow enables the representation of data exchanges common to several contexts, i.e., several flow scenarios. A *flow* can be connected to several application flows.

### Flow qualification

*Measure Schemes Categorization* enable the definition of parameters that characterize the *flows*.

To qualify a flow from a *Flow measure*:

1. Open the **Qualification** page of the flow that interests you.
2. In the **Flow Qualification** section, click the **Connect** button.
   A selection dialog box opens displaying the tree of existing *Measure Schemes Categorizations*.
3. Select the flow measures that interests you and click **Add**.

   > ☛ *To access the list of Application Flow Categories: using the* ***Administration*** *navigation menu, select* ***Categorization*** *and unfold the* ***Measure schemes Categorization***. *The list of Application Flow Categories appears.*

   > ☛ *For more details on the* *Application Flow category* *concept, see* *Defining Data Categories.*

### Associating a Service Interface Used to a flow

> 📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

> 📖 *A service interface use is associated to a service interface. It enables representation of complex exchanges.*

> ☛ *For more details on* *service interfaces*, *see* *Describing a service interface.*

A service interface can be connected to a flow. This enables the identification of the API used in a flow context, for example.

There are two ways to use a service interface within a flow:

- The flow triggers the service interface (API). It is then of **Call/Request** type.
- The flow is the result of the service interface (API). It is then of **Result/Provision** type.

To associate a *service interface* to a *flow*:

1. Open the **Service Interface Used** page of the application flow that interests you.
2. Click **New**.
   A dialog box opens proposing a list of *service interfaces used* whose content is a request point or a result.

---

## Using a Scenario of Application Flows Diagram

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

Two types of diagrams are proposed:

- The *Scenario of flows diagrams* that describe the flows exchanged in different use scenarios of the object described.
- *Scenario of sequence diagrams* that describe the chronology of the flows exchanged in different use scenarios of the object described.

> ☞ *To use Scenario of Application Flows Diagrams, open the* **Options** *window and check that* **IT Architecture > Activate Flow Scenario Sequence Diagrams** *option is activated.*

A Scenario of Application Flows Diagram can be built for an application environment, an application, an application system, an IT service or a microservice. 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.
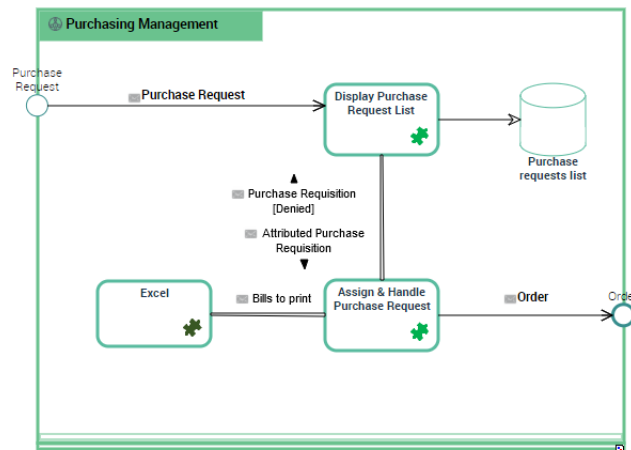


*Example of a Scenario of Application Flows for "Managing Purchase Orders".*

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

- IT Services,services, see Describing an IT Service with Hopex IT Architecture,
- Microservices, see Describing a microservice with Hopex IT Architecture,
- internal or external local application data stores, see  Using Data Stores,
- *System Triggering Events* and *System Triggered Events*, see Creating a System Triggering Event.

The interactions offered between these elements:

- application flows connected to flows,
- 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 from the **Applications** navigation menu:

1. Right-click the application that interests you and click **Create Diagram**.
2. In the Create a diagram window, select **Structured diagram > Internal Architecture**.
   The Scenario of Application Flows Diagram appears.

## 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 **IT Service**.
2.  Click in the described application frame.
    An addition window box prompts you to choose the **IT Service** implemented (for example "Customer management").
3.  Select the IT Service required and click **OK**.
    The IT Service appears in the diagram.

You can add a micoservice in the same way.

> 📖 *A microservice 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.*

## Creating an Application Flow

> 📖 *An application flow represents the use of a flow between two agents (e.g., applications) in a usage context (represented by a scenario of flows). An application flow is based on a flow, which represents the reference flow in context.*

The application flows exchanged between IT services, microservices, or application ports in a scenario of application flows are associated with a *flow*, which is itself associated to a *communication system* and a *content*.

> 📖 *A flow represents the circulation of information between two agents in the Information System (for example, two applications, an application and an actor, etc.). This flow is defined by a sender, a receiver, and exchanged content, symbolizing the data being transported. A flow is defined in absolute terms.*

> 📖 *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.*

When creating an *application flow*, you must associate it with a *communication system* and a *content*.

To create an *application flow*:

1.  In the objects toolbar of the scenario of application flows, click **Application flow** and select the **Type of application flow** that interests you.
    *   **Result/Provision** associated to a service result,
    *   **Call/Request** associated to a service request,
    *   **Signal** associated to an information exchange.

    > ☞ *For more details, see Associating a Service Interface Used to a 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.  Select the *communication system* that will be associated to the *flow*.

    > ☞ *For more details on communication systems, see Using communication systems.*

4. From the **Content** field, select the content you wish to associate with the message flow.
5. Click **OK**.

If it doesn't exist yet, a *flow* is created.

The *application flow*, represented by an arrow between the sender and the receiver, is displayed in the diagram. By default, the name of the *flow* is displayed on the link.

☛ *To display the Content and the Flow on the application flow link, click the application flow to display its contextual menu and select* **Shapes and Details**. *From the* **Flow** *folder, check the* **Short Name Content** *box.*

## Accessing Application Flow Properties

 *An application flow represents the use of a flow between two agents (e.g., applications) in a usage context (represented by a scenario of flows). An application flow is based on a flow, which represents the reference flow in context.*

To access an *Application Flow* properties from a scenario of application flows diagram:

1. Select the link connected to the application flow that interests you:
The application flow properties open to the right of the diagram.
2. In the **Characteristics** page, the **Flow** field enables the access to the associated *Flow* properties.
3. In the **Diagrams** page, enables the access or the creation of a *Communication Chain* diagram associated to the *Flow*.

☛ *For more details on communication chain, see, Using Software Communication Chains.*

## Accessing a flow properties

You can access a *flow* properties from the application flow that references it or from a scenario of flow diagram.

 *A flow represents the circulation of information between two agents in the Information System (for example, two applications, an application and an actor, etc.). This flow is defined by a sender, a receiver, and exchanged content, symbolizing the data being transported. A flow is defined in absolute terms.*

To access a flow from a scenario of application flows diagram:

- ☐ From the link of the application flow, click the flow name that interests you.
  The flow properties open to the right of the diagram.
- The **Characteristics** page provides access to the following sections:
  - **Identification**, presenting the **Name** of the *flow*, its **Description**, the **Application flow type** and the flow **Code**.
  - **Implementing software**, presenting the **Communication system** owned by the *flow* as well as the **application communication chain**.
    - ☛ *A flow can be linked to several application communication chains.*
  - **Qualification**, presenting the *Flow Measures* defined for the *flow*.
    - ☛ *For more details on flow qualification, see Flow qualification.*
- The **Usage** page provides the list of application flows connected to the flow.
- The **Service Interface used** page provides access to the interfaces and operation services used by flows.
  - ☛ *For more details, see Associating a Service Interface Used to a flow.*
- The **Diagrams** page enables the access or the creation of a *Communication Chain* diagram associated to the *Flow*.
  - ☛ *For more details on communication chain, see, Creating a software communication chain from a scenario of flow.*

## 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** 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 application flows of the scenario of application system flows.
3. Select the flows that you want to group and click **OK**.
   The content of the selected flows is displayed in the **Grouped Flow** list.
4. Click the **Refresh Channels** button.
   The application flows grouped in the channel disappears and the corresponding content is displayed around the channel.
   - ☛ *If you remove the channel, only the application flows created from the **Grouped Flows** are removed. The connected application flows are displayed if you click the **Refresh Channels** button.*

## Creating a System Triggering Event

The creation process for a *Creating a System Triggering Event* and a *Creating a System Triggered Event* is the same.

To create a *System Triggering Event*:

1. In the diagram insert toolbar, click the **System Triggering Event** button.
2. Position the object at the edge of the frame of the described object.
   A creation dialog box opens.
3. Click the arrow at the right of the **Referenced Content** field and select the content that interests you.
4. Click **Add**.
   The *System Triggering Event* appears in the diagram.

Any application flow whose origin is the *System Triggering Event* is connected to the same content.

To create an application flow from a *System Triggering Event*:

1. In the diagram insert toolbar, click **Event Participation**.
2. Click the event and, holding the mouse button pressed, draw a link to the object receiving the flow.
   The application flow is displayed with its content associated to the event.

### Reinitializing 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** 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 flow.

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.*

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.
   An addition window prompts you to choose the **Object Type** that represents the physical structure that will concretely support the application data store.

   ☞ *For more information on managing data stores, see *Managing Data*.*

3. Depending on the **Object type**, select then the object 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 microservices can have read or write 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 communication systems

> 📖 *A communication system helps to identify and describe the main integration processes using several Software Communication Chains as well as communication services.*

This *communication system* has its own internal agents (communication services) that enable the definition of communication chains that describe all the steps involved in a flow behavior.

Those chains may be described using the **Enterprise Integration Pattern** notation (http://www.enterpriseintegrationpatterns.com).

This representation allows the modeling of application flows integration process represented in scenario of flows in **Hopex**.

> ☛ *For more details on scenario of flows, see Using a Scenario of Application Flows Diagram.*

> ☛ *For more details on how associating a communication system with a flow, see Creating an Application Flow.*

## Accessing the list of communication systems

To access the list of communication systems from the **Inventories** navigation menu:

- Select**Software > Communication systems**.
  The list of communication systems appears.

## Communication System Properties

The complete description of a communication system is accessed in its property pages.

The **Characteristics** page for a communicate system provides access to:

- its **Name**,
- Its **Owner**, by default the application specified when it was created.
- the text of its description.
- the **Software Communication Chain** section which provides access to the list of components of the described communication system.

  > 📖 *A software communication chain describes the mechanism by which a content is transfered from a sender system to a receiver system.This includes, routing, channeling and message translation.*

- the **Communication Service** section which provides access to the list of objects of the software communication chain.
  Three services types can be proposed:

  - *Message Channel*,

    > 📖 *A channel is used to identify the enterprise resources used by a persona to achieve a step. For example, a channel can be a phone or internet connexion.*

  - *Message Router*,

    > 📖 *A message router is a communication step that identifies which route should be used for next message step.*

  - *Message Translater*,

    > 📖 *A message translator is a communication step that translates a message from a format to another. It can be used for trans-codification, data type conversion.*

    > ☞ *For more details on components of a communication chain, see Describing a Software Communication Chain*

The **Implementing Software** properties provides the liste of *Applications* et *Application Systems* representing the communication system execution.

The **Managed Flow** properties provides access to the list of *Flows* connected to the communication system. For more details, see *Creating an Application Flow*.

The **Reporting** page enables access the Flow rationalization report. For more details, see Flow Process Rationalization.
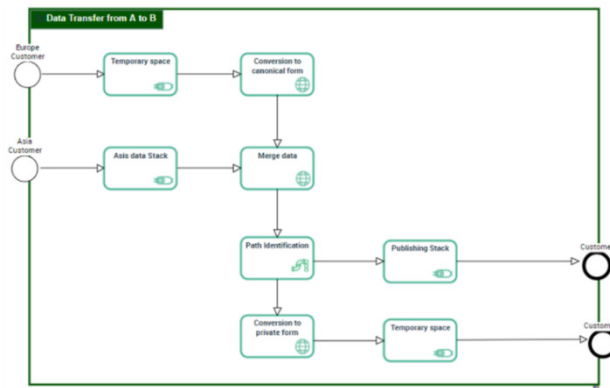
## Using Software Communication Chains

### Describing a Software Communication Chain

> 📖 *A Software Communication Chain Diagram describes the mechanism by which a content is transfered from a sender system to a*

*receiver system. This description includes the routing, the channeling and the messages translation.*



A software communication chain diagram includes:

- A *Communication Start Event* and a *Communication End Event* that designate the starting and the ending points of the described integration process.
- The *Communication Sequences* to describe the steps sequence.
- The *Communication Services* to describe the successive processing of flows. For more details the specification of softwares that implement the communication services, see Describing implementation of a communication service

  The *Communication Service* types are:
  - *Message Channel* that designate the place where an application can read or write informations.

    📖 *A channel is used to identify the enterprise resources used by a persona to achieve a step. For example, a channel can be a phone or internet connexion.*

  - *Message Router* to identify the destination channel to use for the next transport step.

    📖 *A message router is a communication step that identifies which route should be used for next message step.*

  - *Message Translater* to translate a message from a format to another

    📖 *A message translator is a communication step that translates a message from a format to another. It can be used for trans-codification, data type conversion.*

### *Creating a software communication chain from a scenario of flow*

You can create new communication chains from flows or application flows owned by a scenario of flows.

☛ *For more details on scenario of flows, see Using a Scenario of Application Flows Diagram.*

To create a software communication chain from a flow:

1. Open the flow scenario diagram that contains the flow that interests you.
2. Select the flow that interests you to open its **Diagrams** properties.

3. Click **Create a diagram**.
   The diagram creation dialog box appears.
4. Select **Software Communication Chain Diagram**.
   The software communication chain is created as well as its diagram.
5. You can modify the **Name** of the software communication chain from its diagram.

> ☛ *Several software communication chains can be connected to the same flow.*

### Describing implementation of a communication service

A communication service can be processed by an application, a microservice or an application service.

To specify the software that implements a communication service; a router, for example:

1. Open the **Characteristic** property page the Router you are interested in.
2. Expand the **Implemented Software** section.
3. Click **Connect**.
4. In the search window, select the software that represents the execution of the service associated with the router.
5. Click **OK**.
   The software appears in the list.

## Using a flow scenario sequence diagram

> ☛ *To use Scenario of Application Flows Diagrams, open the **Options** window and check that **IT Architecture > Activate Flow Scenario Sequence Diagrams** option is activated.*

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

For each use context, you 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 diagram contains:

- Lines which define service 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.
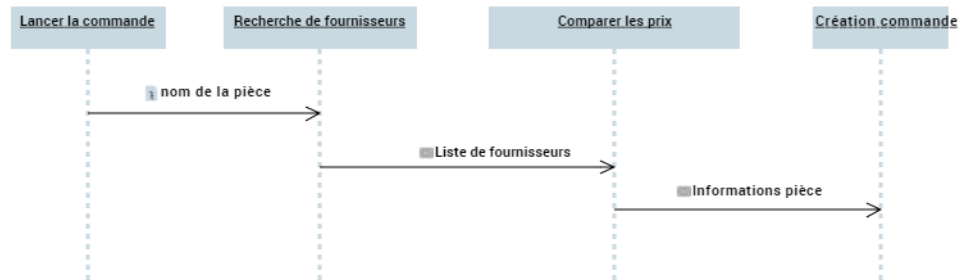


This diagram describes the operation of the "Order Unreferenced Parts" use case :

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.

## Creating a flow scenario sequence diagram

To create an application environment scenario sequence:

1. Open the **Diagram** page of the application environment that interests you and click **New**.
2. In the dialog box, 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 an IT service, the service 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 IT Service instance for example:

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

## Message instance

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

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

To create a message instance:

1. Click the reel 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 THE STRUCTURE AND SERVICES OF AN APPLICATION

## 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.*
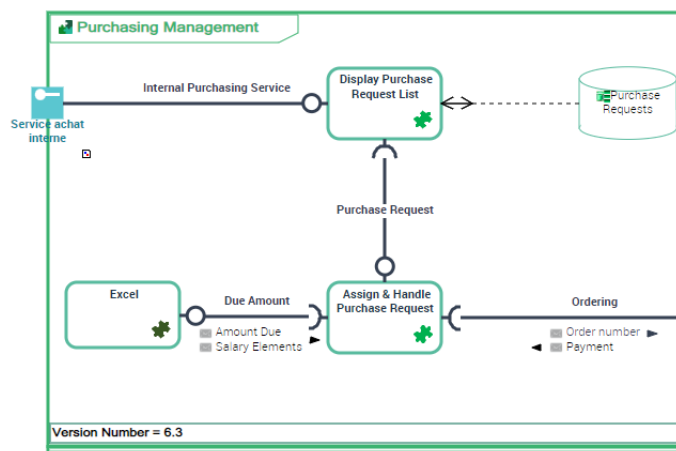


The "Purchase Request Management" application uses two IT Services: "Display purchase request list" and "Assign and handle purchase request". The IT Service "Assign and handle DA" uses the Excel microservice.

### Creating an Application Structure Diagram

To create an Application Structure Diagram, for example:

1. Select the Application that interests you and click **Create Diagram**.
2. In the Create a diagram window, select **Structured diagram > Internal Architecture**.
   The internal architecture diagram appears. The frame of the Application described appears in the diagram.

## 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, it relates to "Display purchase request list" and "Assign and handle purchase request" services.

  &#x1F4D6; *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.*

- *Microservices* which represent the services used independent from the application.

  In the example, it relates to the Excel application.

  &#x1F4D6; *A microservice 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.*

- *Service interactions* between components.

  &#x1F4D6; *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

- *physical data stores* used by the application.

  ☛ *For more details, see Managing Data.*

## Adding an IT Service to an application structure diagram

To describe that an application uses an IT Service, go to:

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

# Describing an IT Service with Hopex IT Architecture

&#x1F4D6; *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.*

## IT Service diagrams

An *IT Service* is described by several types of diagrams:

- an *IT Service structure diagram* is used to represent the service interactions between the IT Service components under the form of service interfaces.

  ☞ *For more details, see Using IT Service Structure Diagram.*

- a *scenario of IT Service Flows* presents the application flows exchanged between the described IT services or microservices used by this IT Service. A scenario can represent a particular use case of the IT service or more globally all the flows exchanged within this IT service.

  ☞ *For more details, see Using a Scenario of Application Flows Diagram.*

## Accessing the list of IT services

To access the list of IT Services from the **Inventories** navigation menu:

☐ Select **Software > IT Services**.
The list of IT services appears in the edit area.

## IT Service properties

The complete description of an IT Service is accessed from its properties pages.

The **Characteristics** properties page for an IT Service provides access to:

- its **Owner**, by default during creation of the IT service, the current library.
- its **Name**,
- the **Type**,
- the **Visibility**,
- the **Review Status**,
- the text of its **Description**
- The **Functional scope** section is used to describe:
  - The *business capabilities* covered by the IT Service,

    📖 *A business capability represents a specific ability that an organization possesses or needs to develop to deliver a particular business outcome.*

    ☞ *For more details on business capabilities, see Describing Business Capabilities with Hopex IT Architecture.*

  - The *Implemented Functionalities* that are fulfilled by the IT service.

    📖 *A functionality is a service required to perform a work. This functionality is generally necessary within an activity in order to execute*

*a specific operation. If it is a software functionality, it can be provided by an application.*

☛ *For more details on functionalities, see Describing a Functionality Map with Hopex IT Architecture.*

☛ *For more details on fulfillments, see Using fulfillment mechanisms.*

- the **Use Cases** section, see  Creating an application Use Case Diagram.
- The **Responsibility** section relates the person(s) responsible for the IT service
  - Software Designer
  - Local Application Owner

    ☛ *For more details on these roles, see Business Roles of Hopex IT Architecture..*

- the **Technologies** section provides access to the list of *Software Technologies* used by the IT Service.

  📖 *A technology is a definition or format that has been approved by a standards organization, or is accepted as a standard by the industry.*

  ☛ *For more details on software technologies, see Describing a Software Technology.*

- the **Risks** section presents the risks associated with the application, see Describing an Application Environment with Hopex IT Architecture.

With **Hopex IT Architecture** an IT service is described by other property pages, see Hopex IT Architecture properties pages content

## Using IT Service Structure Diagram

☛ *For more details on use of a structure diagram, see Application structure diagram*

With **Hopex IT Architecture**, the components of an IT Service can be described by an IT Service structure diagram.

An IT Service Structure Diagram includes:

- IT services,
- Microservices
- Physical data stores; see Managing Data.
- access, request and service points; Creating a Service Point or a Request Point.
- *Service interactions* between components

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

# Describing a microservice with Hopex IT Architecture

📖 *A microservice 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.*

## Microservice diagrams

A *microservice* is described by several types of diagram:

- A *microservice structure diagram* is used to represent the interactions between microservice components based on service interfaces formalism.

  ☛ *For more details, see* *Using a Microservice Structure Diagram.*

- A *microservice flow scenario* presents the flows exchanged between the microservice elements in a given context.

  ☛ *For more details, see* *Using a Scenario of Application Flows Diagram.*

- A *microservice deployment architecture* used to represent technical elements that support the microservice.

  ☛ *For more details, see* *Describing an Application Deployment Architecture.*

## Accessing the list of microservices

To access the list of microservices from the **Inventories** navigation menu:

- ☐ Select **Software > MicroServices**.
  The list of microservices appears in the edit area.

## Microservice properties with Hopex IT Architecture

A Microservice can be described from its properties pages.

The **Characteristics** property page of a microservice provides access to:

- Its **Owner** which is the current library by default,
- its **Name**,
- the **Review Status**,
- the text of its **Description**
- The **Functional scope** section is used to describe:
  - The *Business capabilities* covered by the microservice,

    📖 *A business capability represents a specific ability that an organization possesses or needs to develop to deliver a particular business outcome.*

    ☛ *For more details on business capabilities, see* *Describing Business Capabilities with Hopex IT Architecture.*

  - The *Implemented Functionalities* that are fulfilled by the microservice.

    📖 *A functionality is a service required to perform a work. This functionality is generally necessary within an activity in order to execute*

*a specific operation. If it is a software functionality, it can be provided by an application.*

☞ *For more details on functionalities, see Describing a Functionality Map with Hopex IT Architecture.*

☞ *For more details on fulfillments, see Using fulfillment mechanisms.*

- the **Use Cases** section, see  Creating an application Use Case Diagram.
- The **Responsibility** section relates the person(s) responsible for the IT service
  - Software Designer
  - Local Application Owner

    ☞ *For more details on these roles, see Business Roles of Hopex IT Architecture..*

- The **Technologies** section provides access to the list of *Software Technologies* used by the microservices.

  📖 *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.*

  ☞ *For more details on software technologies, see Describing a Software Technology.*

- the **Risks** section presents the risks associated with the application, see Describing an Application Environment with Hopex IT Architecture.

With **Hopex IT Architecture** a microservice is described by other property pages, see Hopex IT Architecture properties pages content

## Using a Microservice Structure Diagram

☞ *For more details on use of a structure diagram, see Application structure diagram*

With **Hopex IT Architecture**, the components of a microservice can be described by a microservice structure diagram.

A Microservice Structure Diagram includes:

- IT services,
- 
- Physical data stores; see Managing Data.
- access, request and service points; Creating a Service Point or a Request Point.
- *Service interactions* between components

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

# Creating an application Use Case Diagram

☞ *To access UML functions, you must be connected with the **Solution Architect** profile or the **Solution Architecture Functional Administrator** profile.*

A *use case* diagram enables description of service 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.*
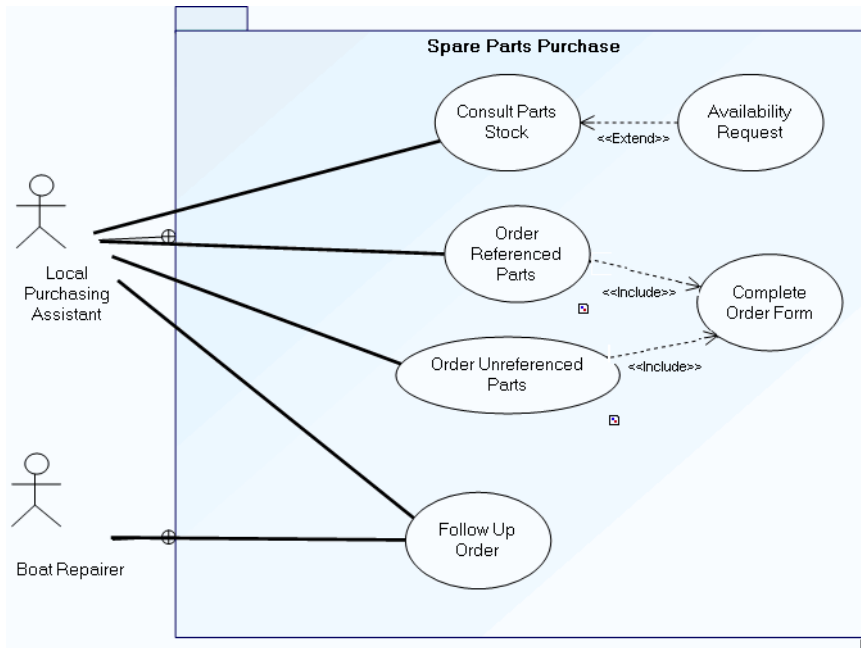


The system is used to consult parts in stock and to order new spare parts.

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.

> ☞ *For more details on use case diagrams, see* Use Case Diagram.

To create a use case diagram from an application, for example:

1. Open the **Characteristics** page of the appropriate object and expand the **Use Case** section.
2. Open the **Diagrams** property page of the use case you are interested in.
3. Click **Create a diagram** and select **Use Case Diagram**.
   The diagram appears in the edit area.

# DESCRIBING SYSTEM 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 diagram example*

The diagram below represents purchase request processing.

- A product search is carried out from the referenced products repository.
- 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.
- 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, an order is sent to the supplier and the process ends.
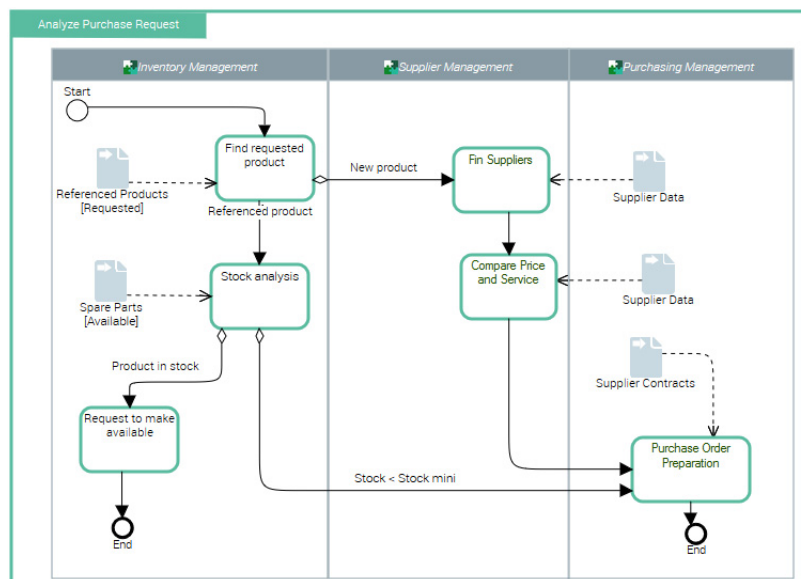


## Managing System Processes with Hopex IT Architecture

&#x1F4D6; *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 from the **Inventories** navigation menu:

- ☐ Select **Software > System processes**.
  The list of system processes appears.

## Creating a system process diagram

The system process diagram uses notation proposed by BPMN standard. The system process algorithm can be expressed by sequencing of tasks and decisions.

A system process diagram can be created and updated in tabular input mode.

☛

☛ *For more information on using tabular entry, see the "Diagrams in Tabular Entry Mode" in the* ***Hopex Common Features*** *guide.*

To create a system process diagram:

1. Select the system process that interests you and click **Create Diagram**.
   The Diagram type selection dialog box opens.
2. Click **System process diagram**.
   The diagram opens in the edit area. The frame of the system process described appears in the diagram.

### *Creating a Task*

*Tasks* correspond to process steps.

📖 *A task is an elementary step that is included within a system process. A task is used when the work in the system process is not broken down to a finer level of the process. Generally, an end-user and/ or an IT service are used to perform the task when it is executed.*

To create a task:

1. In the diagram insert toolbar, click the **Task** button then click in the diagram.
2. Enter the task name and click **OK**.
   The task appears in the diagram.

### *Message flows*

*Message flows* represent exchanges between the system process and the exterior.

📖 *An application flow represents the use of a flow between two agents (e.g., applications) in a usage context (represented by a*

*scenario of flows). An application flow is based on a flow, which represents the reference flow in context.*
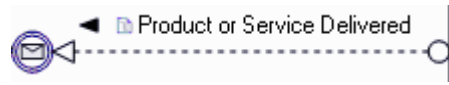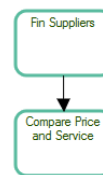


☛ *A message flow can be linked to an event of message type.*

### Sequence flows

Organization of tasks in the system process is represented by *sequence flows* between tasks.

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



☛

☛ *For more information on managing sequence flows, see "Describing Operations Sequence" chapter in the Hopex Business Process Analysis guide.*

### Events

*Events* represent facts occurring during process execution.

📖 *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.*

An example is the start or end of the system process.



*Start*



*Final*

The event can also be sending or receiving a message flow.

### *Gateways*

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

☛

☛ *For more information on managing gateways, see "Using gateways" chapter in  guide Hopex Business Process Analysis.*

### *Creating a participant in a System Process Diagram*

In a system process diagram, a participant enables grouping of tasks assigned to an application or service.

To create a participant:

1. In the diagram insert toolbar, click the arrow at the right of the **Participant (Application)** button.
2. In the list proposed, select for example **Application Participant** and click in the diagram.
   The participant creation dialog box appears.
3. Click the down arrow of the  **Application** field and select the applications that interest you.
4. Click **OK**.
   The participant created appears in the diagram with a header containing the name of an assigned application.

      ☛ *To place a participant with assignment as yet unknown, select the **Participant** icon.*

To assign a task to a participant:

   ☐ place the task within the frame of the participant.

## Specifying the behavior of  a task in a System Process

Conformément à la norme BPMN, un processus peut avoir des comportements différents.

Avec **Hopex Business Process Analysis**, ces comportements sont disponibles pour les processus, les opérations, les processus applicatifs et les tâches.

## Les comportements

Les comportements proposés sont :

- **Transaction** : une transaction est un ensemble d'activités coordonnées entre elles en vue d'obtenir un résultat cohérent et vérifiable.
- **Boucle** : une boucle est une étape d'un processus qui est répétée tant qu'une condition est vérifiée.
  - "Faire tant que" : la condition est évaluée avant la première exécution.
  - "Faire jusqu'à ce que" : la condition est évaluée après la première exécution. Dans ce cas, le processus est toujours réalisé au moins une fois.

  Le prédicat permet de spécifier la condition d'exécution de la boucle.
- **Ad hoc** : les étapes d'un processus de ce type ne sont pas contrôlées, ni ordonnées. L'ordre de réalisation est déterminé par les participants qui exécutent le processus.
- **Multiple** : le processus est répété un nombre prédéfini de fois qui est évalué avant sa première exécution. Il est possible de spécifier le type d'exécution :
  - "Parallèle" : toutes les exécutions sont réalisées en même temps
  - "Séquentielle" : les exécutions sont réalisées les unes après les autres.
- **Compensation** : une compensation définit l'ensemble des activités qui sont réalisées après l'annulation d'une transaction pour compenser les activités réalisées durant le déroulement normal du processus.

Pour décrire, par exemple, qu'un processus est exécuté en boucle :

1. Ouvrez la page de propriétés **Caractéristiques** du processus.
2. Dans la section **Détails**, à droite du champ **Boucle**, sélectionnez le type de boucle correspondant au comportement du processus et ajoutez le texte de la condition.
   La forme du processus est modifiée pour faire apparaître le symbole de la boucle.



## Type de tâche

Pour préciser le type d'une tâche :

1. Ouvrez la page de propriétés **Caractéristiques** du processus.

**2.** Dans la section **Détails**, cliquez sur la flèche à droite du champ **Type de tâche**.
La liste des types de tâche s'affiche.

- **Réception :** tâche élémentaire qui attend l'arrivée d'un message en provenance d'un participant externe au processus. Quand le message a été reçu, la tâche est terminée.
- **Emission** : tâche qui envoie un message à un participant externe au processus. Quand le message a été envoyé, la tâche est terminée.
- **Manuelle** : tâche réalisée sans l'aide d'un moteur d'exécution automatique d'un processus ou d'une application informatique.
- **Règle métier** : tâche d'exécution d'une règle métier qui dispose d'un moteur de règles qui traite les données en entrée et retourne en sortie les résultats du calcul effectué.
- **Script** : tâche réalisée par un moteur d'exécution de processus. Le concepteur définit un script dans un langage que le moteur est capable d'interpréter. Quand la tâche est prête à démarrer, le moteur exécute le script. La tâche est terminée quand l'exécution du script est terminée.

☛ *La forme du processus est modifiée pour faire apparaître le symbole associé au type de tâche.*

# Modeling Tasks of a System Process

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.

## 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.*

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

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

**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.
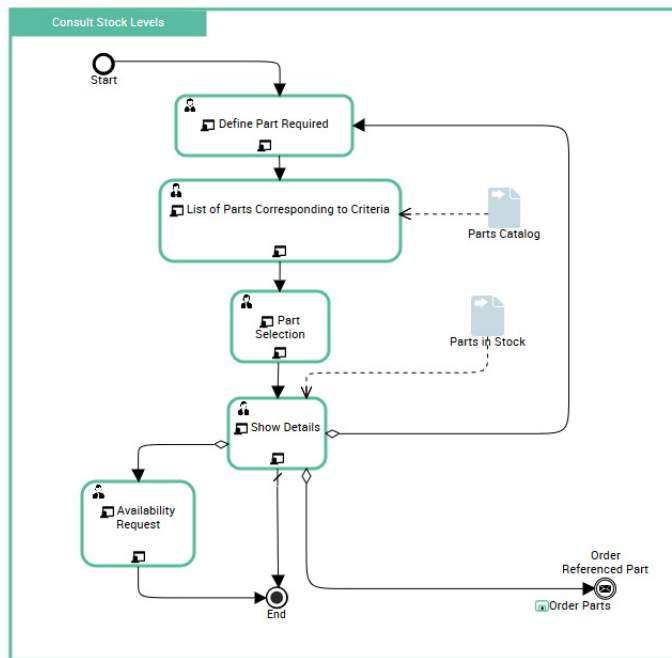


Consulting stock levels begins by display of a screen enabling identification of the required part. The list of parts found in the catalog is presented in the next screen.

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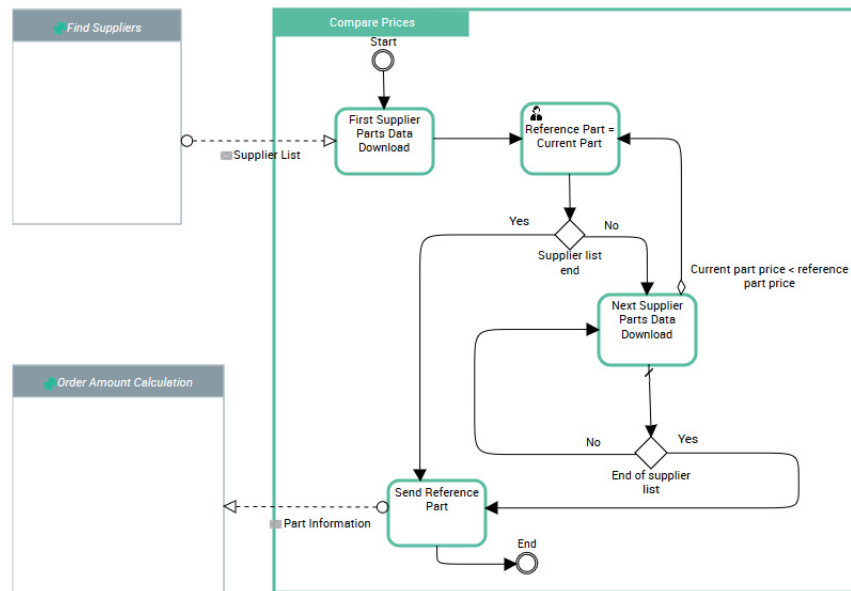
This diagram describes the algorithm of the "Compare Prices" service, which should return the reference of the lowest-priced part.

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.

# MANAGING DATA

*Data stores* are used in architecture diagrams to represent data that must be stored to be share between components.

> 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.

A *Data store* can be supported by  different object types:

- A *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.

- a *file structure*

    > A file structure represents a file folder or a single file used in the technical architecture of an application.

- A *NOSQL data domain*

    > NoSQL data domain represents a set of data stored in a NOSQL database management system and used in the technical architecture of an application.

- A *Relational Schema*

    > Relational schema represents a set schema stored in a database management system and used in the technical architecture of an application.

---

## Using Data Stores

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

> 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

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 scenario of sequences or a scenario of flows, 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.*

   *Sequences scenario that describe the chronology of the flows exchanged in different usage scenarios of the described object.*

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<br>- of application,<br>- of application system,<br>- IT service,<br>- microservices. |
| Application data stores | Scenario sequence diagrams<br>- of application,<br>- of application system,<br>- IT service,<br>- microservices,<br>Scenario of flows diagrams<br>- of application,<br>- of application system,<br>- IT service,<br>- microservices. |

## 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** and select the **Object Type** that supports the data store you describe.

☞ *For more information on proposed object types see* *Access Data Stores supports.*

3. Click in the frame of the described application system.
4. Depending on the **Object type**, select then the object that interests you.
5. Click **OK**.
The local physical data store appears in the diagram with the name of the associated object.

## Creating a 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** and select the **Object Type** that supports the data store you describe.

☞ *For more information on proposed object types see* *Access Data Stores supports.*

3. Click at the edge of the frame of the described application system.
4. Depending on the **Object type**, select then the object that interests you.
5. Click **OK**.
The local physical data store appears in the diagram with the name of the associated object.

## 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.*

# Access Data Stores supports

A *Data store* can be supported by  different object types:

- A *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.*

- a *file structure*

  📖 *A file structure represents a file folder or a single file used in the technical architecture of an application.*

- A *NOSQL data domain*

  📖 *NoSQL data domain represents a set of data stored in a NOSQL database management system and used in the technical architecture of an application.*

- A *Relational Schema*

  📖 *Relational schema represents a set schema stored in a database management system and used in the technical architecture of an application.*

## Accessing to data areas with Hopex IT Architecture

📖 *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 areas, see **Hopex Data Architecture** guide.*

To access the list of *data domains* from the **Inventories** navigation menu:

1. Select **Data > Data Dictionaries**.
   The tree of data dictionaries appears.
2. Expand the **Package** folder.
3. Expand the folder of the package that interests you.
4. Expand the **Data Dictionaries** folder.
   The list of package data domains appears.

## Accessing the list of file structures with Hopex IT Architecture

📖 *A file structure represents a file folder or a single file used in the technical architecture of an application.*

To access the list of *file structures* from the **Inventories** navigation menu:

1. Select **Data > Data Dictionaries**.
   The tree of data dictionaries appears.
2. Expand the **NoSQL Building Block Catalog** folder.
   The list of *File Structures* appears.

### Accessing to NoSQL data domains with Hopex IT Architecture

📖 *NoSQL data domain represents a set of data stored in a NOSQL database management system and used in the technical architecture of an application.*

To access the list of *NoSQL data domains* from the **Inventories** navigation menu:

1. Select **Data > Data Dictionaries**.
   The tree of data dictionaries appears.
2. Expand the **NoSQL Building Block Catalog** folder.
   The list *NoSQL data domains* appears.


### Accessing the list of relational schemes with Hopex IT Architecture

📖 *Relational schema represents a set schema stored in a database management system and used in the technical architecture of an application.*

To access the list of *relational schemes* from the **Inventories** navigation menu:

1. Select **Data > Data Dictionaries**.
   The tree of data dictionaries appears.
2. Expand the **Database** folder.
3. Expand the folder of the database that interests you.
4. Expand the **RDB Physical Structure** folder.
   The list of *relational schemes* of the database appears.

# MODELING TECHNICAL ARCHITECTURES

A deployment architecture allows you to describe the overview elements that must be deployed to implement architecture an application: *Application Deployment Architecture*, *Deployable Data Package* as well as *Package connections* used for data exchange.

Several viewpoints are proposed in **Hopex IT Architecture**:

- ✓ The *Application Deployment Environment* used to represent of the deployments of partner applications as well as microservices identified around the subject application
- ✓ The *Application System Deployment Architecture* used to represent the set of Application Deployment Architectures that must be coordinated to cover required dependencies between them.
- ✓ The *Application Deployment Architecture* used to represent the deployment packages list and the communication lines.

The following points are covered here:

- ✓ Describing an Application Deployment Environment.
- ✓ Describing an Application System Deployment Architecture.
- ✓ Describing an Application Deployment Architecture.
- ✓ Deployment Architecture Templates
- ✓ Describing Software Technologies.
- ✓ Using Cloud Services.

# DESCRIBING AN APPLICATION DEPLOYMENT ARCHITECTURE

📖 *An application deployment architecture describes one possible deployment configuration of an application. It contains the deployment architectures to be hosted, recommends hosting architectures and identifies required communication techniques (communication protocols and port numbers) they use to communicate with each other. . An application may have several deployment architectures (E.g.: autonomous installation, horizontal or vertical deployment, etc.)*

## Accessing the application deployment architectures

To access the list of application deployment architectures from the **Applications** navigation menu:

❱ Open the **Application Deployment** page of the concerned application. The list of application deployment architectures appears.

## Describing an Application Deployment Architecture and its diagram

An application deployment architecture allows you to describe the overview elements that must be deployed to implement an application architecture: *Deployable Application modules*, *Deployable Data Modules* as well as *Package connections* used for data exchange.
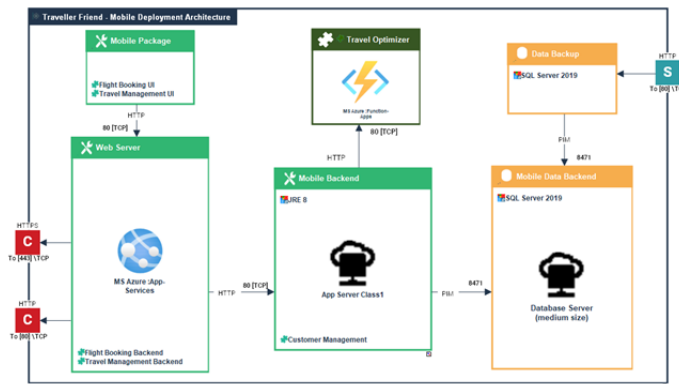
An deployment architecture diagram includes the following elements:

- *Deployable Application Packages*,

  📖 *A deployable application package is a split of application code according to technical criteria for hosting purpose. For example, it may be N tiers, Front End/Back End/… or GUI/Business Logic/Database etc... Each deployable application package is associated to required technologies (for running) and can host code for several IT services. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

- *Deployable Data Packages*,

  📖 *A deployable data package represents a data part of an application deployment that must be hosted and accessed by application services (code) to run. Each deployable data package is associated to required technologies (for data hosting and access) and can host several data structures. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model). Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

  ☛ *For more details on deployable data packages, see Adding a deployable application package in an application deployment architecture diagram.*

- *microservices*,

  📖 *A microservice 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.*

- *Technical Server Port* and *Technical Client Port*,

  📖 *A 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 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.).*

  ☛ *For more details on technical ports, see Adding technical ports.*

- *Package Connections*.

  📖 *A package connection represents a connection requirement between deployable packages. A client (package) requires a connection to a server (package). The connection is defined by a Connection Type and can be characterized with technical flow measures.*

  ☛ *You can create a Application Deployment Architecture by creating an Application Deployment Architecture diagram directly from the application that interests you.*

  ☛ *For more details on technical communications, see Describing package connections.*

## Creating an Application Deployment Architecture

To create an application deployment architecture from the **Applications** navigation menu:

1. Open the **Application Deployment** page of the concerned application. The list of application deployment architectures appears.

**2.** Click **New**.
A dialog box opens to select an application deployment architectures template.

☛ *For more details on application deployment templates, see Deployment Architecture Templates.*

**3.** Select **Empty application deployment architectures template** and click **OK**.

☛ *For more details on application deployment templates, see Deployment Architecture Templates.*

A new application deployment architectures is created with its diagram.

☛ *You can also create a Application Deployment Architecture by creating an Application Deployment Architecture diagram directly from the application that interests you.*

# Using an application deployment architecture diagram

To create an application deployment architecture diagram from an existing application deployment architecture diagram:

**1.** Open the **Application Deployment** page of the concerned application.
**2.** Right-click the application deployment architecture that interests you and select **Create diagram**.
**3.** In the dialog box, select **Deployment Architecture**.
The diagram opens in the edit area. The described deployment architecture components appears in the diagram.

☛ *When an application deployment architecture is created, an application deployment architecture diagram automatically created.*

## Adding a deployable application package in an application deployment architecture diagram

### Adding a deployable application package

📖 *A deployable application package is a split of application code according to technical criteria for hosting purpose. For example, it may be N tiers, Front End/Back End/… or GUI/Business Logic/Database etc... Each deployable application package is associated to required technologies (for running) and can host code for several IT services. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

To add a **deployable application package**:

**1.** In the objects toolbar of the application deployment architecture, click **Deployable Application Package** button.
**2.** Click in the described application frame.
A dialog box prompts you to choose the **Deployable Application Package** that you wish to use.
**3.** Then, create the deployable application package and click **OK**.
The deployable application package appears in the diagram.

☛ *For more details on the description of a deployable application package, see Describing a Deployable Application Package.*

### Adding a deployable data package

You add a deployable data package like a deployable application package.

            *A deployable data package represents a data part of an application deployment that must be hosted and accessed by application services (code) to run. Each deployable data package is associated to required technologies (for data hosting and access) and can host several data structures. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model). Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

## Adding technical ports

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

            *A 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 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.).*

Communication 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.*

To create a *technical client port*:

1. In the diagram objects toolbar, click the **Technical Client Port** button
2. Click on the frame of the described deployment architecture.
3. In the technical port creation dialog box, select *Network application protocols* and the *Network application connection*.
4. Click **Add**.
   The technical port appears in the diagram. The protocol name appears above the technical port.

## Describing package connections

The communications between the deployable application packages and the deployable data packages can be described by package connections. A package connection supports the *network application protocol* defined to create the communication.

            *A package connection represents a connection requirement between deployable packages. A client (package) requires a connection to a server (package). The connection is defined by a Connection Type and can be characterized with technical flow measures.*

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

To create a *package connection*:

1. In the diagram objects toolbar, click **package connection**.
2. Draw a line between the two communicating objects.
3. In the package connection creation dialog box, select *Network application protocol* and the *Connection Type*.

4. Click the **New** button.
The package connection appears in the architecture. The protocol name appears along the line.

☛ *In a package connection **Characteristics** page, the **Used Communication Format** field specifies the **Communication Format**. The selected format appears in the diagram in addition to the protocol name.*

# Describing a Deployable Application Package

📖 *A deployable application package is a split of application code according to technical criteria for hosting purpose. For example, it may be N tiers, Front End/Back End/… or GUI/Business Logic/Database etc... Each deployable application package is associated to required technologies (for running) and can host code for several IT services. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

### Defining the software technologies used by a deployable application package

📖 *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 a **Deployable Application Package**:

1. Open the **Characteristics** property page of the **Deployable Application Package** that interests you.
2. In the **Required Software Technologies** section, click **Connect**.
In the selection dialog box, select the **Software Technology** that you want to use.

The software technologies selected appear in the icon of the deployable application package.

### Defining a deployable data package components

To specify, for example, that a Cloud Service is used by a **deployable application package**:

1. Open the **Characteristics** property page of the **Deployable Application Package** that interests you.
2. Unfold the **Deployable Application Component** section.
3. In the **Prescribed Computing Device** field, click **Connect**.
In the dialog box, select the **Cloud Service** that you want to use.

☛ *For more details on Cloud Services, see Using Cloud Services.*

# DESCRIBING AN APPLICATION DEPLOYMENT ENVIRONMENT

The Application Deployment Environment is considered as the center of the integration and all required deployments of partner applications or microservices.

📖 *An application deployment environment diagram represents the subject deployment application architectures, the partner deployment application architectures and the partners microservices, as well as the techniques used for their communications.*

## Accessing the list of application deployment environments

To access the list of application deployment environments from the **Application** navigation menu:

1. Open the **Deployment Architecture** page of the concerned application.
   The list of application deployment architectures appears.
2. Open the **Environment** page of the concerned architecture deployment application.
   The list of application deployment environments appears.

## Describing an Application Deployment Environment

An application deployment environment is described by an application deployment environment diagram.
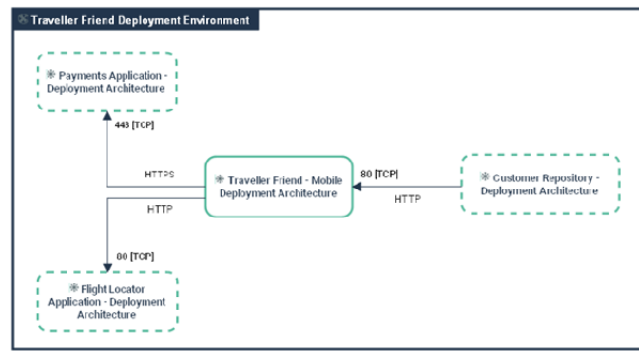


An application deployment environment diagram includes the following elements:

- *Subject application deployment architectures* and *Partner application deployment architectures*,

  📖 *An application deployment architecture describes one possible deployment configuration of an application. It contains the deployment architectures to be hosted, recommends hosting architectures and identifies required communication techniques (communication protocols and port numbers) they use to communicate with each other. . An*

*application may have several deployment architectures (E.g.: autonomous installation, horizontal or vertical deployment, etc.)*

- des *microservices partenaires*,

  📖 *A microservice 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.*

- *Package Connections*.

  📖 *A package connection represents a connection requirement between deployable packages. A client (package) requires a connection to a server (package). The connection is defined by a Connection Type and can be characterized with technical flow measures.*

## Creating an Application Deployment Environment

To create an Application Deployment Environment from **Application** navigation menu:

1. Open the **Deployment Architecture** page of the concerned application. The list of application deployment architectures appears.
2. Open the **Environment** page of the concerned architecture deployment application.
3. Click **New** button.
   The new application deployment environment appears in the list.

# Using an Application Deployment Environment Diagram

To create an Application Deployment Environment Diagram from **Application** navigation menu:

1. Open the **Deployment Architecture** page of the concerned application. The list of application deployment architectures appears.
2. Open the **Environment** page of the concerned architecture deployment application.
3. Select the application deployment environment that interests you and click **Create Diagram** button.
4. In the dialog box, select **Application Deployment Architecture Diagram**.
   The application deployment environment diagram opens in the edit area. The *Subject application deployment architecture* is placed in the center of the frame.

# DESCRIBING AN APPLICATION SYSTEM DEPLOYMENT ARCHITECTURE

The Application System Deployment Architecture consists of the set of Application Deployment Architectures that must be coordinated to cover required dependencies between them.

    📖 *An application system deployment architecture describes one of the configurations possible for deploying an application system. It contains the deployment architectures of application components and specifies the communication protocols (and port numbers) they use to communicate with each other.*

## Accessing the list of application system deployment architectures

To access the list of application system deployment architectures from the **Application Systems** navigation menu:

❱ Open the **Deployment Architecture** page of the concerned application system.
The list of application system deployment architectures appears.

## Describing an Application System Deployment Architecture

An application system deployment architecture is described by an application system deployment  architecture diagram composed of the following elements:



An application system deployment architecture diagram includes the following elements:

- *Application Deployment Architecture*,

    📖 *An application deployment architecture describes one possible deployment configuration of an application. It contains the deployment architectures to be hosted, recommends hosting architectures and*

*identifies required communication techniques (communication protocols and port numbers) they use to communicate with each other. . An application may have several deployment architectures (E.g.: autonomous installation, horizontal or vertical deployment, etc.)*

☛ *For more details on application deployment architectures, see Describing an Application Deployment Architecture.*

- *Application System Deployment Architectures*,

  📖 *An application system deployment architecture describes one of the configurations possible for deploying an application system. It contains the deployment architectures of application components and specifies the communication protocols (and port numbers) they use to communicate with each other.*

  ☛ *For more details on application system deployment architectures, see Describing an Application System Deployment Architecture.*

- *microservices*,

  📖 *A microservice 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.*

- *Deployable Data Packages*,

  📖 *A deployable data package represents a data part of an application deployment that must be hosted and accessed by application services (code) to run. Each deployable data package is associated to required technologies (for data hosting and access) and can host several data structures. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model). Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

  ☛ *For more details on deployable data packages, see Describing an Application System Deployment Architecture.*

- *Package Connections*.

  📖 *A package connection represents a connection requirement between deployable packages. A client (package) requires a connection to a server (package). The connection is defined by a Connection Type and can be characterized with technical flow measures.*

- *Technical Server Port* and *Technical Client Port*,

  📖 *A 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 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.).*

  ☛ *You can create an application system deployment architecture diagram is the same way than an application deployment architecture diagram. For further details, see .For further details, see Using an application deployment architecture diagram.*

## Properties of an application system deployment architecture

The complete description of an application system deployment architecture can be accessed from its property pages.

The **Components** property page of an application system deployment architecture provides access to:

- its **Name**,
- Its **Owner**, by default the application specified when it was created.
- the text of its description.

With **Hopex IT Architecture** an application system deployment architecture is described by other property pages.

The **Components** page that enables access to the described architecture components.

> ☛ *For more information on the components of an application system deployment architecture diagram, see Describing an Application System Deployment Architecture.*

- The **Deployment Architecture** enables access to the following tabs:
  - *Application System Deployment Architecture*,

    > 📖 *An application system deployment architecture describes one of the configurations possible for deploying an application system. It contains the deployment architectures of application components and specifies the communication protocols (and port numbers) they use to communicate with each other.*

  - *Deployment Architecture*, to access to the list of *application deployment architectures*,

    > 📖 *An application deployment architecture describes one possible deployment configuration of an application. It contains the deployment architectures to be hosted, recommends hosting architectures and identifies required communication techniques (communication protocols and port numbers) they use to communicate with each other. . An application may have several deployment architectures (E.g.: autonomous installation, horizontal or vertical deployment, etc.)*

    > ☛ *For more details on application deployment architectures, see Describing an Application Deployment Architecture.*

  - *Owned Microservice Deployment*, to access the list of *microservices*,

    > 📖 *A microservice 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.*

- The **Data Packages** section provides access to the list of *Deployable data packages*,

  > 📖 *A deployable data package represents a data part of an application deployment that must be hosted and accessed by application services (code) to run. Each deployable data package is associated to required technologies (for data hosting and access) and can host several data structures. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model). Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

  > ☛ *For more details on deployable data packages, see Adding a deployable application package in an application deployment architecture diagram.*

- The **Deployment Connections** section provides access to the list of *connection packages*.

  > 📖 *A package connection represents a connection requirement between deployable packages. A client (package) requires a connection*

to a server (package). The connection is defined by a Connection Type and can be characterized with technical flow measures.

☞ *For more details on connection packages, see* Describing package connections.

- The **Technical Ports** section enables access to the following tabs:
  - **Server Port**

    📖 *A 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.).*

  - **Client Port**

    📖 *A 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.).*

    ☞ *For more details on technical ports, see* Adding technical ports.

The **Reports** page, used to access the different reports available on the described application system deployment architecture.

# DEPLOYMENT ARCHITECTURE TEMPLATES

*Deployment architecture templates* are used to simplify the creation of your deployment architecture. The new application deployment architecture components are automatically created using the deployment architecture template components.

Then, the new deployment architecture can be updated or modified.

Some *deployment architecture templates* are provided with the solution.

## Accessing the list of deployment architecture templates

To access the list *Deployment architecture templates* of a repository:

❱ From the **Administration** navigation menu, select **Templates > Deployment Architectures**.
The list of deployment architecture templates appears.

## Describing an Application Deployment Template

### Components of an Application Deployment Template

An application deployment template is described by an application deployment template diagram composed of the following elements:

- *Application Deployment Templates*, used to create the deployable application packages of a new application deployment architecture.

  📖 *A deployable application package is a split of application code according to technical criteria for hosting purpose. For example, it may be N tiers, Front End/Back End/… or GUI/Business Logic/Database etc... Each deployable application package is associated to required technologies (for running) and can host code for several IT services. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

  ☛ *For more details on deployable Application packages, see Adding a deployable application package in an application deployment architecture diagram.*

- *Data Deployment Templates*, used to create the deployable data packages of a new application deployment architecture.

  📖 *A deployable data package represents a data part of an application deployment that must be hosted and accessed by application services (code) to run. Each deployable data package is associated to required technologies (for data hosting and access) and can host several data structures. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model). Architect can also*

*prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

☛ *For more details on deployable data packages, see Adding a deployable application package in an application deployment architecture diagram.*

- *microservices,*

  📖 *A microservice 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 deployable microservices, see Describing a microservice with Hopex IT Architecture,*

- *Technical Server Port and Technical Client Port,*

  📖 *A 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 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.).*

  ☛ *For more details on technical ports, see Adding technical ports.*

- *Package Connections.*

  📖 *A package connection represents a connection requirement between deployable packages. A client (package) requires a connection to a server (package). The connection is defined by a Connection Type and can be characterized with technical flow measures.*

  ☛ *You can create a Application Deployment Architecture by creating an Application Deployment Architecture diagram directly from the application that interests you.*

  ☛ *For more details on Package Connections, see Describing package connections.*

## Creating an Application Deployment Template

To access the list of the *application deployment templates* of a repository:

1. From the **Administration** navigation menu,, select **Templates > Deployment Architectures**.
   The list of application deployment Models is displayed.
2. Click **New**.
   The application deployment template appears in the list.

   ☛ *For more details on the creation of application deployment diagrams, see Using an application deployment architecture diagram.*

# Presentation of standard Deployment Architecture Templates

Deployment architecture templates are provided to simplify the creation of your application deployment architectures.

## "3 Tiers Architecture (RDBMS)" Application deployment template
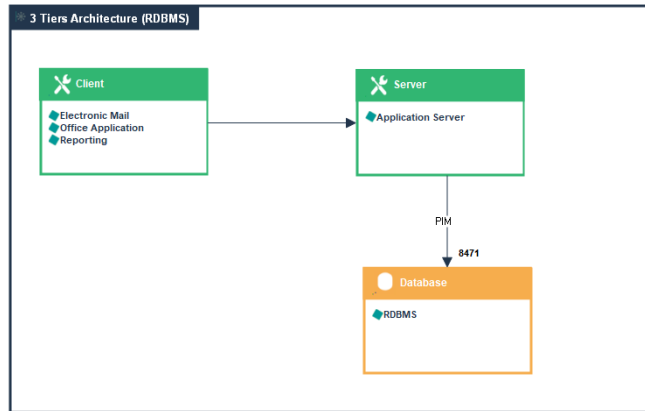


*Diagram of the "3 Tiers Architecture (RDBMS)" Application deployment template*

## "Mobile Application Architecture" Application deployment template
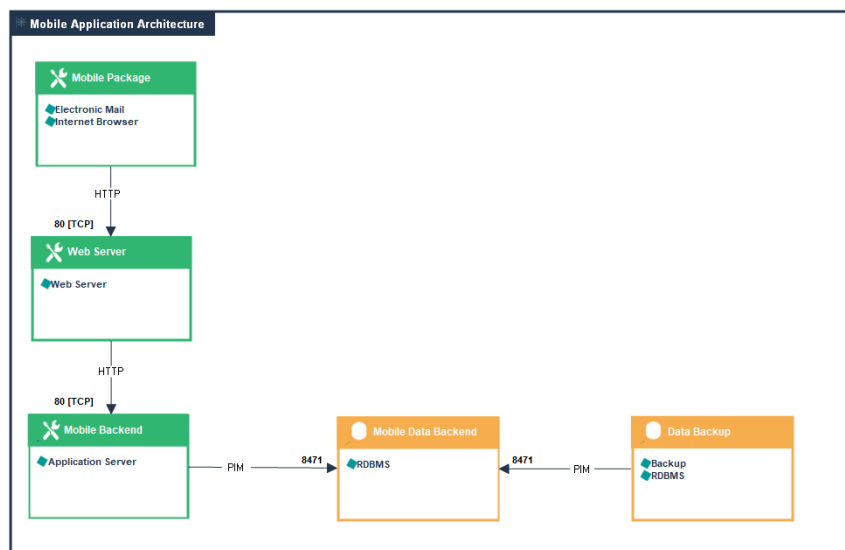


*Diagram of the "Mobile Application deployment" deployment architecture template*

**"Standard Web Application Architecture" Application deployment template**
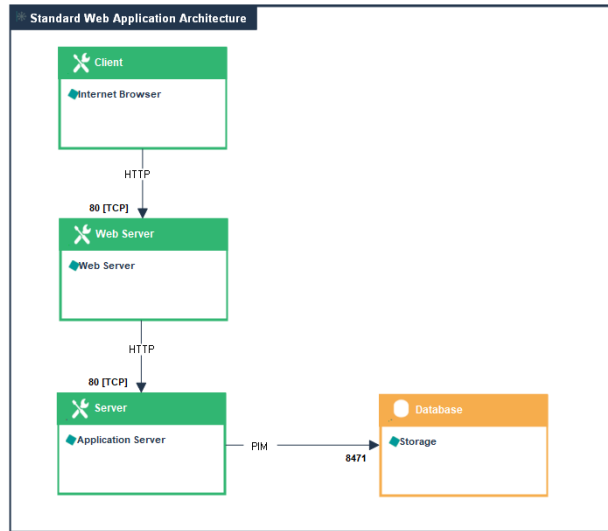


*Diagram of the "Standard Web Application Architecture"" Application deployment template*

## Using an Application Deployment Template

To create an application deployment architecture from a *deployment architecture template*:

1. Open the **Deployment Architecture** page of the concerned application.
2. Select the **Application Deployment Template**.

☛ *For more details on application deployment templates, see* *Deployment Architecture Templates.*

**3.** Click the **Next** button.
A dialog box displays the list of components of the new architecture.



**4.** (Option) In the **Create/Reuse Template** column, select the components you want to reuse.

> ☛ *Only components connected to another Application Deployment Architecture of the same application can be reused.*

**5.** Click **OK**.
The diagram opens and Then, you can modify the content of your new application deployment architecture.

> ☛ *For more details about the update of an Application deployment templates, see Using an application deployment architecture diagram.*

# DESCRIBING SOFTWARE TECHNOLOGIES

This description is based on *Software Technologies* and *Software technology Stacks*.

## Describing a Software Technology

    📖 *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.*

### Accessing the list of software technologies

To access the list of software technologies from the **Inventories** navigation menu:

❱ Select **Deployment > Software Technologies**.
The list of software technologies appears in the edit area.

### The properties of a software technology

The complete description of a software technology is accessed from its properties pages.

The **Characteristics** property page of a software technology provides access to:

- its **Name**,
- its **Owner**, by default during creation of the technology, the current library.
- its **Code**,
- its **Vendor**,
- the **Comment** text.

The **Characteristics** property page provides access to the following sections:

- **Technologies Types** that defines the concerned software technology,
- **Responsibility**,
- **Owned Realizations** which represent the list of technology capabilities covered by this software technology.

        ☛ *For more details on functionalities, see Describing functionalities with Hopex IT Architecture.*

        ☛ *For more details on realizations, see Describing the fulfillment of a Functionality.*

# Describing a Technology Stack

📖 *A software technology stack is a set of software technologies.*

📖 *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.*

## Accessing the list of technology stacks

To access the list of software technology stacks from the **Inventories** navigation menu::

❯ Select **Deployment > Software Technologies**.
The list of software technology stacks appears in the edit area.

## Properties of a software technology stack

The complete description of a software technology stack can be accessed from its property pages.

The **Characteristics** property page of a software technology stack provides access to:

- its **Name**,
- its **Owner**, by default, on creation of the software technology stack, the current library.
- its **Code**,
- the **Comment** text.

The property page provides access to the following sections:

- **Components** which provides access to the list of concerned software technologies,
- **Responsibility**,
- **Owned Realizations** which represent the list of technology capabilities covered by this software technology stack.

☞ *For more details on technology capabilities, see Describing functionalities with Hopex IT Architecture.*

☞ *For more details on realizations, see Describing the fulfillment of a Functionality.*

# USING CLOUD SERVICES

> 📖 *The Cloud Service (considered as an IoT device) can be used in a deployment architecture. The Cloud service picture appears in the frame of the corresponding deployable package.*

You can import in your **Hopex** repository *Cloud Services Catalogs* such as: Amazon (AWS), Microsoft (Azure) and Google (GCS). The imported files contain the name and the pictures of the Cloud Services proposed by the editor.

> ☛ *The Cloud Services are provided by your administrator using the module import features. For more details, see **Modules > Importing a Module** documentation.*

## Accessing the list of Cloud Services

To access the list of elements concerning the Cloud services from the **Inventories** navigation menu::

1. Select **Cloud Service Catalogs**.
2. Unfold the **Vendor Catalog** folder.
   The list of Cloud Service Catalogs appears in the edit area.
3. Expand the folder of a vendor catalog.
   The following folders provide access to different elements:
   - **Service Cloud** : provides access to the Cloud services of the catalog.
   - **Technology capability map**:  consolidates the set of technology capability maps covered  by the Cloud services of the catalog.

     > ☛ *For more details on functionalities, see Describing functionalities with Hopex IT Architecture.*

   - **Publishing Vendor**: with **Hopex**, a vendor is represented by an org-unit.

To view the functional coverage of the Cloud services connected to the technology capability map:

1. Select the technical functionality map of the catalog that interests you.
2. Open the **Reporting > Building Block Breakdown Report** property page.

3.  Select **Cloud Service** in the **Show** field.



## Cloud Service properties

The **Characteristics** property page of a Cloud service provides access to:

- its **Name**,
- the **Service Type**,
- the **Description** text,
- The **Fulfillments** section that provides access to the technology capabilities covered by the Cloud Service

  ☛ *For more details on a component fulfillment, see Describing the fulfillment of a Functionality.*

- the **Using Application Deployment Architecture** section that provides a tree of all the application architectures using the Cloud service through a deployable application package component.

  ☛ *For more details on the use of Cloud services by a deployable application package, see Describing a Deployable Application Package.*

# ALIGNING IT AND BUSINESS

The goal of this step, on a strategic level, is to check the suitability between the *business capabilities* of the enterprise and the logical architecture elements that deliver them.

This consists of the following tasks:

- ✓ Describing Logical Application Architecture
- ✓ Describing Business Capabilities with Hopex IT Architecture,
- ✓ Using Functionalities with Hopex IT Architecture,
- ✓ Using fulfillment mechanisms.

# DESCRIBING LOGICAL APPLICATION ARCHITECTURE

**Hopex IT Architecture** provides ways to define logical application architectures that represent ideal architectures. These representations make it possible to design logical structures for application architectures, to rationalize exchanges between these structures and to identify the data used. Logical application architectures can then be compared with the implemented architectures to detect gaps between the real and the ideal.

## Describing a Logical Application System with Hopex IT Architecture

A project for describing the logical architecture of an information system inventories the existing *logical application systems* and their interactions.

> 📖 *A logical application system is an assembly of other application architectures, logical applications and end users, interacting with application components to implement one or several functions.*

A *Logical Application System* can be described by two types of diagram.

- an application system structure diagram that represents the different components of the application system and their interactions.

  > ☛ *For more details on application system structure diagrams, see Describing a logical application system structure.*

- A scenario of logical application system flow diagram is used to describe the exchanges inside the described logical application system in a specific context.

  > ☛ *For more details on scenarios of flows diagrams, see Using a flow scenario sequence diagram.*

### Accessing the list of logical application systems with Hopex IT Architecture

To access the list of logical application systems form **Inventories** navigation menu:

❭ Select **Software > Logical Software Architecture** in the navigation menu.
The tree of logical application systems appears.

### Creating a Logical Application System

To create a *logical application system*:

1. From the **Inventories** navigation menu, select **Software > Logical Software Architecture**.
2. Click the **New** button.
   The **Creation of a Logical Application System** dialog box appears.
3. Enter the **Name** of your logical application system as well as its **Owner** and click **OK**.
   The new logical application system appears in the list.

## Logical Application System Properties

The **Characteristics** properties page for a logical application system provides access to:

- its **Name**,
- its **Owner**, by default, during creation of the logical application system, the current library.
- the text of its **Description**.

With **Hopex IT Architecture**, a logical application system is described by the following pages:

- the **Properties** page, used to specify the properties that appear in the diagrams at the bottom of the described object frame.
- the **Component** page provides access to the list of application system components described in its different diagrams as well as the communications that exist between them.

> ☞ *For more information on the components of a logical application system, see Describing a logical application system structure.*

- the **Implementation** page is used to specify the logical or physical elements that implement the described logical application system.

> ☞ *For more details on other property pages proposed by **Hopex IT Architecture**, see Hopex IT Architecture properties pages content.*

## Describing a logical application system structure

With **Hopex IT Architecture**, the components of a logical application system and their exchanges are described in a **logical application system structure diagram**.

```
The logical application system structure diagram, for
managing "Internet Purchase Requests", presents different
logical applications, access to a logical database as well
as service and request points for "Book" or "Order".
```
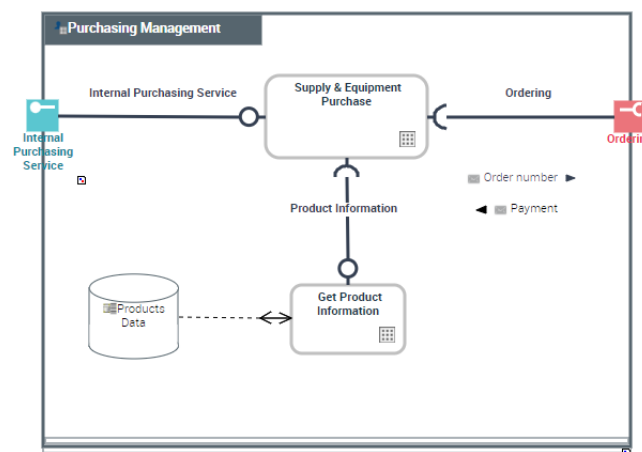


*" Purchasing request Management" Logical application system structure diagram*

A logical application system structure diagram includes the following elements:

- *end users*

  &#128213; *The end user represents an organizational unit interacting at the boundaries of an application system or a logical application system.*

  &#9758; *For more details on adding end users, see Adding an end user to the logical application system structure diagram.*

- *Logical Application System Components* and *Logical Application Components*

  &#128213; *A logical application is a set of application functionalities that is independent of a particular implementation. For example, the classification of all purchase request processing applications implemented in an enterprise.*

  &#9758; *For more details on adding applications, see Adding a logical application to a logical application system structure diagram.*

- *Service interactions* between the components representing requests for services

  &#128213; *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

  &#9758; *For more details on interaction services between logical application system components, see Managing Service Interactions.*

- *service points*

  &#128213; *A service point is a point of exchange by which an agent offers a service to potential customers.*

- *request points*

  &#128213; *A request point is a point of exchange by which an agent requests a service from potential suppliers.*

  &#9758; *For more information on access points, see Describing Service and Request Points.*

### Adding an end user to the logical application system structure diagram

To create an **end user**:

1. In the objects toolbar of the logical application system structure diagram, click **End User**.
2. Click in the frame of the described logical application system.
   An addition window prompts you to choose the **Object Type** that you wish to use:
3. For example, select the **Org-unit** object type.

   &#128213; *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

4. Select the org-unit that interests you and click **OK**.
   The actor appears in the diagram.

### Adding a logical application to a logical application system structure diagram

To describe that a logical application system implements a logical application:

1. In the objects toolbar of the logical application system structure diagram, click **Logical Application component** and click in the frame of the logical application system described.
   An addition dialog box prompts you to select the **Logical Application** used.
2. Select an existing logical application.
3. Click **OK**.
   The logical application appears in the diagram.

# Describing Logical Applications with Hopex IT Architecture

📖 *A logical application is a set of application functionalities that is independent of a particular implementation. For example, the classification of all purchase request processing applications implemented in an enterprise.*

## Accessing the list of logical applications with Hopex IT Architecture

To access the list of logical application form **Inventories** navigation menu:

1. Select **Software > Logical Software Architectures** and unfold tje dossier **Logical Application System** folder.
2. Unfold the logical application system tree that interests you and unfold the **Logical Application Component** folder

## Creating a logical application

To create a *logical application* from the **Inventories** navigation menu:

1. Select **Software > Logical Software Architectures**.
   The list of logical application systems appears.
2. Open the **Components** property page of the application system of your choice.
3. In the list of component types, select **Logical Application Component** and click **New**.
   The **Create a Logical Application** window appears.
4. Enter the **Name** of your logical application and click **OK**.
   The new logical application appears in the list.

### Logical Application Properties

The **Characteristics** properties page of the logical application provides access to:

- its **Name**,
- its **Owner**, by default during creation of a logical application, the current library.
- the text of its **Description**.

With **Hopex IT Architecture**, a logical application is described by the following pages:

- the **Properties** page, used to specify the properties that appear in the diagrams at the bottom of the described object frame.
- the **Implementation** page is used to specify the logical or physical elements that implement the logical application.
- The following pages are available: **Components**, **Executed Processes** as well as **Reports**.

> ☛ *For more details on other property pages proposed by Hopex IT Architecture, see Hopex IT Architecture properties pages content.*

## Logical Application System Environment Description

> 📖 *A logical application system environment presents a logical application system use context. It describes the service interactions between the logical application system and its external partners, which allows it to fulfill its mission and ensure the expected functionalities.*

A *Logical Application System Environment* can be described by two types of diagram.

- A scenario of logical application system environment flow diagram is used to describe the exchanges inside the described logical application system environment in a specific context.

  > ☛ *For more details on scenarios of flows diagrams, see Using a flow scenario sequence diagram.*

- A logical application system environment diagram, used to represent the service interactions between the internal logical application system, its users and the partner logical systems.

  > ☛ *For more details on application system environment diagrams, see Using the Logical Application System Environment Diagram.*

## Example of logical application system environment

A logical application system environment diagram describes the service interactions between the main internal components of the environment described and the external components.

    Purchase requests are formulated by users in conditions
    specified by "Sales" and "Marketing" services.

    The internal logical application system "Purchase request
    processing" uses a logical "Delivery" application system
    that is external to the described environment.

s
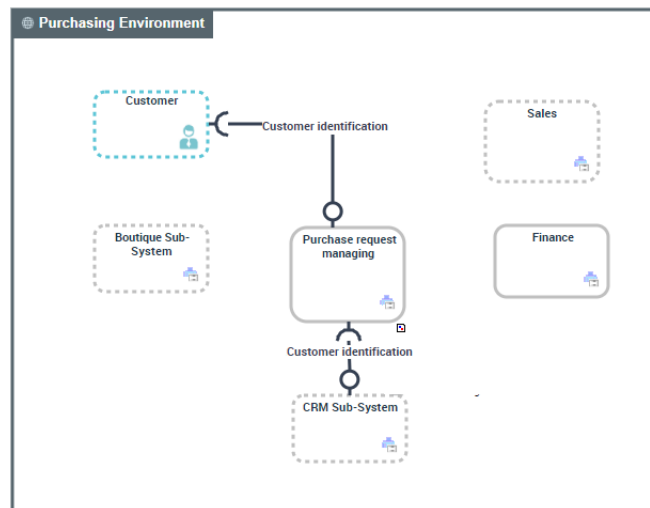


*Logical application system environment diagram*

## Accessing the list of logical application system environments

To access the list of logical application systems environments form **Inventories** navigation menu:

1. Select **Software > Logical Software Architectures**.
   The list of logical application systems appears.
2. Open the **Environment** page of the logical application system of your choice.
   The list of logical application system environments appears.

## Creating a logical application system environment

To create a *logical application system environment*:

1. From the **Inventories** navigation menu, select **Software > Logical Software Architecture**.
2. Open the **Environment** page of the logical application system that interests you and click **New**.
   The **Creation of Logical Application System Environment** window appears.

3. Enter the **Name** of your application system environment and click **OK**.
The new logical application system environment appears in the list.

## Logical application system environment properties

The **Characteristics** properties page for a logical application system environment provides access to:

- its **Name**,
- its **Owner**, by default during creation of a logical application system environment, the current library.
- the text of its **Description**.

☛ *For more details on other property pages proposed by **Hopex IT Architecture**, see Hopex IT Architecture properties pages content.*

## Using the Logical Application System Environment Diagram

A *logical application system environment* is used to represent the service interactions between the internal logical application systems, its users and the partner logical application systems.

A logical application system environment diagram includes:

- *logical application systems* that represent the logical application systems internal to the described environment.

  In the example, this is the logical application system
  "Purchasing Requests Processing".

  📖 *A logical application system is an assembly of other application architectures, logical applications and end users, interacting with application components to implement one or several functions.*

- *partner logical application systems* that represent the logical application systems external to the described environment.

  In the example, this is the logical application system
  "Delivery".

  📖 *A partner logical system is a logical application system external to the environment of the described logical application system. The partner logical system can be a service supplier or a service consumer with respect to components of the logical application system.*

- *Org-Units* and *Position types* that represent the user category of services provided by the environment.
- *Service interactions* between the components representing requests for services.

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

  ☛ *For more details, see Creating a Service interaction.*

# DESCRIBING BUSINESS CAPABILITIES WITH HOPEX IT ARCHITECTURE

The goal of this step, on a strategic level, is to check the suitability between the business capabilities of the enterprise, the functionalities required and the applications that deliver them.

## Business capabilities examples with Hopex IT Architecture

A *business capability* defines an expected skill.

> 📖 *A business capability represents a specific ability that an organization possesses or needs to develop to deliver a particular business outcome.*

For example, to respond to a customer satisfaction objective, the organization must be able to provide services conforming to contractual commitments.

A *business capability map* describes what the enterprise is capable of producing for its internal needs or for meeting the needs of its clients. It is thus based on the main business capabilities of its activity at a given moment.

> 📖 *A business capability map is a set of business capabilities with their dependencies which define a framework for an enterprise stage.*

For example, the standard ability to manage "Operational Activities" is based on the business capabilities to process "Supply", "Sales" and "Complaints", "Order Management" and "Customer Management".
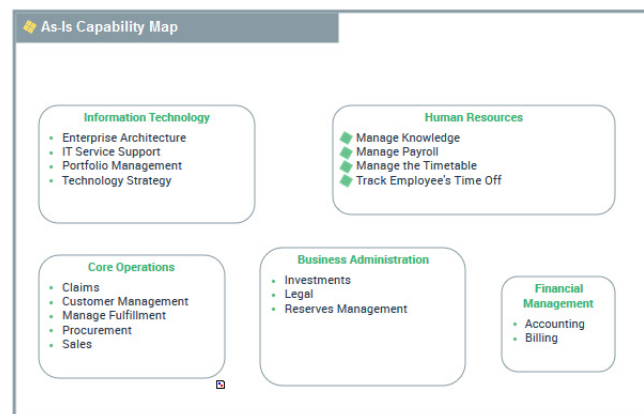


> ☞ *For more details on managing a business capability map, see the "Describing a business capabilities map" chapter in the Hopex IT Business Management guide.*

The description of *business capabilities* and *functionalities* is particularly interesting if business capabilities are associated with the functionalities that fulfill them.

Furthermore, if *applications* are connected to the *functionalities* they implement, they are indirectly connected to *business capabilities*. In **Hopex IT Business Management** , a report allows to check the functional coverage of your applications.

☛ *For more details on the business capabilities reports, see Building Block Breakdown report.*

# Using the Business Capability Maps with Hopex IT Architecture

📖 *A business capability map is a set of business capabilities with their dependencies which define a framework for an enterprise stage.*

## Accessing the list of business capability maps

To access the list of business capability maps from **Capabilities** navigation menu:

❱ Select **Capabilities > Business Capabilities**.
The business capability map list appears

## Creating a business capability map

To create a Business capability map from the **Capabilities** navigation menu:

1. Select **Capabilities > Business Capabilities**.
The business capability map list appears
2. Click the **New** button.
The **Creation of Business Capability Map** window appears.
3. Enter the **Name** of the new business capability map and click **OK**.
The new business capability map appears in the list.

## The properties of a business capability map

The **Characteristics** property page of Capabilities map provides access to:

• its **Owner**, by default during creation of the object, the current enterprise.
• its **Name**,
• the text of its **Description**.

With **Hopex IT Architecture**, a business capability map is described by the following pages:

- the **Structure** page that specifies the list of business capability map components owned and the dependencies between them.

  ☞ *For more details on business capability map components, see Creating a business capability map.*

- the **Fulfillment** page, which provides access to the application environments or logical application system that implement the capability map.

  ☞ *For more details on managing a business capability map fulfillment, see Creating Fulfillment of a Business capability.*

  ☞ *For more details on other property pages proposed by Hopex IT Architecture, see Hopex IT Architecture properties pages content.*

## Creating a business capability map diagram

A  business capability map can be described by two types of diagram.

- A business capability decomposition tree is a diagram that describes the tree structure of a business capability. Focusing on a particular business capability, this type of diagram enables summary representation of business capability breakdown into sub-business capabilities.
- A business capability map diagram that describes the set of business capabilities of the structure.

To create a business capability map diagram:

1. Open the **Diagrams** page of the business capability map that interests you.
2. Click **Create a diagram**.
3. Select **Capability Structure**.
   The **Business Capability Map Diagram** appears. The frame of the business capability map described appears in the diagram.

   ☞ *You can construct this diagram in tabular input mode.*

   ☞

   ☞ *For more information on using tabular entry, see the "Diagrams in Tabular Entry Mode" in the* **Hopex Common Features** *guide.*

# Using Business Capabilities with Hopex IT Architecture

📖 *A business capability represents a specific ability that an organization possesses or needs to develop to deliver a particular business outcome.*

## Accessing the list of business capabilities with Hopex IT Architecture

Business Capabilitues can be accessed from Business Capability Maps list.

To access the list of business capability maps from **Capabilities** navigation menu:

1. Select **Capabilities >  Business Capabilities**.
   The business capability map list appears

2. Open the **Structure** page of the business capability map that interests you.
3. Unfold the **Owned Capability Component** section.
   The list of business capabilities appears in the edit area.

## Creating a business capability

To create a business capability from the **Capabilities** navigation menu:

1. Select **Capabilities >  Business Capabilities**.
   The business capability map list appears
2. Open the **Structure** page of the business capability map that interests you.
3. In the **Owned Capability Component** section, click **New**.
   The **Creation of Business Capability**  window appears.
4. Enter the **Name** of the business capability and click **OK**.
   The new business capability appears in the list.

## Describing a business capability

A business capability is described in more detail by the following elements:

- a more detailed granularity capability breakdown;
- the expected effects of the capability;
- The required functionalities, see Defining the functionalities associated with Business Capabilities;
- the dependencies between capabilities (expected effect of one dependent from the result of the other).

> ☞ *For more details on managing a business capability, see the "Describing a business capability" chapter in the* ***Hopex IT Business Management*** *guide.*

```
For example, the business capability grouping operational
activities is broken down into several business
capabilities: "Customer management", "Supply", "Sales",
"Complaints" and "Order Management".
```
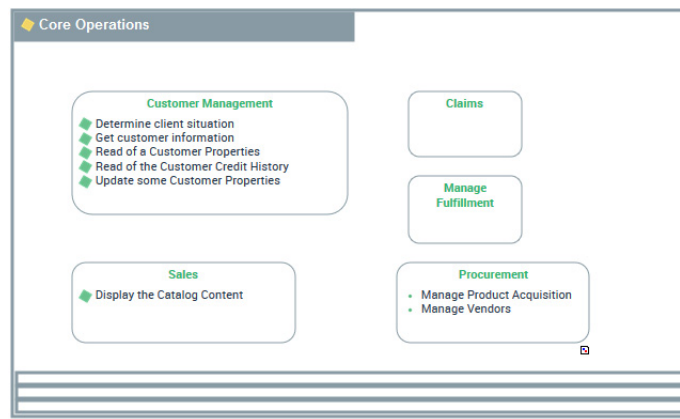
A  business capability can be described by three types of diagram.

- The *Business Capability Decomposition Tree* is a diagram that describes the tree structure of a business capability. Focusing on a particular business capability, this type of diagram enables summary representation of business capability breakdown into sub-business capabilities.
- The *Business Capability Tree* is a diagram that describes the tree structure business capability.
- A *business capability map diagram* that describes the set of business capabilities of the structure.

## Defining the functionalities associated with Business Capabilities

&#x1F4D6; *A technology capability is the ability to deliver a technology service which is required by a technology artifact or an application.*

Each business capability is associated with functionalities that it is able to provide as well as skills that it needs to ensure its functionalities.

```
For example, the "Customer Management" needs the "get
customer information" functionality.
```

☛ *For more information on enterprise functionalities, see Describing functionalities with Hopex IT Architecture.*

To associate a *functionality* with a business capability:

1. Open the **Expected Capabilities** properties window of the business capability.
2. In the **Expected Functionality** section, click **New**.
   An **add functionality** dialog box appears:
3. You can connect an existing functionality or create a new one by entering the name of the new functionality
4. Click **OK**.
   The expected functionality appears in the list of functionalities associated with the business capability.

The functionalities and the expected effects appear in the diagrams, at the bottom of the frame of the capability described.

☛ *For more information on enterprise functionalities, see Describing functionalities with Hopex IT Architecture.*

# USING FUNCTIONALITIES WITH HOPEX IT ARCHITECTURE

A *functionality* is an aptitude expected from an equipment.

  📖 *A functionality is a service required to perform a work. This functionality is generally necessary within an activity in order to execute a specific operation. If it is a software functionality, it can be provided by an application.*

A *functionality map* describes all the functionalities the enterprise is able to cover for its internal needs or for meeting the needs of its clients.

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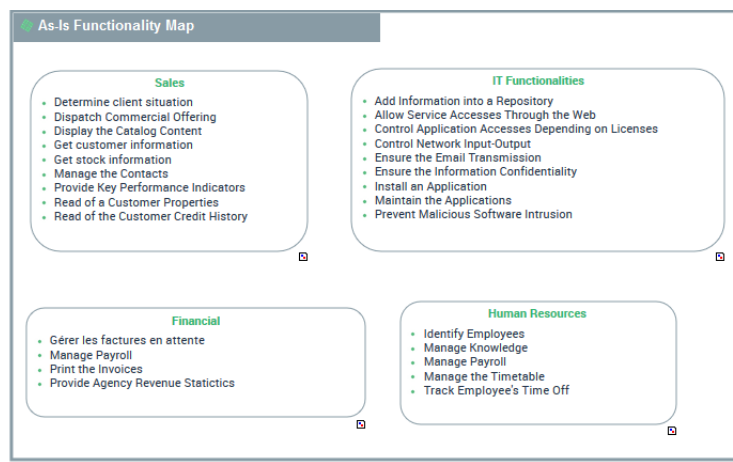


*Example of a functionality map*

  ☛ *For more details on Functionality Maps management, see "Describing the Functionality Map" of* **Hopex IT Business Management** *guide.*

---

## Describing a Functionality Map with Hopex IT Architecture

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

### Accessing the list of functionality maps with Hopex IT Architecture

To access the list of business capabilities from **Capabilities** navigation menu:

❱   Select **Capabilities > Functionalities**.
   The list of functionality maps appears in the edit area.

## Creating a functionality map

To create a functionality map from the **Capabilities** navigation menu:

1.  Select **Capabilities > Functionalities**.
    The list of functionality maps appears in the edit area.
2.  Click **New**.
3.  Modify the **Name** of the functionality map and click **OK**.
4.  Select **Functionality Map**.
    The functionality map appears in the list.

## Creating a functionality map diagram

A  functionality map can be described by two diagram types:

- a functionality decomposition tree is a diagram that describes the tree structure of a functionality. Focusing on a particular functionality, this type of diagram enables summary representation of functionality breakdown into sub-functionalities.
- A functionality map used to represent the set of functionalities of the described map.

To create a functional map diagram:

1.  Right-click the functionality map that interests you and select **Create Diagram**.
2.  Select the diagram type.
    The diagram opens in the edit area. The frame of the functionality map described appears in the diagram.

To create a functionality in a functionality map diagram and describing the dependencies between the functionalities, see "Describing the Functionality Map" chapter in the Hopex IT Business Management guide.

## The properties of a functionality map

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

- its **Owner**, by default, when creating the enterprise or business capability map, this is the current library.
- its **Name**,
- the text of its **Description**.

With **Hopex IT Architecture**, a functionality map is described by the following pages:

- the **Structure** page is used to specify a list of components owned and the dependencies between them.

  ☛ *For more information on the components of a functionality map, see  and* *Creating a Functionality Diagram with Hopex IT Architecture.*

- the **Implementation** property page is used to specify the environments that make it possible to create the described functionality map.

  ☛ *For more details on implementation of functionalities, see* *Creating Fulfillment of a Functionality.*

# Describing functionalities with Hopex IT Architecture

📖 *A functionality is a service required to perform a work. This functionality is generally necessary within an activity in order to execute a specific operation. If it is a software functionality, it can be provided by an application.*

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

- its **Owner**, by default during creation of the functionality, the current enterprise.
- its **Name**,
- the text of its **Description**.
- its **Desired capability effect**.

    ☛ *For more information on the desired capbility effects, see Creating a Functionality Diagram with Hopex IT Architecture.*

## Creating a Functionality Diagram with Hopex IT Architecture

The **Functionality Decomposition Tree** is a diagram that describes the tree structure of a functionality. Focusing on a particular functionality, this type of diagram enables summary representation of functionality breakdown into sub-functionalities.

To create a functionality diagram:

1. Right-click the functionality that interests you and select **Create Diagram**.
2. Select **Functionality Diagram**.
   The diagram opens in the edit area. The frame of the functionality described appears in the diagram.

To create a functionality in a functionality diagram, see "Creating a functionality component in a functionality map diagram" chapter in **Hopex IT Business Management** guide.

To define the dependencies of sub-functionalities, see "Defining Functionality dependencies" chapter in **Hopex IT Business Management** guide.

# Describing a Technology Capability Map with Hopex IT Architecture

📖 *A technology capability map is a set of technology capabilities and their dependencies that, together, defines the scope of a hardware or software architecture.*

## Accessing the list of technology capability maps with Hopex IT Architecture

To access the list of *technology capability maps* from **Inventories** navigation menu:

❱ Select **Deployment > Technology Capabilities**.
   The list of technology capability maps appears in the edit area.

## Describing a technology capability

📖 *A technology capability is the ability to deliver a technology service which is required by a technology artifact or an application.*

With **Hopex IT Architecture**, the use of technology capabilities and technology capabilit maps is identical to that of the  functionalities and functionality maps.

☛ *For more details on the operation of functionality maps and functionalities, see Describing a Functionality Map with Hopex IT Architecture et Describing functionalities with Hopex IT Architecture.*

## Describing a hardware capability

📖 *A hardware capability is the ability to deliver a physical outcome which is required by an organizational resource in order to perform its work. This hardware capability is generally necessary within a computing process in order to execute a specific operation.*

With **Hopex IT Architecture**, the use of hardware  functionalities and hardware 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 a Functionality Map with Hopex IT Architecture et Describing functionalities with Hopex IT Architecture.*

To access the list of *hardware capability maps* from the **Inventories** navigation menu:

❱ Select **Hardware > Hardware Capabilities**.
The list of hardware capabilities appears in the edit area.

# USING FULLFILLMENT MECHANISMS

The fulfillment mechanism is used to connect an element which corresponds to what we know how to do or what we want to do, to a way of realizations that are represented by:

- Concrete elements, such as *applications* or *application systems*.
- Elements at a conceptual level, that is upstream of organizational and technical choices.

## Describing Fulfillment of a Business Capability

This involves connecting the *business capability*, which corresponds to what we know how to do or what we want to do, to a way of achieving that which is represented by:

- *logical applications* or *logical application systems*, for example, at a conceptual level, that is upstream of organizational and technical choices.

      📖 *A logical application system is an assembly of other application architectures, logical applications and end users, interacting with application components to implement one or several functions.*

- *applications* or *application systems*, for example, at a technical level.

For example, constructing the *business capability map* on the one hand and the *logical application system environment* on the other hand, you can check that the business capabilities are implemented by the logical applications.

    ☞ *Conceptual representations are made before organizational and technical choices.*

### Creating Fulfillment of a Business capability

A business capability can be implemented either by an application or application system, or at a conceptual level, by a logical application or logical application system.

To associate an application with a business capability, you must create a business capability fulfillment.

    📖 *A business capability implementation is the physical agent (e.g. an Application System) or the logical agent (e.g. a Business Function) that implements the capability.*

To specify that a business capability is fulfilled by an existing application:

1. Open the **Fulfillments** property page of the business capability that interests you.
2. Click **New**.
   The creation window for a business capability implementation opens.
3. Check the **Add a component and connect a type** box and select the type **Application**.

4.  In the applications list that appears, select the application you wish to connect and click **OK**.
    The capability realization appears in the list with the name of the selected application.

## Analyzing enterprise capability implementation

**Hopex IT Architecture** provides reports to display realization coverage of business capability elements by operational elements such as applications, and according to different perspectives: Organizational, Business/Data, Logical/Physical Application, etc.

☞ *For more details on fulfillment reports for enterprise capabilities, see Building Block Breakdown report.*

# Describing the fulfillment  of a Functionality

The aim here is to connect the *functionalities*, which correspond to what is expected to achieve the objectives, to the means of implementation represented by *applications* (or *application systems*) or, at a conceptual level, to  *logcical applications* (or *logical application systems*) .

☞ *Conceptual representations are made before organizational and technical choices.*

## Creating Fulfillment of a Functionality

A function can be implemented either by an application or application system, or at a conceptual level, by a logical application or logical application system.

To associate an application with a functionality, you must create a functionality fulfillment.

📖 *An implementation describes the relationship between a logical entity and a physical entity that implements it. The physical entity gives the list of logical entities that it implements.*

To specify that a functionality is implemented by a new application:

1.  Open the **Fulfillments** property page of the functionality that interests you.
2.  Click **New**.
    The creation window for a functionality implementation opens.
3.  Check **Ajouter un composant avec un nouveau type** and select the **Application** type.
4.  Click **OK**.
    An application creation dialog box opens.
5.  Enter the **Name** and the **Owner** of your application and click **OK**.
    The functionality fulfillment appears in the list with the name of the selected application.

☞ *The components implemented by technology or hardware capabilities appear in the diagrams representing the functionality.*

### Identifying the  applications associated with functionalities

*Applications* cover *functionalities* associated with *business capabilities*. **Hopex IT Architecture** provides reports to display realization coverage of functionalities by operational elements such as logical or physical application components:

> ☛ *For more details on this breakdown report, see Building Block Breakdown report.*

> ☛ *An example of technology capabilities fulfillment by cloud services is provided, see Accessing the list of Cloud Services*

## Access to implementations from a service point

Services provides by software building blocks (applications or application services) can be accessed by *service points*.

> 📖 *A service point is a point of exchange by which an agent offers a service to potential customers.*

> ☛ *For more information on service points, see Describing Service and Request Points.*

> ☛ *The services requested are defined by a service interface assigned to the service point. For further detail on service interfaces, see Describing a service interface.*

The services provided by software building blocks can address the fulfillments of functionalities or business capabilities.

As a consequence, a *service point* can be connected to one of the fulfillments of the business capability that owns it.

To specify the business capability fulfillments adressed by a service point:

1. Open the **Published  Fulfillments** property page of the service point that interests you.
2. Select the tab corresponding to the fulfillment that interests you.

> ☛ *Only the fulfillments of the object that owns the service point can not be connected to & service point*

# MODELING IT INFRASTRUCTURES

Functionalities proposed by **Hopex IT Architecture** for modeling complex infrastructures enable representation of equipment, IT and organizational resources required for system deployment and operation: service interactions between components, communication means supporting these service interactions, and services offered and used by the modeled architecture.

All the modeled infrastructure elements can be accessed from the navigation menu **Infrastructure > Infrastructure**.

The following points are covered here:

- ✓ Describing Resource Architectures.
- ✓ Describing IT Infrastructures.
- ✓ Describing the Computing Devices.
- ✓ Describing communications in an IT Infrastructure.

# DESCRIBING RESOURCE ARCHITECTURES

A *resource architecture* comprises equipment, IT and organizational resources required for operation of a complex infrastructure (system).

Communications between these components are represented by service interactions and the equipment means supporting these service interactions are the communication channels.

📖 *A resource architecture is the combination of physical and organizational assets configured to supply a capability.*

Services offered by the system to its users are represented by service points. Service points are physically supported by communication ports that enable access to communication means of the system.

## Describing Resource Architectures

📖 *A resource architecture is the combination of physical and organizational assets configured to supply a capability.*

To create a resource architecture from the **Infrastructure** navigation menu:

1. Select **Infrastructure > Resource Architecture**.
   The tree of resource architectures appears.
2. Select **Resource Architecture** and click **New > Resource Architecture**.
   The **Creation of a Resource Architecture** window opens.
3. Enter the **Name** of your architecture as well as its **Owner** and click **OK**.
   The new resource architecture appears in the tree.

### Creating a Resource Architecture Assembly Diagram:

To create a Resource Architecture Assembly Diagram:

1. Select **Infrastructure > Resource Architecture**.
   The tree of resource architectures appears.
2. Select the resources architecture that interests you and click **Create Diagram**.
3. Select **Structured diagram**.
   The resource assembly diagram appears. The frame of the described resource architecture map appears in the diagram.

### Using a Resource Architecture Assembly Diagram

#### Adding a Resource Architecture

To describe that a resource architecture, such as a "call center", implements another resource architecture, such as a "customer management service", for example, you will add the resource architecture used in the user resource architecture diagram.

To add a resource architecture to a Resource Architecture Assembly Diagram:
1.  In the objects toolbar of the Resource Architecture Assembly Diagram, click **Resource Architecture**.
2.  Click in the frame of the Resource Architecture Assembly Diagram.
    An addition dialog box prompts you to select a resource architecture.
3.  Select an existing resource architecture and click **OK**.

> ☞ *To create a resource architecture, simply enter its name and click* ***Create*** *button.*

### *Adding an IT Infrastructure or a Resource Configuration*

> 📖 *An IT Infrastructure is composed of several connected computer devices (IT network nodes or computing systems) and computer networks.*

> ☞ *For more details on IT infrastructures, see* *Describing IT Infrastructures.*

> 📖 *A resource configuration is a set of physical and human resources configured to provide a business capability.*

> ☞ *For more details on resource configurations, see* *Describing a resource configuration.*

To describe that a resource architecture is based on IT resources such as a communication network, workstations housing applications, you will add *IT Infrastructure* type components to the Resource Architecture Assembly Diagram.

> 📖 *An IT Infrastructure is composed of several connected computer devices (IT network nodes or computing systems) and computer networks.*

> ☞ *For more details on IT infrastructures, see* *Describing IT Infrastructures.*

To create an *IT Infrastructure*:
1.  In the diagram objects toolbar, click **IT Infrastructure**.
2.  Click in the diagram frame.
    An addition dialog box prompts you to select the IT infrastructure to be deployed.
3.  Select the IT infrastructure that interests you and click **OK**.
    The IT infrastructure appears in the diagram.

> ☞ *To create an IT infrastructure, simply enter its name and click* ***Create*** *button.*

> ☞ *In the same way, you can add a resource configuration in the Resource Architecture Assembly Diagram. For more details on resource configurations, see* *Describing a resource configuration.*

### *Adding an Org-Unit or a Position Type*

> 📖 *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external*

*org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

📖 *A position type represents a status assigned to an individual or a group of individuals with the aim of defining an organization or a hierarchy.*

To describe that a Resource Architecture such as a call center uses operators to take calls and handle requests, you will create a *Position Type* component.

To add a Position Type to a Resource Architecture Assembly Diagram:

1. In the objects toolbar of the Resource Architecture Assembly Diagram, click **Position Type**.
2. Click in the frame of the resource architecture described.
   An addition window prompts you to choose the **Position Type** that you wish to use:
3. Select the Position Type concerned and click **OK**.
   The Position Type appears in the diagram.

    ☛ *To create a position type, simply enter its name and click **Create** button.*

    ☛ *In the same way, you can add an org-unit in the Resource Architecture Assembly Diagram.*

### Describing the Services in a Resource Architecture Assembly Diagram

A resource architecture is created to assure one or several services.

Expected and realized services are represented by:

- *service points*

    📖 *A service point is a point of exchange by which an agent offers a service to potential customers.*

- *request points*

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

    ☛ *For more details, see Service points and Request points.*

### Describing Service interactions in a Resource Architecture Assembly Diagram

In a resource architecture assembly diagram, *Service Interactions* enable representation of exchanges between organizational entities.

📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

Exchange terms are defined by a *service interface* assigned to the service interaction.

📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

You can define service interactions between:

- Two components of *resource architecture* type to represent exchanges between these entities,
- A component of *resource architecture* type and an *IT infrastructure* to represent the terms of use of the equipment resource by the organizational resource.  For example, you can represent that operator hardware use is arranged by booking.
- two components of *IT infrastructure* type to represent the terms of use of one IT resource by another in the context of the modeled resource architecture.
- a *service point* and one or more *resource architecture* type components to represent implementation of the service within the resource architecture,
- a component of  *resource architecture* type and a *request point* to represent that the entity calls a resource of an external organization.

☛ *For further details, see .For further details, see* Service interactions*.*

### Channels and communication ports

In a resource architecture, network channels support the transfer of information from one hardware asset to another.

☛ *For more details on creation of these channels and the associated communication protocols, see* Network channels*.*

Communication ports enable connection of resource architecture physical assets with external equipment elements.

# Describing a Resource Architecture Environment

📖 *A business architecture environment represents the relationships of a business functional area with its partners.*

## Creating a resource architecture environment

To create a resource architecture environment using the **Infrastructure** navigation menu:

1. Select **Infrastructure > Resource Architecture**.
   The tree of resource architectures appears.
2. Select the resource architecture that interests you.
3. Open the **Environment** property page.
   The list of resource architecture environments appears.
4. Click **New**.
   The new resource architecture environment appears in the list.
5. Open the properties of the resource architecture environment to modify its **Name** and its **Owner**.

## The properties of a resource architecture environment

The **Characteristics** properties page of resource architecture environment provides access to:

- its **Owner**, by default during creation of the object, the current enterprise.
- its **Name**,
- the text of its **Description**.

With **Hopex IT Architecture**, a resource architecture environment is described by the following property pages:

- The **Component** page which provides access to the list of internal and partner components of resource architecture environment.
- The page **Component** wich provides access the diagrams.
- The **Reports** page provides access to the reports available for object.

## To create a resource architecture environment diagram

To create a resource architecture environment diagram:

1. Open the **Environments** page of the resource architecture that interests you.
2. Select the resource architecture environment that interests you and click **Create Diagram**.
3. Select **Structured diagram**.
   The resource architecture environment diagram appears.

## Describing a resource architecture environment diagram

### Adding a Resource Architecture

    📖 *A resource architecture is the combination of physical and organizational assets configured to supply a capability.*

To add a resource architecture to a resource architecture diagram environment:

1. In the objects toolbar of the resource architecture environment, click **Resource Architecture**.
2. Click in the frame of the resource architecture environment described.
   A dialog box prompts you to select the implemented resource architecture. You can select an existing resource architecture or create a new one.

       ☛ *In the case where the resource architecture you want to use does not yet exist in the repository, simply enter its name.*

3. Click **OK**.

### Creating a Partner Resource Architecture

To describe that an external resource architecture is implemented in the described environment, you will add a *resource architecture partner* component in the environment diagram.

    📖 *A partner resource architecture is the installation of an external resource architecture in another resource architecture or an environment.*

To create a **Partner Resource Architecture**:

1. In the resource architecture environment diagram toolbar, click **Partner Resource** and select **Resource Architecture**.
2. Click in the frame of the resource architecture environment described. An addition dialog box prompts you to select a resource architecture to add. You can enter the name of a new resource architecture.
3. Click **OK**.

### Creating a human asset

To describe that the resource architecture environment services are used by customers, for example, you will create a *position type* or an *org-unit*.

> 📖 *A position type represents a status assigned to an individual or a group of individuals with the aim of defining an organization or a hierarchy.*

> 📖 *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

> ☛ *For more details on creating a human asset, see Adding an Org-Unit or a Position Type.*

### Describing communications

The elements below allow you to describe the technical and organizational communications:

- Ports and network channels,

  > ☛ *For further details, see .For further details, see Describing technical communications.*

- Service interactions, service and request points,

  > ☛ *For further details, see .For further details, see Describing the services communications.*

  > 📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

# Describing a resource configuration

> 📖 *A resource configuration is a set of physical and human resources configured to provide a business capability.*

## Creating a resource configuration

To create a resource configuration from the **Infrastructure**  navigation menu:

1. Select **Infrastructure > Resource Configuration**.
   The list of resource configurations appears.

2. Click **New**.
   The **Creation of a resource configuration** window opens.

3. Enter the **Name** of your resource configuration as well as its **Owner** and click **OK**.
   The new resource configuration appears in the list.

## Creating a resource configuration diagram

To create a resource configuration diagram:

1. Select the resource configuration and click **Create a diagram**.

2. Select **Structured diagram**.
   The resource configuration diagram opens.

## Using a Resource Configuration Diagram

In a resource configuration diagram, you can insert:

- *IT Infrastructures*; see Describing an IT infrastructure

       *An IT Infrastructure is composed of several connected computer devices (IT network nodes or computing systems) and computer networks.*

- *IoT Devices*,

       *An IoT device is both a hardware device and a computing device which provides cvombined hardware and information services to the users using it directly. As a hardware device, it embeds sensors - e.g. accelerometer - which provide data to the embedded computing device. As a computing device, it can host data stores or run applications. Examples: smartwatch with GPS tracker, on-line surveillance video camera with live IP video feed, connected weighting scale with weight history management*

- *IT networks*; see Describing an IT network

       *An IT network is set of IT equipment components (e.g.: routers, switches, firewalls) that allow remote communications between computing devices (e.g.: IT server). An IT network can be broken down into sub-networks.*

- *Hardware* elements, see Describing an Hardware,

       *Non-IT Hardware can embed computers. Together with their embedded computers, they provide information and IS services. Examples: Connected Truck with Delivery Calendar Application and connected Drone with Online Payment Application. Hardware device can also provide hardware functionalities. Example: Connected fridge*

*providing ordering functionalities and of course a freezing hardware functionality and connected drones fly and provide Online Payment.*

- *Position types* or *Org-Units*.

  📖 *A position type represents a status assigned to an individual or a group of individuals with the aim of defining an organization or a hierarchy.*

  📖 *An org-unit represents a person or a group of persons that intervenes in the enterprise business processes or information system. An org-unit can be internal or external to the enterprise. An internal org-unit is an organizational element of enterprise structure such as a management, department, or job function. It is defined at a level depending on the degree of detail to be provided on the organization (see org-unit type). Example: financial management, sales management, marketing department, account manager. An external org-unit is an external entity that exchanges flows with the enterprise. Example: customer, supplier, government office.*

  ☛ *For more details on creating a human asset, see Adding an Org-Unit or a Position Type.*

- Ports and network channels, see Describing technical communications.
- service and request points, see Describing the services communications.
- Service Interaction, see Describing technical communications.

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

# Describing an Hardware

📖 *Non-IT Hardware can embed computers. Together with their embedded computers, they provide information and IS services. Examples: Connected Truck with Delivery Calendar Application and connected Drone with Online Payment Application. Hardware device can also provide hardware functionalities. Example: Connected fridge providing ordering functionalities and of course a freezing hardware functionality and connected drones fly and provide Online Payment.*

## Creating an Hardware

To create an Hardware element from the **Inventories** navigation menu:
1. Select **Hardware > Hardware**.
   The list of Hardware elements appears.
2. Click **New**.
   The created hardware element appears in the list.

## Creating a Hardware Assembly Structure Diagram

To create a hardware assembly structure diagram:
1. Select the hardware you are interested in and click **Create a diagram**.
2. Select **Structured diagram**.
   The  Hardware Assembly Structure Diagram opens.

## Using a hardware assembly structure diagram

In a Hardware Assembly Structure Diagram, you can insert:

- *IT servers* and *Computing devices*, see Describing a Computing Device,

  📖 *An IT Server is an IT component providing a service to users connected via an IT network. This IT component can house databases and run applications.*

  📖 *A computer device is a device which provides a computing service to the end-users directly. This computer can house databases and run applications. This is, for example, a workstation, a laptop or a smartphone.*

- *IoT Devices*, see Describing a Computing Device,

  📖 *An IoT device is both a hardware device and a computing device which provides cvombined hardware and information services to the users using it directly. As a hardware device, it embeds sensors - e.g. accelerometer - which provide data to the embedded computing device. As a computing device, it can host data stores or run applications. Examples: smartwatch with GPS tracker, on-line surveillance video camera with live IP video feed, connected weighting scale with weight history management*

- *Hardware Component*, see Describing an Hardware,
- Communication ports and network channels, see Describing technical communications.
- service and request points, see Describing the services communications.
- service interactions.

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

  ☛ *For more information about service interactions, see Describing the services communications.*

# DESCRIBING IT INFRASTRUCTURES

## Describing an IT infrastructure

📖 *An IT Infrastructure is composed of several connected computer devices (IT network nodes or computing systems) and computer networks.*

You can describe the components of an *Infrastructure* in an infrastructure assembly diagram.

### Creating an IT infrastructure

To create an IT infrastructure from the **Infrastructure** navigation menu:
1.  Select **Infrastructure > IT Infrastructures**.
    The IT Infrastructures tree deplays.
2.  Select the **IT Infrastructures** file and click **New > IT Infrastructure**.
    The **Creation of IT technical device** window appears.
3.  Enter the **Name** of your infrastructure as well as its **Owner** and click **OK**.
    The new IT Infrastructure pops up in the tree.

### Creating an Infrastructure Assembly Structure Diagram

To create an infrastructure assembly structure diagram:
1.  Select the IT infrastructure and click **Create a diagram**.
2.  Select **Structured diagram**.
    The infrastructure assembly structure diagram opens in the edit zone.

### Using an infrastructure assembly structure diagram

In an infrastructure assembly structure diagram, you can insert:

You can insert in this diagram:

- *IT servers* and *Computing devices*, see Describing a Computing Device,

  📖 *An IT Server is an IT component providing a service to users connected via an IT network. This IT component can house databases and run applications.*

  📖 *A computer device is a device which provides a computing service to the end-users directly. This computer can house databases and run applications. This is, for example, a workstation, a laptop or a smartphone.*

- *IoT Devices*, see Describing a Computing Device,

  📖 *An IoT device is both a hardware device and a computing device which provides cvombined hardware and information services to the users using it directly. As a hardware device, it embeds sensors - e.g. accelerometer - which provide data to the embedded computing device. As a computing device, it can host data stores or run applications. Examples: smartwatch with GPS tracker, on-line surveillance video*

*camera with live IP video feed, connected weighting scale with weight history management*

- *Network devices*, see Describing a Computer Network Device,

  📖 *An IT device can host and run Software Technology. Conjointly with its hosted software, it provides services. This consists of, for example: Wifi Access Point, Firewall, router, switch, printer, Hard Drive.*

- *IT Network Components*, see Describing an IT network,

  📖 *An IT network is set of IT equipment components (e.g.: routers, switches, firewalls) that allow remote communications between computing devices (e.g.: IT server). An IT network can be broken down into sub-networks.*

- Ports and network channels, see Describing technical communications.
- service and request points, see Service points see Request points.
- service interactions.

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

  ☛ *For more information about service interactions, see Describing the services communications.*

The **Infrastructure description** report allows you to analyze the infrastructure components and the expected relationships between them to verify that each element described in the diagram hosts a software or hardware component.

☛ *For further details, see .For further details, see Infrastructure Description Report.*

# Describing an IT network

📖 *An IT network is set of IT equipment components (e.g.: routers, switches, firewalls) that allow remote communications between computing devices (e.g.: IT server). An IT network can be broken down into sub-networks.*

## Creating an IT network

To create an IT network:

1. From the **Inventories** navigation menu, select **Infrastructure > IT Networks**.
   The list of IT networks appears.
2. Click **New**.
   The **IT Network Creation** window appears.
3. Enter the **Name** of your network as well as its **Owner** and click **OK**.

## Creating an IT network

An IT network is described by an infrastructure assembly structure diagram.

To create an infrastructure assembly structure diagram from an IT network:

1. Right-click the IT network and select **Create a diagram**.

2. Select **Structured diagram**.
   The infrastructure assembly structure diagram opens in the edit zone.

   ☛ *For more details on this type of diagram, see Using an infrastructure assembly structure diagram.*

# Describing a Facility

📖 *A facility is a model of site of interest for the enterprise. Examples: Data Center, Factory or Outlet*

## Creating a facility

To create a facility:

1. From the **Inventories** navigation menu, select **Infrastructure > Facilities**.
   The list of facilities appears.
2. Click **New**.
   The **Creation of facility** dialog box appears.
3. Enter the **Name** of your facility as well as its **Owner** and click **OK**.

## To create a resource configuration diagram from a facility

A resource configuration is described by a resource configuration diagram.

To create a resource configuration diagram from a facility:

1. From the **Inventories** navigation menu, select **Infrastructure > Facilities**.
2. Right-click the facility and click **Create a diagram**.
3. Select **Structured diagram**.
   The resource configuration diagram opens in the edit zone.

   ☛ *For more details on this type of diagram, see Using a Resource Configuration Diagram.*

# DESCRIBING THE COMPUTING DEVICES

## Describing a Computing Device

### Accessing the list of computing devices

To access all the different types of computing devices:

❱ From the **Inventories** navigation menu, select **Infrastructure > Computing Devices**.
The list of all computing devices appears:

- *IoT Devices*,

  📖 *An IoT device is both a hardware device and a computing device which provides cvombined hardware and information services to the users using it directly. As a hardware device, it embeds sensors - e.g. accelerometer - which provide data to the embedded computing device. As a computing device, it can host data stores or run applications. Examples: smartwatch with GPS tracker, on-line surveillance video camera with live IP video feed, connected weighting scale with weight history management*

- *IT Servers*,

  📖 *An IT Server is an IT component providing a service to users connected via an IT network. This IT component can house databases and run applications.*

- *Cloud services*

  📖 *The Cloud Service (considered as an IoT device) can be used in a deployment architecture. The Cloud service picture appears in the frame of the corresponding deployable package.*

  ☛ *For more information on Cloud services, see Using Cloud Services.*

- *Computer devices*

  📖 *A computer device is a device which provides a computing service to the end-users directly. This computer can house databases and run applications. This is, for example, a workstation, a laptop or a smartphone.*

### Creating an Computer Device

To create **Computer device**:

1. From the **Inventories** navigation menu, select **Infrastructure > Computig Devices**
   The different computing device types are sorted out in folders.
2. Select the folder corresponding to the type of computing device you want to create and click **New**.
3. Enter the **Name** of your computing device as well as its **Owner** and click **OK**.
   The new computing device appears in the list.

## Creating a Computing Device Assembly Diagram

To create a computing device assembly diagram:

1. Right-click the computing device and click **Create Diagram**.
2. Select **Structured diagram**.
   The computing device assembly diagram opens in the edit zone.

You can insert the following in a computer assembly diagram:

- *deployable Package Hosts*, see  Adding a deployable application package in an application deployment architecture diagram,

  📖 *A deployable application package is a split of application code according to technical criteria for hosting purpose. For example, it may be N tiers, Front End/Back End/… or GUI/Business Logic/Database etc... Each deployable application package is associated to required technologies (for running) and can host code for several IT services. Architect can also prescribes a kind of hosting artefact (IaaS/PaaS cloud service or IT server model).*

- *software technology hosts*,

  📖 *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.*

- *microservice hosts*

  📖 *A microservice 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.*

- *data store hosts*,

  📖 *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.*

- ports and network channels,
- service and request points,
- service interactions.

  ☛ *For more details on the creation of service interactions, service and request points, channels and associated communication protocols, see Describing communications in an IT Infrastructure.*

# Describing a Computer Network Device

## Accessing the list of computer network devices

The types of computer network devices commonly available in **Hopex IT Architecture** are:

- Hub,
- Printer,
- Modem,
- Firewall,
- Wifi Hotspot,
- Bridge,
- Network Device,
- Router,
- Satellite,
- Switch.

To access the list of a specific type of computer network device:

1. From the **Inventories** navigation menu, select **Infrastructure > Computer Network Devices**
2. Unfold the file corresponding to the type of device you are interested in. The list of computer network devices searched for displays.

## Creating a Computer Network Device

To create a **Printer**-type computer network device, for example:

1. From the **Inventories** navigation menu, select **Infrastructure > Computer Network Devices**
2. Select the file corresponding to the type of network device you are interested in.
3. Click **New**.
4. Enter the **Name** of your network device and its **Owner**, and click **OK**. The new computer network device pops up in the list.

# DESCRIBING COMMUNICATIONS IN AN IT INFRASTRUCTURE

In an IT Infrastructure, communications are based on:
- service points, request points, and service interactions for communications related to the service,
- ports and network channels, for technical communications.

## Describing the services communications

### Service interactions

*Service interactions* show the exchanges planned between the organization entities.

> 📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

Exchange terms are defined by a *service interface* assigned to the service interaction.

> 📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

You can define service interactions between:
- Two components of resource architecture type to represent exchanges between these entities,
- A component of resource architecture type and an IT infrastructure to represent the terms of use of the equipment resource by the organizational resource. For example, you can represent that operator hardware use is arranged by booking.
- two components of IT infrastructure type to represent the terms of use of one IT resource by another in the context of the modeled resource architecture.
- a service point and one or more resource architecture type components to represent implementation of the service within the resource architecture,
- A component of architecture use type and a request point to represent that the entity calls a resource of an external organization.

For more information on service interaction management terms, see Managing Service Interactions.

## Service points

Services provided by infrastructure elements are represented by *service points*.

    📖 *A service point is a point of exchange by which an agent offers a service to potential customers.*

The service id requested according to specific terms that are defined by a *service interface* assigned to the service point.

    📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

The resources activated to carry out a service are connected to the service point by service interactions. If the activation of several resources is necessary, several service interactions must be created between the service point and the architecture resources.

To create a service point, see Describing Service and Request Points.

## Request points

A *request point* is used to represent the use of an external service.

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

The service is requested according to specific terms that are defined by a *service interface* assigned to the request point.

    📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

The resources emitting a request are connected to the request point via a service interaction.
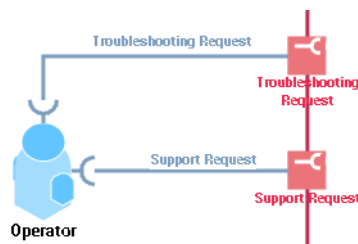


In the example, request points represent service requests made between call center operators and other organizations.

The request point creation procedure is identical to that for service points. For further details, see .For further details, see Describing Service and Request Points.

# Describing technical communications

## Communication ports

*Communication Ports* are physical points of communication that can be defined in technical infrastructures and resource architectures.

> 📖 *A communication port is a physical point of communication with a resource. It adheres to the specific communication protocol. A communication port implements service and requests points.*

*Communication Ports* assure physical transfer of information exchanged on service points and request points.

Communication ports comply with specific "Communication Protocols". See Network communication protocols.

## Network channels

The *network channels* connect hardware resources between each other, to organizational resources or to communication ports.

> 📖 *A network channel is a physical connector between resource elements. It supports service interactions defining communication protocols between physical resources It connects external resource elements through their Communication Ports.*

### Creating a network channel

To create network channel:

1. In the object inserting bar of the resource assembly diagram, click

   **Network channel** ✏ .
2. Draw a link between the two communication entities.
   The channel appears directly in the diagram.

To define the communication protocol associated with the channel:

1. Open the **Supported Protocols** property page and click **Connect**.
2. In the query window that appears, select the communication protocol that interests you and click **Connect**.
   The protocol name appears alongside the channel.

## Network communication protocols

A *Communication Protocol* is supported by network channel.

> 📖 *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.*

For example, an HTTPS protocol is based on an HTTP protocol for transport, those protocols are based on TCP, which is itself based on Ethernet.

A user may wish to build a customized layer of communication protocols and assign these to communication ports and communication channels.

> ☞ *Communication protocols supported by a communication port must be compatible with the communication ports to which they are connected.*

# Connecting a Service Interaction to a Network Channel

To indicate that a service interaction is supported by network channel:

1. In the resource assembly diagram objects toolbar, click the link button.
2. Draw a link between the service interaction and network channel supporting it.
   A dotted line appears in the diagram.

To access the list of service interactions supported by a network channel:

❱ Open the **Managed service interactions** property page of the network channel you are interested in.
   The name of the service interaction appears in the list.

> ☞ *You can use the **Connect** button to connect other interactions to network channel.*

# ACCESSING THE SOFTWARE DESIGN

**Hopex IT Architecture** offers the tools to assist architects in specifying updates to their IT system.

☛ *To access UML functions, you must be connected **Solution Architect** profile or with **Solution Architecture Functional Administrator** profile.*

The following points are covered here:

✓ UML modeling of data.
✓ Describing Batch Processing.
✓ Defining User Interfaces.

# UML MODELING OF DATA

**Hopex IT Architecture** provides the tools required to model logical data via class diagrams and data models.

> ☛ *To access UML functions, you must be connected* **Solution Architect** *profile or with* **Solution Architecture Functional Administrator** *profile.*

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

> 📖 *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.*

> ☛ *For more details on creating and updating a data model, see the "Data Model" chapter in the* **Hopex Data Architecture** *guide.*

The different logical view concepts are described in the paragraph.

## 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.
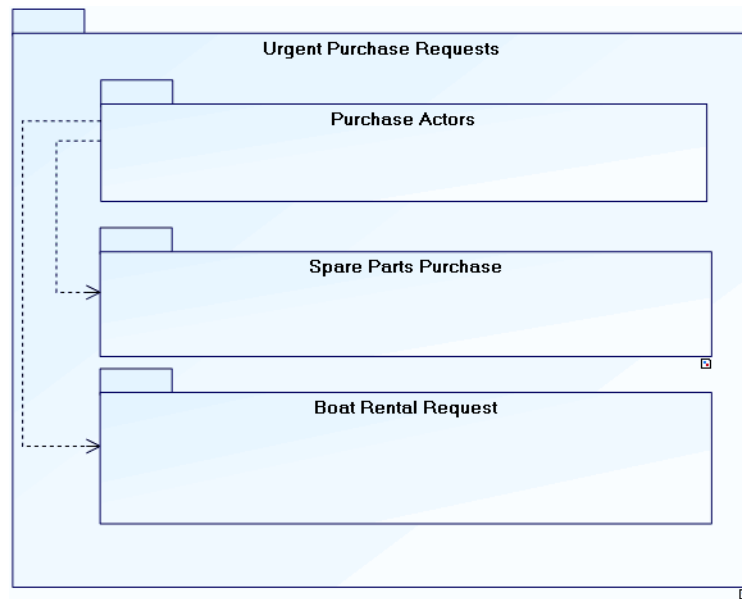


```
Urgent purchase requests are provided to process purchase
of spare parts and boat rental requests. In both of these
cases, users are actors of the purchasing domain.
```

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

☛ *For more details on building a class diagram, see The Class Diagram.*

# 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.

### *Example*

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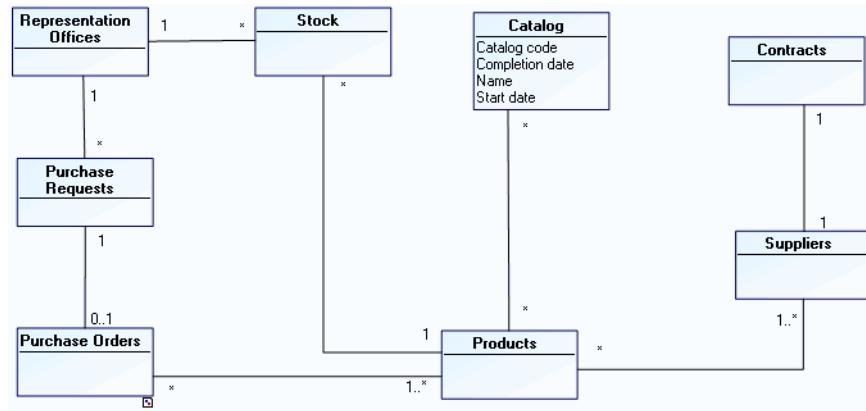


The application manages purchase requests, orders and product stock levels in each of the representation offices.

A centralized catalog of products and suppliers is installed.

Contracts with referenced suppliers are also accessible from the application.

# Data areas

A Data Area represents a restricted data structure dedicated to the description of a software Data Store (see Managing Data). 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 domain diagram represents a data structure relating to Orders; it describes classes and their relationships in a Whole/Part formalism.
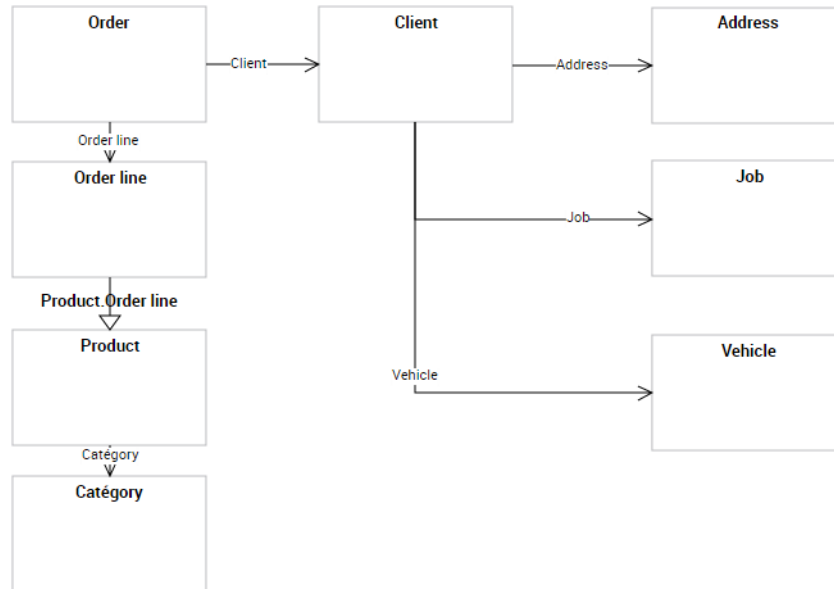


To address these specific use cases, you can create *Data Views* in which you can see and modify the scope covered by the classes.

> 📖 *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.*

# DESCRIBING BATCH PROCESSING

With **Hopex IT Architecture**, you can describe the sequencing of automated processing in **batch planning structure diagrams**.

☛ *To access UML functions, you must be connected **Solution Architect** profile or with **Solution Architecture Functional Administrator** profile.*

☛ *To see the **Batch Processing**, open the **Options** window and check that **IT Architecture > User Interface and Batch Features (ADES)** option is activated.*

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 described by a **batch planning** or 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 planning that consists of running execution of a program using the appropriate parameters.*

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

☛ *For further details, see Building a Batch Planning Structure Diagram.*

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 further details, see Creating a Batch Program Structure Diagram.*

## 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.*

## Creating a batch planning structure diagram

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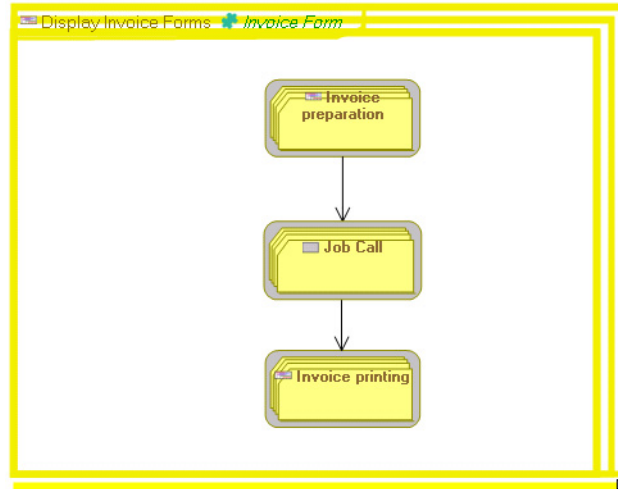


To create a batch planning structure diagram:

1. Click **Design (UML)** navigation menu.
2. Select **Batch and Program Implementation > Batch and Program**.
   The list of batch plannings appears.
3. Open the **Diagrams** property page of the batch planning in question and select **Create a diagram**..
4. Creating **Batch Planning Structure Diagram**.
   The diagram opens.

## Adding a call for batch processing in the diagram

The components of a **batch planning** 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 **Object Type** field and select **Batch Planning** in the drop-down list.
3. Click the arrow at the right of the **Short Name** field and select the batch planning that interests you.
4. 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 planning 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**.
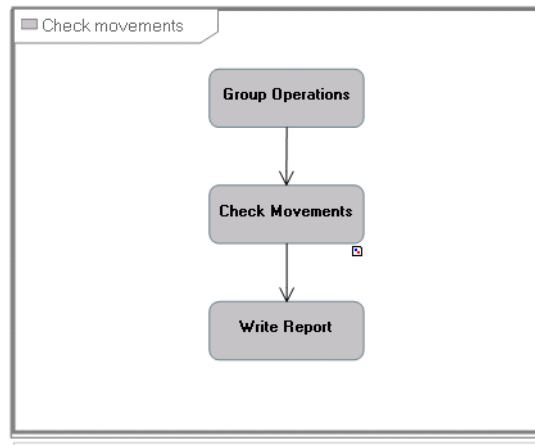


To create the batch program structure diagram:

1. Open the **Diagrams** property page of the program of your choice and click **Create a diagram**.
2. In the dialog box, select **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 **Name** field and select the **Program** that interests you.
3. Click **OK**.
   The program call appears in the diagram.

The execution scheduling of programs is defined by **batch sequences**, see Defining batch sequencing.

## Using system process batch realizations

A realization mechanism is provided to specify that a system process describes the execution of a **Batch Planning** or a **Program**.

To describe that an batch plan is associated with an application process:
1. Open the **Characteristics > Realization** property page of the batch planning that interests you.
2. Click the **New** button.
   The realization creation window opens.
3. In the **Object type** field, select **System process batch realizations** and click **Next**.
4. Select the application process that interests you and click **New**.
   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.

> ☞ *To access UML functions, you must be connected **Solution Architect** profile or with **Solution Architecture Functional Administrator** profile.*

> ☞ *To see the **User Interfaces**, open the **Options** window and check that **IT Architecture > User Interface and Batch Features (ADES)** option is activated.*

## Creating a user interface

To create a user interaction using **Design (UML)** navigation menu:

1. Select **User Interfaces**.
   The list of user interfaces appears.
2. Click **New**.
3. Enter the name of the interface.
4. Click **OK**.

## Building a User Interface Diagram

To create an interface diagram:

1. Select the User interface that interests you and click **Create Diagram** button.
2. In the dialog box, select **User Interface Diagram**.
   The UI diagram opens in the Edit window.

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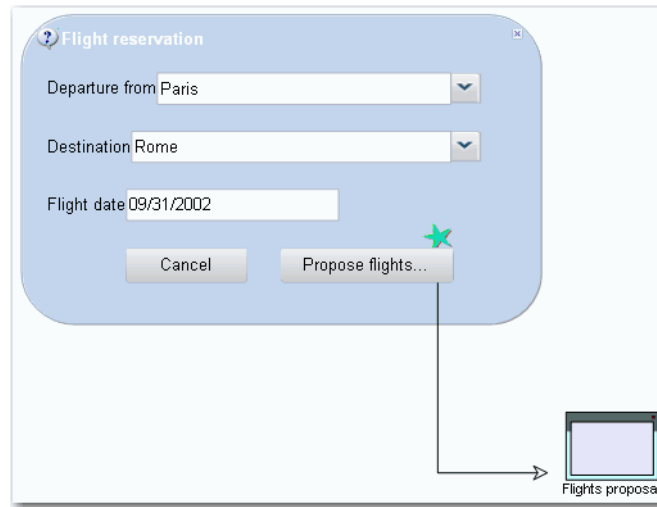


The interface is presented in the form of a dialog box, in which various fields must be completed.
- Departure city
- Destination
- Flight date

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 Group
- 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 **Characteristics** property page of the sketch.

2. In the **User Interface Event Type** text box, click the arrow and select
**Query User Interface Event Type**.
The Query dialog box appears:

3. Click **Find**.
The list of event types appears.

4. Select the type required and click **OK**.

### *Connecting the event to an element*

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

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

# DESCRIBING INFORMATION EXCHANGES

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

To simplify the service interface creation, service operation templates and service interface templates can be used.

✓ Managing Service Interactions;
✓ Describing a service interface.
✓ Describing a Service Operation.
✓ Using a Service Interface Template.

# MANAGING SERVICE INTERACTIONS

A *Service Interaction* represents the exchange of information between architecture components.

> 📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

The content of a service interaction is described by a *service interface*.

> 📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

> ☛ *For more details on service interfaces, see Describing a service interface.*

In a "Purchasing Requests Processing" application system structure diagram, two service Interfaces are used by the different service interactions.
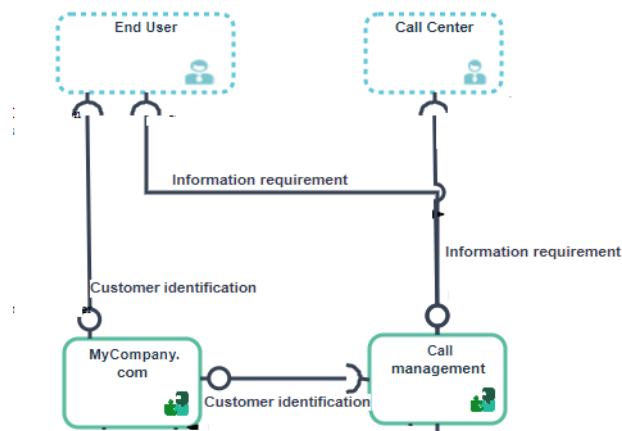


*Adding a service interaction to an application system structure diagram*

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

# Creating a Service interaction

To create a service interaction:

1. In the objects toolbar for a diagram, click **Service interaction** ⬱
2. Click the entity requesting the service and draw a link to the entity providing the service.
3. In the add service interaction dialog box, specify the service interface you wish to use.

> ☛ *You can also use a new service interface. For more details, see Creating a service interface.*

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*.

> 📖 *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 a *service interface* assigned to the service point.

> 📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

> ☛ *For further detail on service interfaces, see Describing a service interface.*

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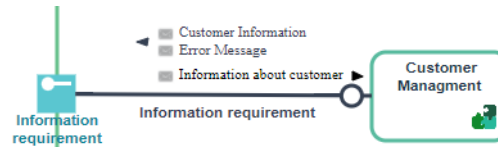


In the example presented here, the IT Service "Customer Management" is activated by the interaction service "Information request".

☛ *To create a service point, see Creating a Service Point or a Request Point.*

☛ *The **Published Fulfillments** property page of the service point enables the access to the capabilities implemented by the service point that interests you. To create a service point, see Access to implementations from a service point.*

## Request points

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

&#x1F4D5; *A request point is a point of exchange by which an agent requests a service from potential suppliers.*

The service is requested according to precise terms defined by a *service interface* assigned to the request point.

&#x1F4D5; *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

☛ *For further detail on service interfaces, see Describing a service interface.*

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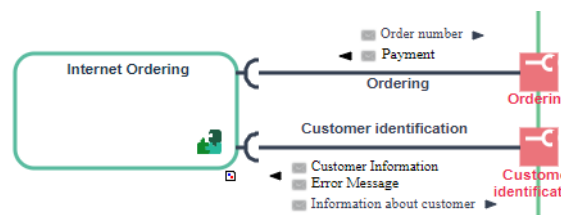


In the example, request points represent requests for service executed by the "Email Order Management" IT service to identifier a customer and issue an order.

☛ *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 **Service Interface** field to define the service interface enabling activation of this service point, and select, for example, **Connect Service interface**.
   A query window opens.
4. Select the service interface associated with this service point and click **Connect**.
5. Click **Next**.
   A dialog box opens proposing a list of the service interface 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.

# DESCRIBING A SERVICE INTERFACE

A *Service interface* represents the exchange of information between architecture components.

  📖 *A Service Interaction represents an interaction for service purpose between entities in a specific context inside or outside a company. These entities can be enterprise org-units, applications, activities or processes, as well as external org-units. The content of this interaction is described in a service interface.*

The content of a service interaction is described by a *service interface*.

  📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

A service contract is described by a sequence of operations which are represented:

- By *service interfaces used*,

  📖 *A service interface use is associated to a service interface. It enables representation of complex exchanges.*

- Or by *service operations used*.

  📖 *A service operation use represents the usage of a service operation in an service interface.*

  ☛ *For further detail on service operations, see Describing a Service Operation.*

---

## Examples of Service Interface Diagrams (BPMN)

A service interface is described by a sequence of steps which are represented:

- By *service operations used*,
- By *service interfaces used*.

  📖 *A service interface use is associated to a service interface. It enables representation of complex exchanges.*

The service interface roles, presented at the border of the frame, represent participants:

- customer/supplier, or
- sender/recipient

A service interface can be described by involving more than two participants. In this case, a role is consumer of the service interface and the others are providers.

## Example of Service Interface Diagram (BPMN)

The service interface diagram associated with the "customer identification protocol
" describes in BPMN formalism the operations executed.
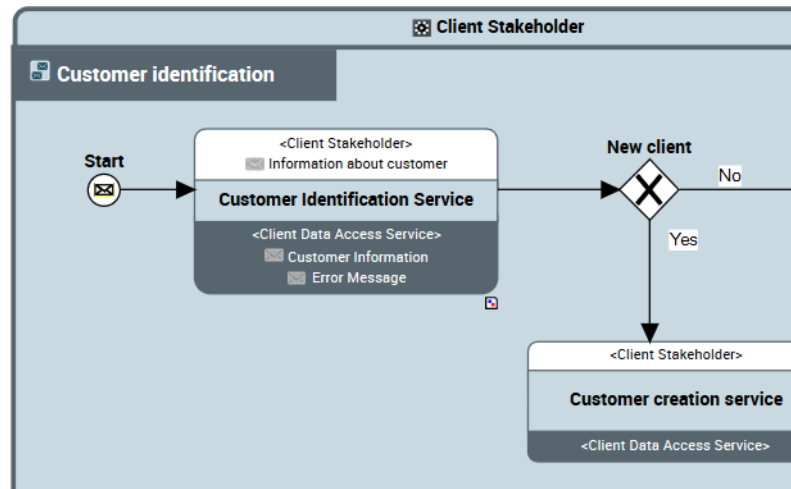


*Service interface Diagram (BPMN) "Customer Identification"*

```
Customer identification protocol starts with a customer
identification step. If the customer is found the service
interface returns customer information, if not, a "customer
creation" service interface is activated.
```

**Example of an advanced service interface communication**
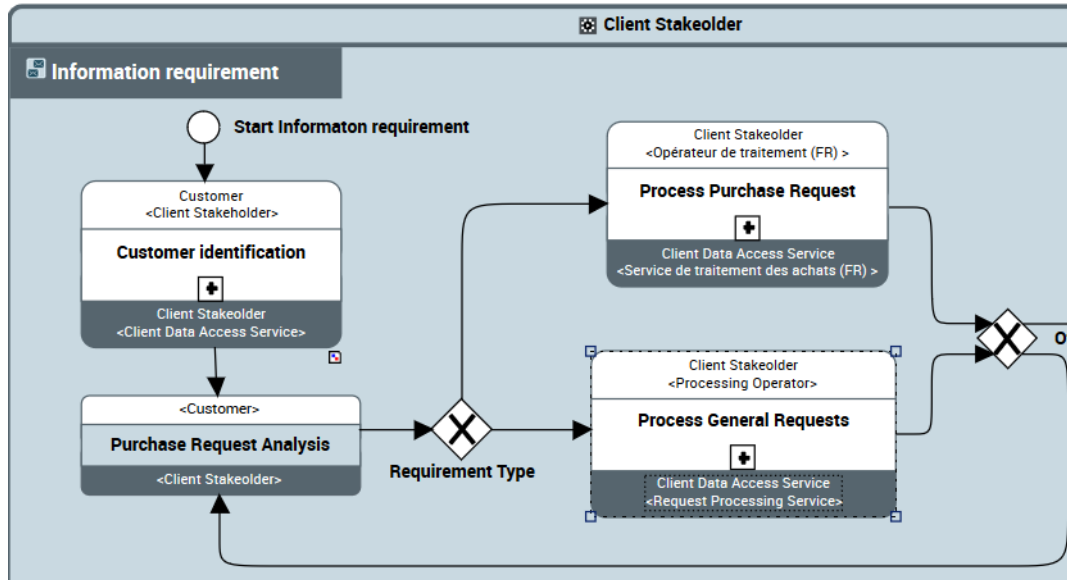


*"Information Requirement" Service Interface diagram (BPMN)*

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

```
This service interface 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.
```

☛ *The **Roles** property page provides access to the list of contributor roles and to the initiator role of a service interface.*

# Accessing the list of service interfaces

To access the list of service interfaces of a library:

1. From the **Environment** navigation menu, open the exploration area **Container > Libraries**.
2. Unfold the desired library, and then the **Service interfaces** folder. The list of service interfaces accessible from the library appears.

# Creating a service interface

You can create a service interface:

- from a library,
- from a diagram using service interactions, for example.

Whatever the point of origin, you can create service interface in standard mode or using a *service interface template*.

> ☞ *For more details on service interfaces, see Using a Service Interface Template.*

## Creating a service interface in standard mode from a diagram

To create a service interface in standard mode, in a diagram, from a service interaction:

1. In the objects toolbar for a diagram, click **Service interaction** ⬱
2. Draw a link between the two communication entities.
3. In add service interaction window, click the arrow at the right of the field **Service interface** and select **Create a service interface**.
   The creation window appears.
4. Select the **Creation Mode**: **Standard Creation**.
   > ☞ *For more details on service interface template use, see Creating a service interface from a service interface template.*
5. Enter the service interface name in the **Name** field.
6. Click **OK**.
7. In the service interaction creation dialog box, enter the name of the service interaction using the name of the service interface and click **Add**.
   The service interaction and the service interface are created.

# Building a Service Interface Diagram (BPMN)

## Creating a Service Interface Diagram (BPMN)

A service interface is represented by a **Service Interface Diagram (BPMN)**.

To create a Service Interface Diagram (BPMN) from an interaction service:

1. From the **Environment** navigation menu, open the exploration area **Container > Libraries**.
2. Unfold the desired library, and then the **Service interfaces** folder.
   The list of service interfaces accessible from the library appears.
3. Select the associated service interface and, in its pop-up menu, click **Create Diagram**
4. In the dialog box, select **Service Interface Diagram (BPMN)**
   The diagram opens with service interface frame and the two *roles* representing consumer and the supplier.
   > 📖 *A role is a participant in an interaction service, 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 a Service operation or a Service interface

In a service interface diagram (BPMN), operations are described by:

- *Service interfaces used*
- *Service operations used*

📖 *A service interface use is associated to a service interface. It enables representation of complex exchanges.*

📖 *A service operation use represents the usage of a service operation in an service interface.*

To create a used *service interface used*:

1. Select the **Service Interface Used** 🖨 button and click in the diagram within the service interface frame.
   The service interface appears in the diagram.
2. Open the **Characteristics** property page of the service interface.
3. Click the arrow to the right of the **Specification of a service interface used** box.
4. Select **Connect service interface** from the drop-down list and choose the service interface that you want to use.

   ☛ *The* **Service operations** *page provides access to the list of components of the exchange contract.*

# DESCRIBING A SERVICE OPERATION

The content of a service interaction is described by a *service interface*.

    📖 *A Service Interface is a template of a contract between entities (organizational, IT …). The contract is described by available operations which can be triggered trough messages exchanged by roles (vendor, buyer..).*

    ☛ *For further detail on service interfaces, see Describing a service interface.*

A service interface is described by a sequence  of service operations or service interfaces.

    📖 *A service operation specifies exchanges between participants.*

A service operation diagram describes the sequence flows of a *service operation*.



*"Customer Identification Service" Service Operation Diagram*
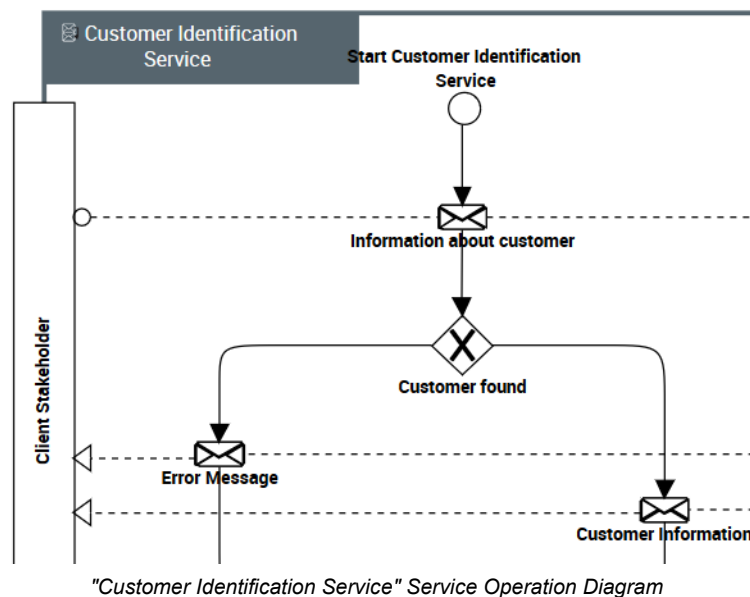
```
The customer identification service protocol begins by
sending information enabling identification of the
customer. An error message appears if the customer is not
found, otherwise customer information is sent (customer
identification, status of orders, etc.).
```

# Accessing the list of service operations

To access the list of service operations of a library:

1. From the **Environment** navigation menu, open the exploration area **Container > Libraries**.
2. Unfold the desired library, and then the **Service operations** folder.
   The list of service operations accessible from the library appears.

# Creating a service operation

You can create a *service operation* a service interface diagram (BPMN).

> ☛ *For more details on service operation templates, see* Using a Service Interface Template.

To create a *service operation* from a service interface diagram (BPMN):

1. Select the **Service Operation Used** ⇨ button  and click in the diagram within the service interface frame.

   > 📖 *A service operation use represents the usage of a service operation in an service interface.*

   The service operation appears in the diagram.
2. Open the **Characteristics** property page of the service operation.
3. Click the arrow at the right of the **Service operation specification**  and select **Create a service operation**.
   The Creation of Operation Joint Action dialog box opens.
4. Enter the **Name** of your service operation click **OK**.
   The service operation is automatically created.

# Describing a Service Operation

## Creating a Service Operation Diagram (BPMN)

A *service operation* is described by a service operation diagram presenting the sequence flow of messages exchanged.

To create a service operation diagram:

1. From the **Environment** navigation menu, open the exploration area **Container > Libraries**.
2. Unfold the desired library, and then the **Service operations** folder.
3. Select service operation that interests you and click **Create Diagram**.
4. In the dialog box, select **Service operation Diagram (BPMN)**
   The diagram opens. The frame of the service operation is positioned and the two roles (Consumer and Provider) are created.

## Creating a message flow with content

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

> 📖 *A message translator is a communication step that translates a message from a format to another. It can be used for trans-codification, data type conversion.*

> 📖 *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 service operation 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 Flow** 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 description of the service assured by the service interface.

> 📖 *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 service 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*

# USING A SERVICE INTERFACE TEMPLATE

*Service interface templates* as well as *Service operation templates* and *content templates*  simplify the service Interfaces creation by duplicating the components of the model used.

Then, the service interface be updated or modified.

## Presentation  of standard service interface Templates

Service interface templates are provided to simplify the creation of your service interfaces. These service interfaces are supported by service operation templates.

Some *service interface templates* are provided with the solution.

### The service interface template "One way communication"
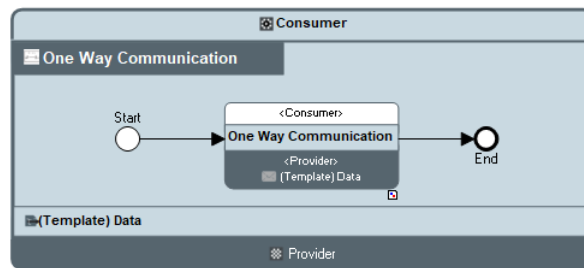


*"One way communication"  service  interface template diagram (BPMN)*

This *service interface* is based on an  *service operation used* noted  "One way communication" between the consumer and the provider.

&#x1F4D6;  *A service operation use represents the usage of a service operation in an service interface.*

The *service operation used* represents the content "(Template) Data" exchanged between the consumer and the provider.
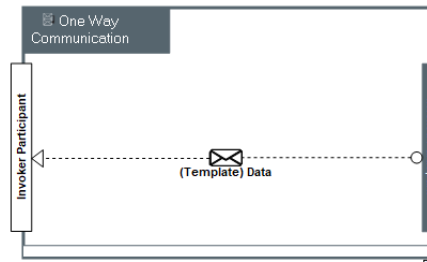


*"One way communication" service operation diagram (BPMN)*

## The service interface template "Request-Response"
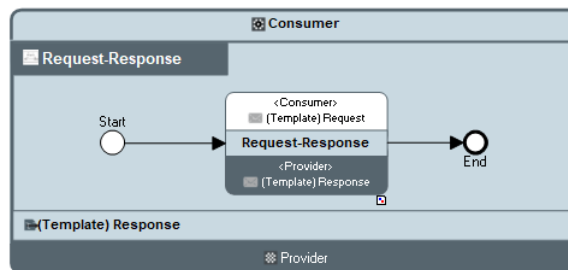


*"Request-Response" service interface template diagram (BPMN)*

This *service interface* is based on an *service operation used* noted "Request-Response" between the consumer and the provider.

📖 *A service operation use represents the usage of a service operation in an service interface.*

The *service operation used* represents the service operations of the contents "(Template) Request" and "(Template) Response" exchanged between the consumer and the provider.
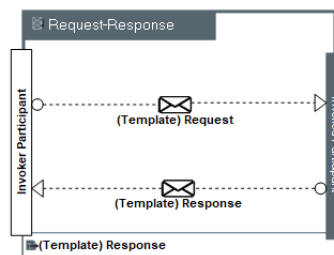


*"Request-Response" service operation diagram (BPMN)*

```
This service operation represents the sending of a request
content and the sending of the response content.
```

**The service interface template "Publish-Subscribe"**
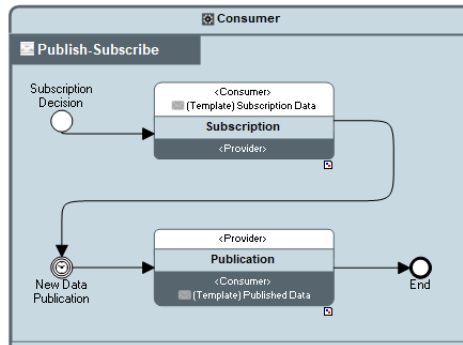


*"Publish-Subscribe" Service interface Diagram (BPMN)*

This *service interface* is based on a *service operation used* noted "Publish-Subscribe" between the consumer and the provider. The request for subscription is sent. An event represents the waiting time before the acceptance for publication.

# Accessing the list of service interface templates

To access the list *service interface templates* of a repository:

❱ From the **Administration** navigation menu, select **Templates > Service Interfaces**.
The list of service interface templates appears.

In the same way, to access to the list of *service operation templates*:

❱ From the **Administration** navigation menu, select **Templates > Service Operations**.
The list of service operation templates appears.

Furthermore, to access the list of *content templates*:

❱ From the **Administration** navigation menu, select **Templates > Contents**.
The list of content templates appears.

# Creating a service interface from a service interface template

To create a service interface from a list using a service interface template:

1. From the **Environment** navigation menu, open the exploration area **Container > Libraries**.
2. From the library that interests you, create a Service Interface.
3. In the following dialog box, select the **Creation Mode**: **Template Based Creation**

4. Select the template that interests you and click **Next**.
   A dialog box displays the list of components of the Service Interface.
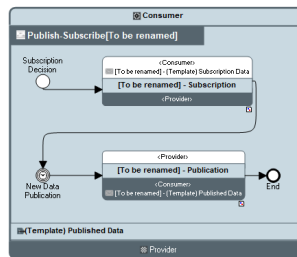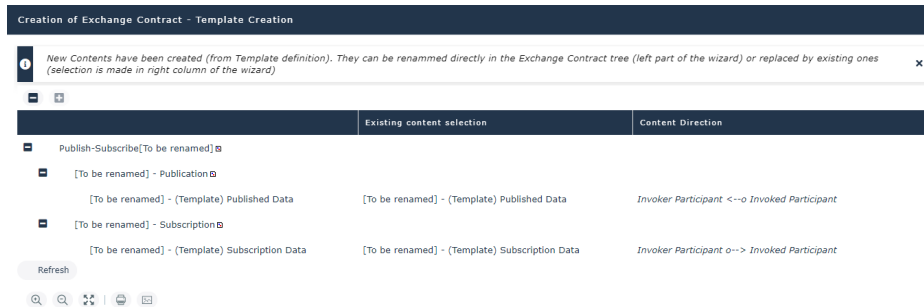




The name of duplicated components is prefixed with "[To be renamed]".
The content templates used are duplicated.

5. Double-click the name you wish to modify.
6. (Option) In the **Existing content selection** column, select the content you want to reuse.
   As a consequence, the created content ""[To be renamed]" is destroyed.
7. Click **OK**.
   The service interface is created.

> ☞ *Then you can change the service interface components, for example from its diagram, see Building a Service Interface Diagram (BPMN).*

# Creating a Service Interface Template

You can use an existing service interface to create service interface template.

To specify that a service interface is a template:

1. Select the service interface that interests you.
2. Open the **Characteristics** properties page.
3. Check the **Interaction Behavior Template** box.
   The service interface is added to the list of existing service interface templates.

The service interface template components declared as templates are duplicated when the service interface template is used.

To access the list of a service interface template components declared as template:

1. From the **Administration** navigation menu, select **Templates > Service Interfaces**.
2. Open the **Template Definition** property page of the Service Interface that interests you.
3. Check the  **Template** box of the components to be duplicated.

## Creating a Service Operation Template

To specify that a service operation is a template service operation:

1. Select the service operation that interests you.
2. Open the **Characteristics**properties page.
3. Check the **Interaction Behavior Template** box.
   The service operation is added to the list of existing service operation templates.

   ☛ *To access to the list of service operation templates: from the **Administration** navigation menu, select **Templates > Service Interfaces**.*

# HOPEX IT ARCHITECTURE REPORTS

**Hopex IT Architecture** provides facilities for analyzing and tracking the changes implemented in the IT Infrastructure of your architecture. **Hopex** Suite uses reports to group sets of repository objects and study their interactions.

> ☛ *For more details on reports, see the **Hopex Common Features** guide, "Generating Reports".*

Report templates proposed as standard by **Hopex IT Architecture** offer various analysis presentation possibilities. Some reports are shared with other solutions, for example **Hopex IT Business Management**.

Furthermore, the **Exploded Diagram Reports** are available on several types of object. This type of report enables the building of a summary view of a complex object architecture into one single diagram ; it consists in the generation of an exploded diagram view of a complex object by inclusion in a diagram describing the root object. The diagrams describing the objects mentioned in the source diagram, recursively, so that several diagramming levels can appear in one picture.

> ☛ *For more details, see "Launching the exploded diagram report" chapter in the **Hopex Common Features** guide.*

# APPLICATION ARCHITECTURE REPORTS

## Technical Architecture Matrix

This report displays the distribution of IT Service of an Application on its Technical Areas (Application or Data).

### *Report example*
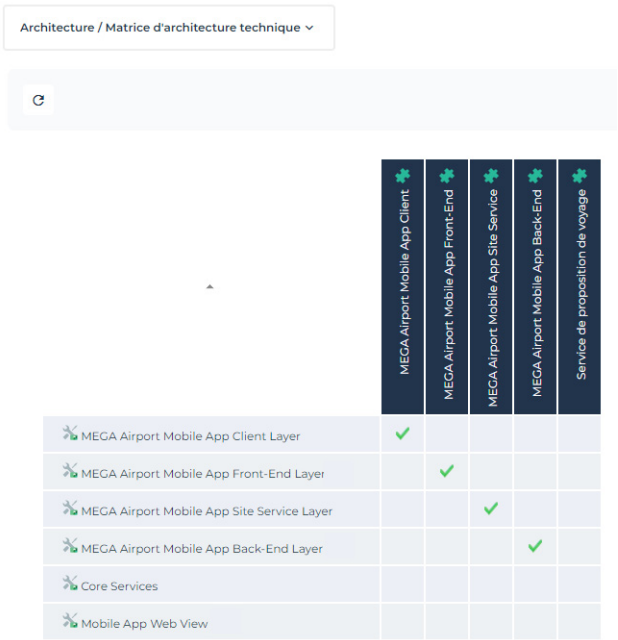


### *Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application. | One object mandatory. |

# Application Exchange Density

This report displays the density of exchanges around an application in order to help defining application systems.

The lines display the fact that there is at least one content exchanged between two applications.

The color of the line indicates the number of exchanged contents.

- Gray: 1 or 2 contents,
- Green: between 3 and 5 contents,
- Orange: between 6 and 10 contents,
- Brown: between 11 and 15 contents,
- Red: more than 15 contents.

### *Report example*
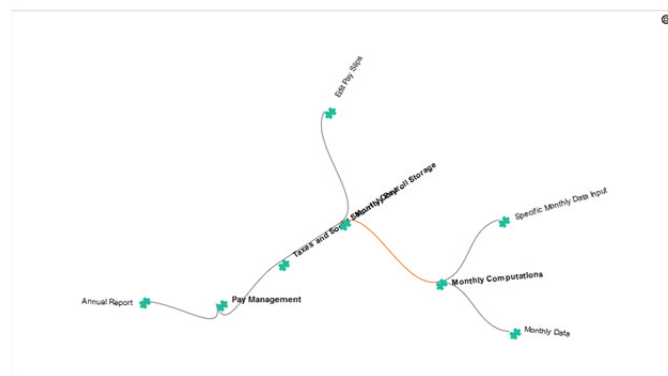


### *Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application system, Application or IT service | One object mandatory. |
| Depth | Integer | |

# Exchange Consistency Structure Scenario

This report allows to check exchange consistency between scenario and structure descriptions (via Application flows or Service Interactions).

It analyzes every content sent/received by an agent via an application flow and checks presence of equivalent service interactions and vice versa.

### Exchange Consistency Structure Scenario report example



### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application system, Application or IT service | One object mandatory. |

## Content Consistency (Structure)

This report allows to check exchange consistency between "external" views and "internal" views, limited to structure descriptions (Service Interactions).

A Service Interface between the analyzed object and another one (via a Service Interaction) in an "external view" must be established also with a component of the analyzed object (via another Service Interaction) in an "internal view". And vice versa.

> ☞ *All Service Interactions/Service Interfaces with another agent must be handled by an internal component and all Service Interactions/ Service Interfaces handled by an internal component must be used by another agent.*

### Content Consistency (Structure) report example



### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application system, Application or IT service | One object mandatory. |

## Content Consistency (Scenario)

This report allows to check exchange consistency between "external views" and "internal views", limited scenario descriptions (application flow).

A content send/received in an "external view" with another agent must appear also in an "internal view" and vice versa.

> ☛ *All flows exchanged with another agent must be handled by an internal component and all flows handled by an internal component must be used by another agent.*

*Content Consistency (Scenario) report example*



*Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application system, Application or IT service | One object mandatory. |

# External Contents Matrix (Structure)

This report displays a matrix of contents sent or received by analyzed agent with other agents, limited to structure descriptions (Service Interactions).

### External Contents Matrix (Structure) example
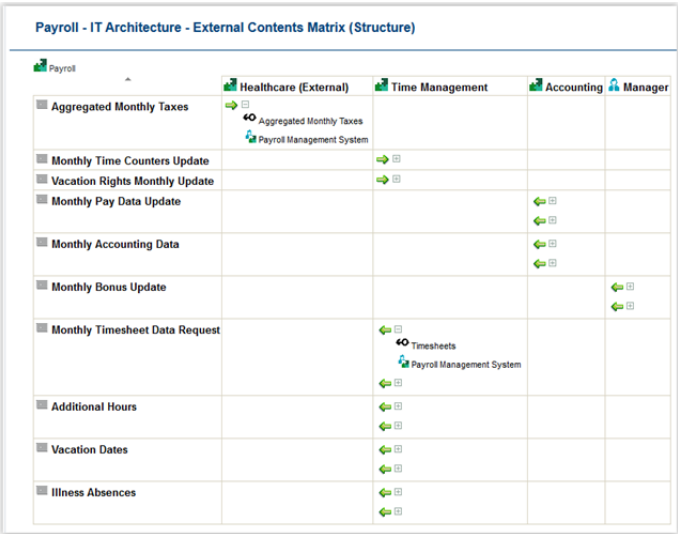


### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application system, Application or IT service | One object mandatory. |

# External Contents Matrix (Scenario)

This report displays a matrix of contents sent or received by analyzed agent with other agents, limited to scenario descriptions (Application Flows).

*External Contents Matrix (Scenario) example*
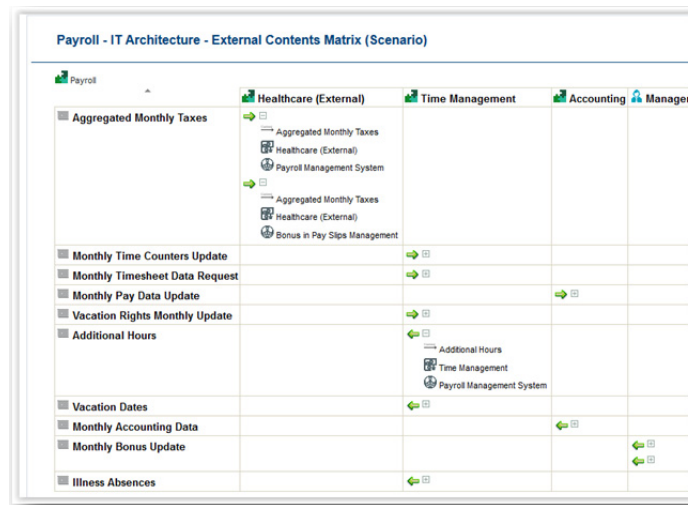


*Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application system, Application or IT service | One object mandatory. |

# External Service Interface Matrix

This report displays a matrix of Exchanged Contracts used by the analyzed object to service with other agents.

### *External Service Interface Matrix Example*
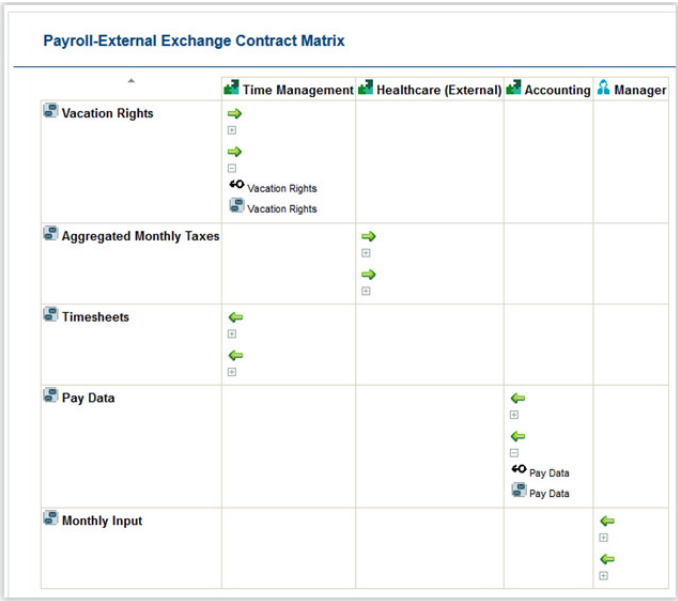


**Payroll-External Exchange Contract Matrix**

### *Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|------------|----------------|-------------|
| Root object | Application system, Application or IT service | One object mandatory. |

# Graph of Flows between Agents

This graph report displays a synthesis of all interactions between some agents (Application System, Application, IT Service, Microservice etc.) selected by the user.

Filters can be applied on display by selecting some service interfaces and/or service interaction contexts.

Report can be visualized in 2D or 3D.

***Graph of Flows of between Agents Example***
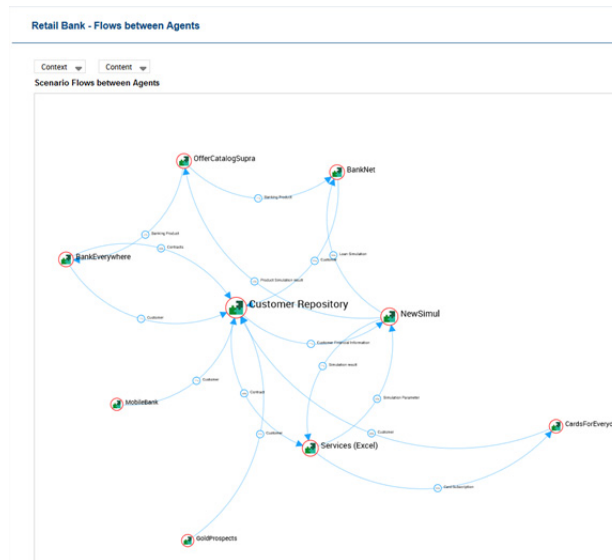


***Report parameters***

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|------------|----------------|-------------|
| Root object | Application System, Application, IT Service or Microservice. | One object mandatory. |

# Graph Flows of an Agent

This graph report displays a synthesis of all application flows exchanged by an agent (Application System, Application, IT Service, Microservice etc.).

Filters can be applied on display by selecting some exchanged contents and/or flow contexts.

Report can be visualized in 2D or 3D.

### Example of a 2D graph of Flows of an Agent
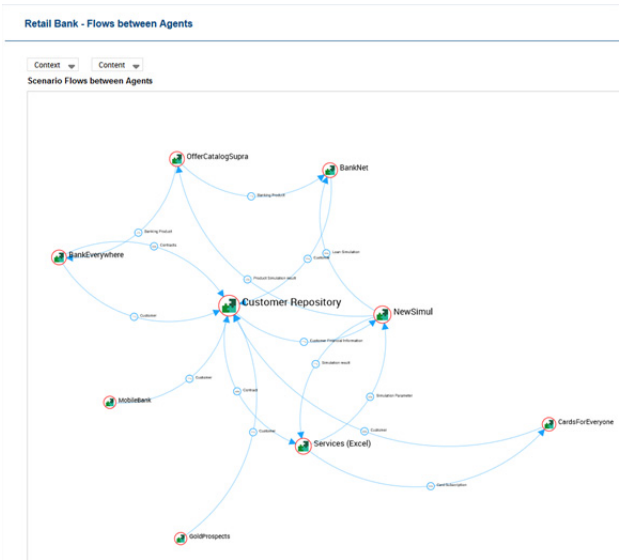


### Example of a 3D graph of Flows of an Agent
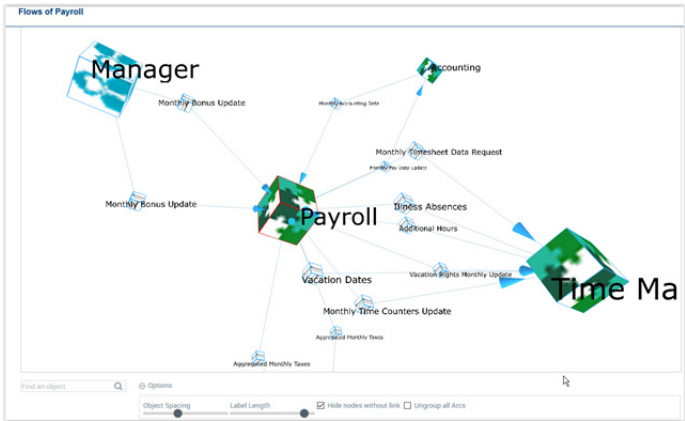


### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
| --- | --- | --- |
| Root object | Application System, Application, IT Service or Microservice. | One object mandatory. |

## Flow Process Rationalization

This graph shows the distribution of multi-software communication chains. It allows to quickly identify the contents with the most associated communication chains and therefore potentially the least urbanized in terms of flows.

### *Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Communication System | One object mandatory. |

## Graph of Service Interactions between Agents

This graph report displays a synthesis of all service interactions between some agents (Application System, Application, IT Service, Microservice etc.) selected by the user.

Filters can be applied on display by selecting some service interfaces and/or service interaction contexts.

Report can be visualized in 2D or 3D.

*Graph of Service Interactions between Agents Example*



### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application System, Application, IT Service or Microservice. | One object mandatory. |

## Graph of Service Interactions of an Agent

This graph report displays a synthesis of all service interactions between some agents (Application System, Application, IT Service, Microservice etc.) selected by the user.

Filters can be applied on display by selecting some service interfaces and/or service interaction contexts.

Report can be visualized in 2D or 3D.

*Example of a graph of interactions of an Agent*
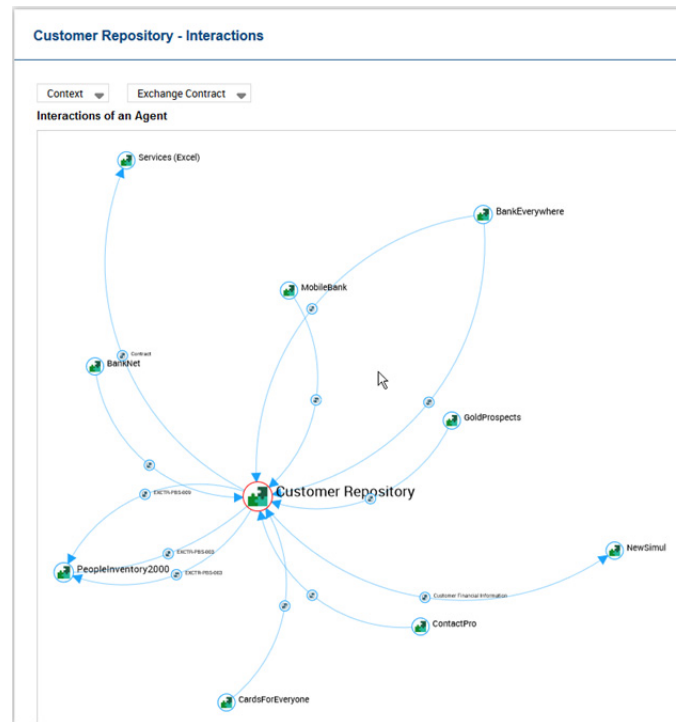


*Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application System, Application, IT Service or Microservice. | One object mandatory. |

# REPORTS ON THE ARCHITECTURE FUNCTIONAL COVERAGE

## Building Block Breakdown report

This report aims at detailing the breakdown of a root Class of Building Block object into its Class of Block Component object and emphasizing the realizing EA artifacts of each component.

- The depth of analysis can be defined,
- The types of analyzed components can be displayed or filtered out (EA dimension),
- The types of realizing items can be displayed and filter out by types (EA layers),
- The look and feel can be fine-tuned (color palette, number of displayed columns).

☛

☛ *For more details on use of breakdown report, see "Handling a Breakdown report" in the* ***Hopex Common Features*** *guide.*

### Report examples

The example below enables viewing of the functional breakdown of the functionality map specified as parameters.
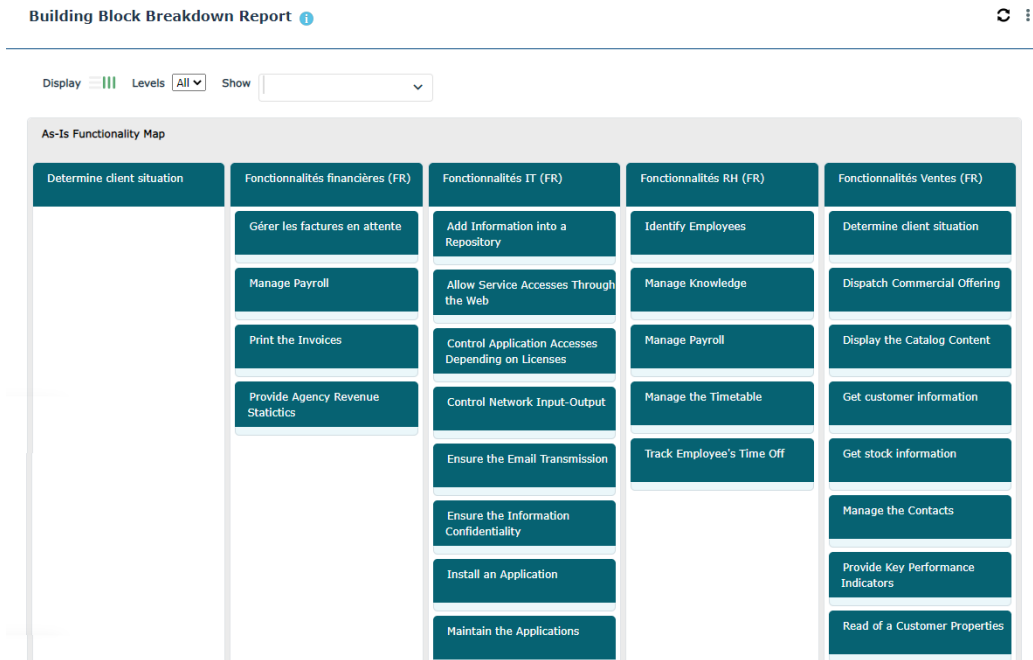


☛ *Example of functionality breakdown report.*

In the example below, the applications that implement the functionalities are presented.
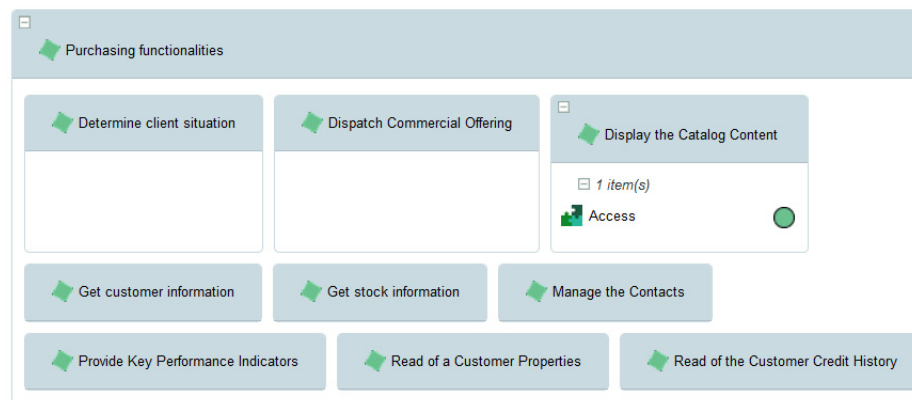
Another representation helps to see the capability maps fulfilled by applications.

### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Building block | One object mandatory. |
| Depth level | Short | Defines the breakdown level of the business capability map or the capability entered as a parameter. |
| Number of columns | Short | Defines the number of columns displayed by breakdown level (for eg. 2 or 3) |
| Color palette | HOPEX palette | Mandatory.<br>The palette delivered by default is "BoxInBox Report Monochrome Grey" |
| EA Level | Multiple choice:<br>- business function level,<br>- organizational level,<br>- application level,<br>- technical level. | Define which objects of which type of architecture level are displayed for capability realizations;<br><br>*For example, activation of the "applications level" displays the business capability realizations for the Application System Environment, the Application Systems or the Applications* |
| EA dimension | Multiple choice:<br>- capability models,<br>- agent models,<br>- process model,<br>- information models,<br>- performance models,<br>- results models | Define which types of objects are examined within the framework of the breakdown analysis<br><br>*For example, activation of "capability models" will display the business skills or functionalities required by the capabilities that are broken down* |

☛ *For more details on how to associate a business capability with a functionality, see Describing Fulfillment of a Business Capability.*

# Overlapping Applications

This report presents a matrix of application systems, applications, IT Services and Microservices that have the same functional perimeter as the described element.

☛ *For more details on how to associate a concret element with a functionality, see Describing the fulfillment of a Functionality.*

### Overlapping Applications report example



### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application System, Application, IT Service or Microservice. | One object mandatory. |

# Business Capability Breakdown Report

You can use this report to display the realization coverage of business capability elements by operational elements such as logical and physical applications, application systems, etc.

> ☞ *For more details on how to associate a business capability with an application, see Creating Fulfillment of a Business capability.*

> ☞ *For more details on use of a breakdown report, see Building Block Breakdown report*

### Report examples

The example below enables viewing of the coverage rate of
the capability map specified as parameters.
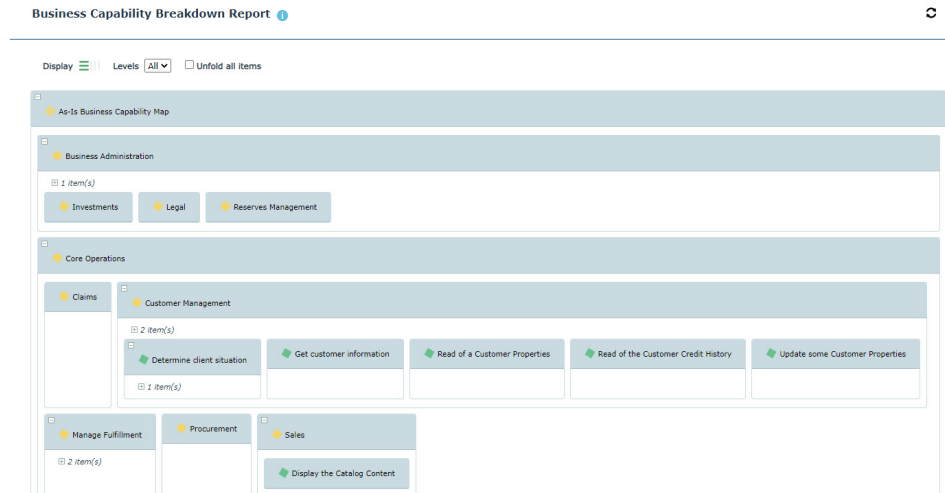


The example below shows how the functionalities associated
with capabilities are implemented by application
components.

☞ *For more details on how to associate a business capability with a
functionality, see* Describing Fulfillment of a Business Capability.
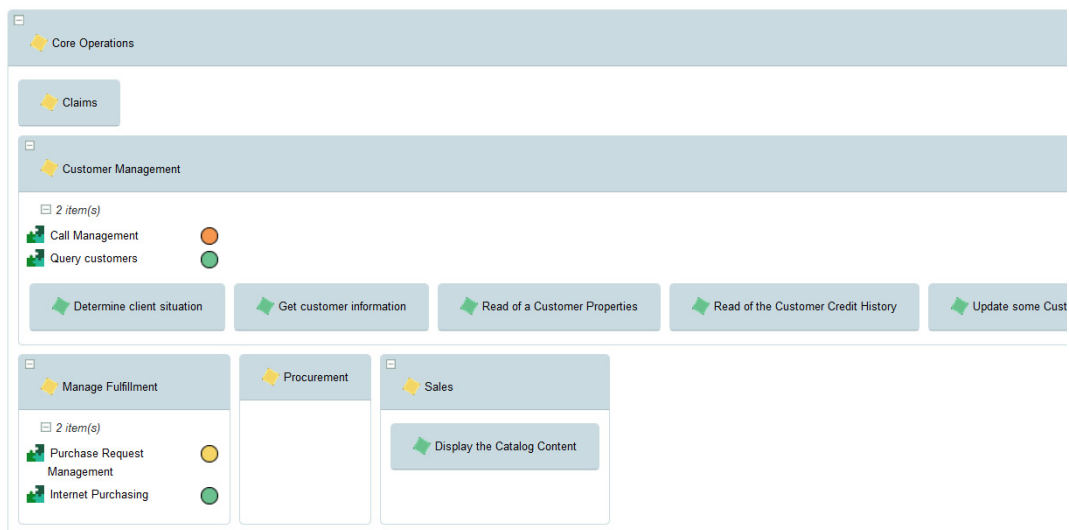
### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|------------|----------------|-------------|
| Root object | Business Capability<br>Business Capability Maps | One object mandatory. |

☛ *For more details on how to associate a business capability with a functionality, see Describing Fulfillment of a Business Capability.*

# INFRASTRUCTURES REPORTS

## Infrastructure Description Report

The report displays a description of the infrastructure and lists the defined communication channels between components.

☛ *For more details, see Modeling IT Infrastructures.*

### Report example
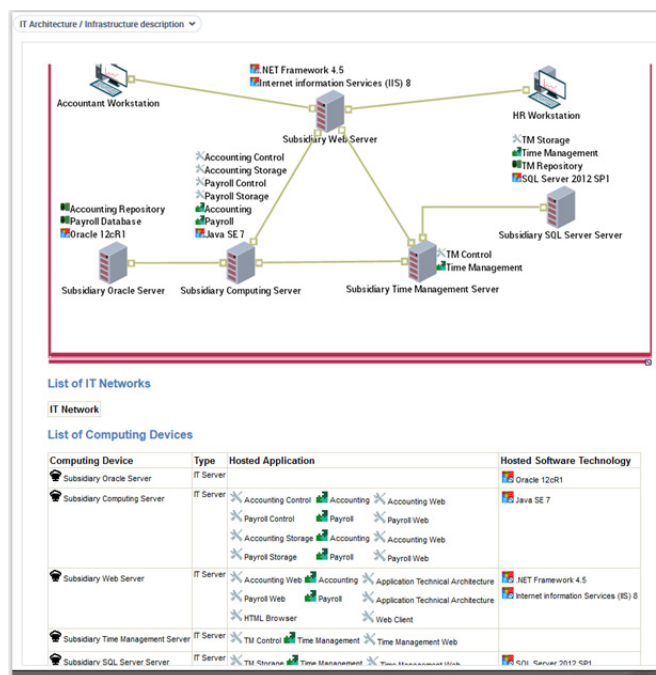


### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | IT Infrastructure or IT Network | One object mandatory. |

# Application Technology Requirements x IT Infrastructure Provided Technologies Matrix

This report compares Technology requirements from Technical Architecture of an Application and Technology provided by IT Infrastructure of same Application.

- Green color indicates a compliance between requirement and a hosting device.
- Red color indicates a non-covered requirement by the device.
- Orange color indicates device capability that is not required.

### *Report example*



### *Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application | One object mandatory. |

# Network Channel x Service Interactions

This report displays a matrix of the Network Channels x the Applicative Interactions.

## *Report example*



## *Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | IT Infrastructure or IT Network | One object mandatory. |

# Network Channel x Package Connection Matrix

This report displays a matrix of the Network Channels x Package Connections.

## *Report example*
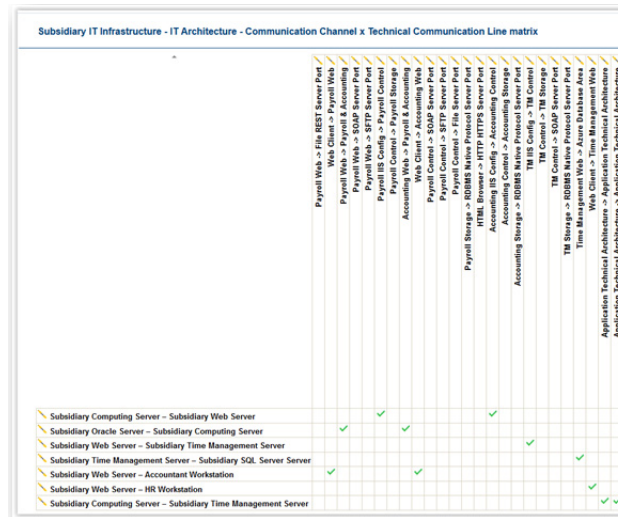


## *Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | IT Infrastructure or IT Network | One object mandatory. |

# DEPLOYMENT ARCHITECTURE REPORTS

## Deployment Architecture Report

This report displays in the form of tables, the characteristics of the following events:
- Deployable Packages
- Package Connections,
- Prescribed Hosting Devices

### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application deployment architecture<br>Application system deployment architecture | One object mandatory. |

## Deployment architecture matrix

This report displays the distribution of IT Service of an Application on its Technical Areas (Application or Data).

### Report example



### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application | One object mandatory. |

# Package Connection x Service Interactions Matrix

This report allows to analyze support of Service Interactions by Package Connections.

Checkmark indicates presence (or not) of a link between a Service Interaction and a Package connection in a Deployment Architecture.
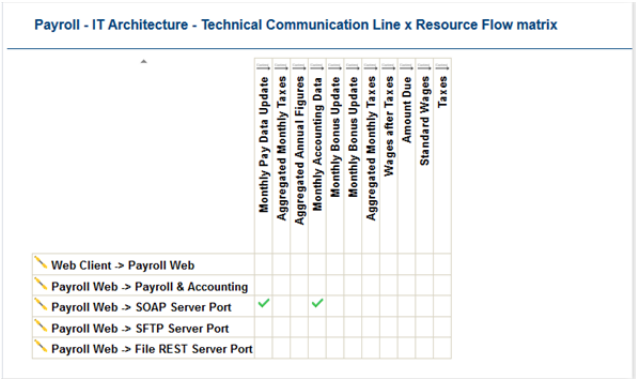
*Report example*



*Report parameters*

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
| --- | --- | --- |
| Root object | Application deployment architecture<br>Application system deployment architecture | One object mandatory. |

# Package Connection x Resource Flow Matrix

This report allows to analyze support of application flows by package connections.

Checkmark indicates presence (or not) of a link between an application flow and package connections of the analyzed Deployment Architecture.

### Report example



### Report parameters

This consists of defining report input data.

| Parameters | Parameter type | Constraints |
|---|---|---|
| Root object | Application deployment architecture<br>Application system deployment architecture | One object mandatory. |

# UML modeling

# ABOUT UML IMPLEMENTATION

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

> ☛ *To access UML functions, you must be connected **Solution Architect** profile or with **Solution Architecture Functional Administrator** profile.*

The aim of this part is to introduce you to the main UML functionalities provided by **Hopex IT Architecture**.

# OVERVIEW

The facilities provides by **Hopex IT Architecture** for UML modeling are described below.

## 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 .

## 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 .

## 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:
-
-

## 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", page 357.

## 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", page 328.

## 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", page 331
- "Composite Structure Diagram", page 335

## Specifying deployment

Implementation of objects in an concrete working  environment can be specified in the **deployment diagram**.

See "The deployment diagram", page 375.

# 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.).
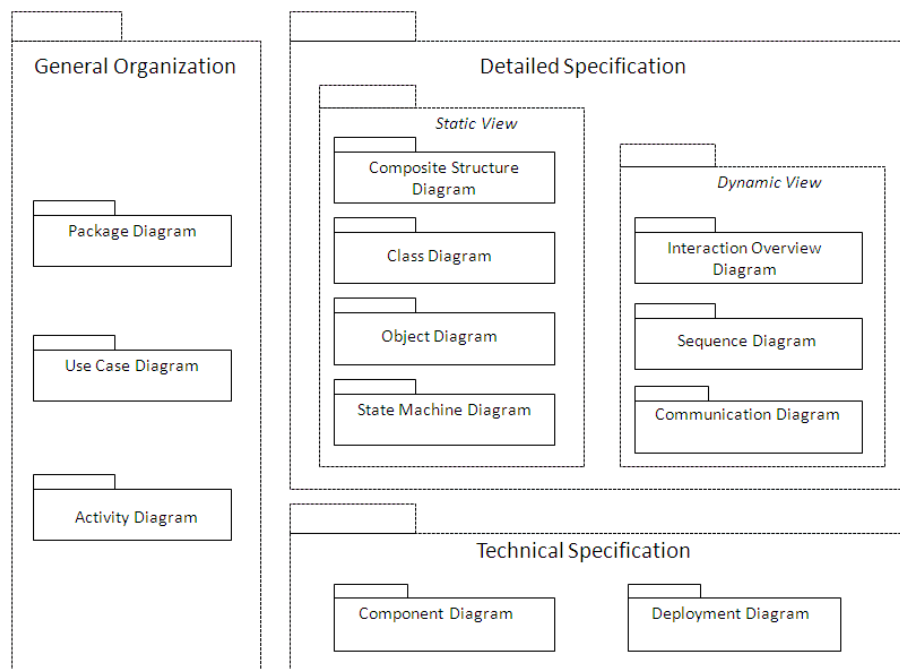


## Detailed specification

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 overview diagram | interaction |
| Activity diagram (UML2) | Activity |
| State machine diagram | State machine, , protocol state machine |

In **Hopex IT Architecture**, the entry points above are accessible in **Design (UML)** navigation menu.

# 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:

        ✓   Creating a Use Case Diagram
        ✓   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 IT Architecture** , you can also create a use case diagram for the application environment of a project. See Creating an application Use Case Diagram.

## Creating a Package

To create an interaction with **Hopex IT Architecture** using **Design (UML)** navigation menu:

1. Click **Packages** sub-menu.
2. In the edit area, click **New**.
   The Creation of Package dialog box appears.
3. Enter its **Name**.
4. 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.*

5. 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. Select the package stream that interests you and click **New Diagram**.
2. In the dialog box, select **Use Case Diagram**.
   The diagram opens in the edit window. The frame of the package is positioned within the drawing.

# USE CASE DIAGRAM ELEMENTS

✓  Actors
✓  Use Cases
✓  Packages
✓  Participations
✓  Use Case Associations: Extensions and Uses
✓  Generalizations
✓  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:

Client         Sales representative         Warehouse manager         Shipping department
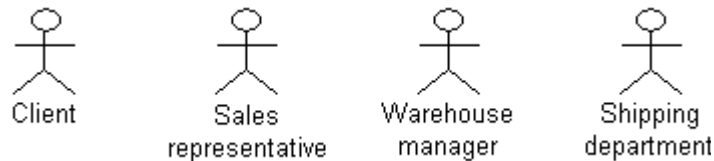
To create an *actor* in a use case diagram:

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.

    ☺  *You can create several elements successively without clicking in the toolbar each time by double-clicking the* **Actor** *button.*

    ☺  *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.
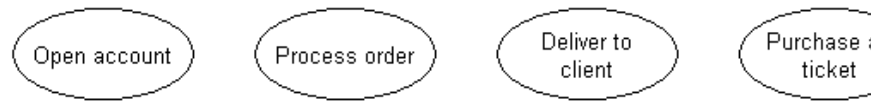


To create a use case in a diagram:

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.
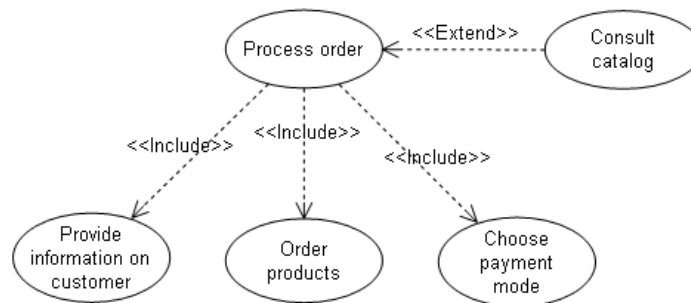


☺ *Zooming in on the description of an object is possible for all elements described by a diagram.*

# 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 *package*: the commercial information system, accounting, production management, digital control of a machine, an alarm system, etc.
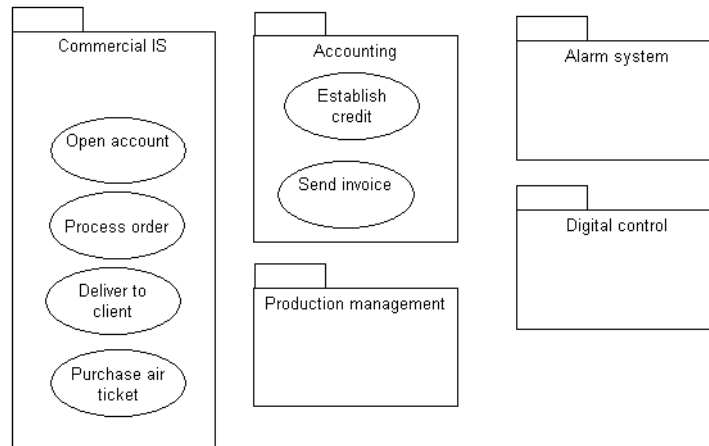


You can create a package using the **Package** ⬚ button in the toolbar. You can then increase its size in order to place use cases within it.

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
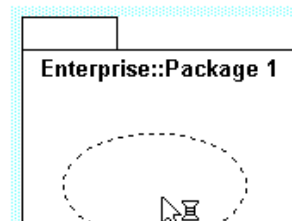


☛ *If the linked objects disappear under the package, click the package and select the **Send to Back** button in the Edit toolbar.*

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
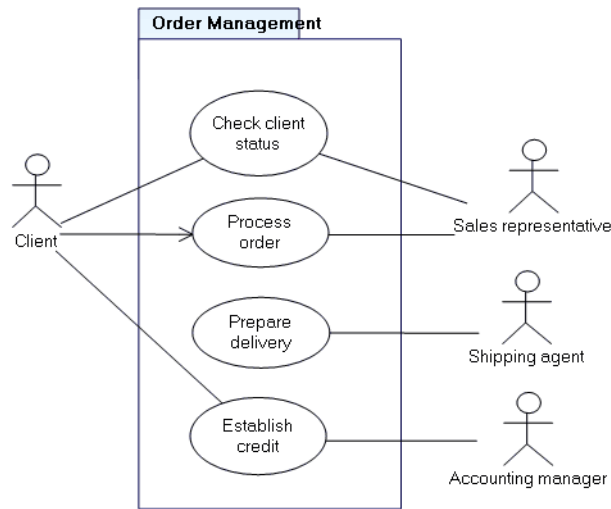


*   The sales representative participates in order processing and in checking client status;
*   The shipping agent participates in delivery;
*   The accounting manager participates in setting up loans, etc.

## 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.

> 💣 **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 form 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.

# Use Case Associations: Extensions and Uses

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".
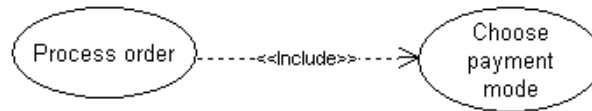


### *Examples of inclusion*

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
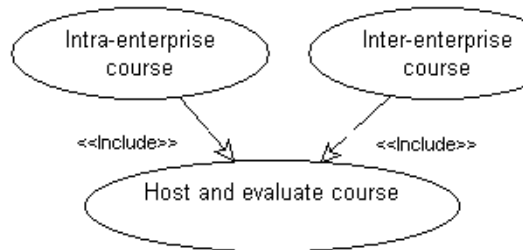


In a company doing direct sales, the use case:
- Place an order

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".
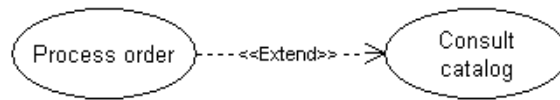


### *Extension example*

The purchase of an airline ticket can also include booking a hotel room or renting a car.
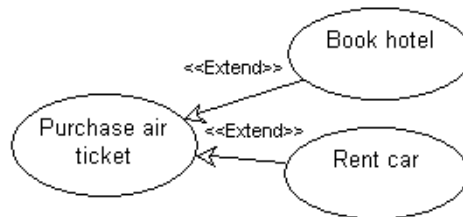


### *Extension point*

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.
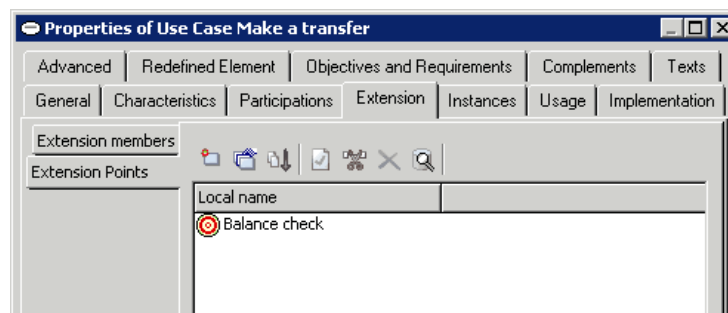
An extension point can be associated with a *constraint* which indicates the moment at which the extension intervenes. You can add a constraint at creation of the extension or later, in the extension properties dialog box.

&#x1F4D6; *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.
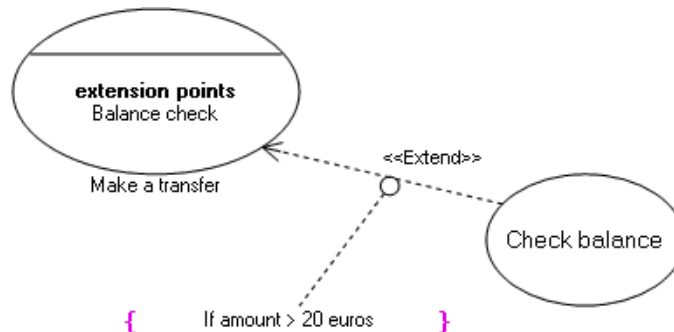


# Generalizations

&#x1F4D6; *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:
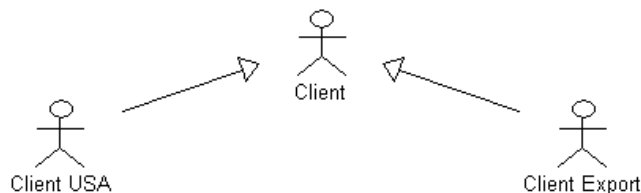


The "Client" actor can have the USA or Export specialization.

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.
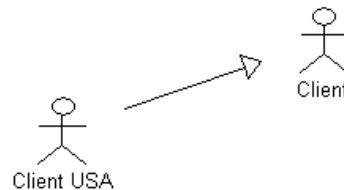


☛ *In the same way you can create a generalization between two use cases.*

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.
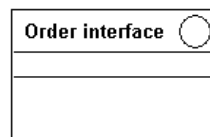


## Connecting an interface to a use case

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.
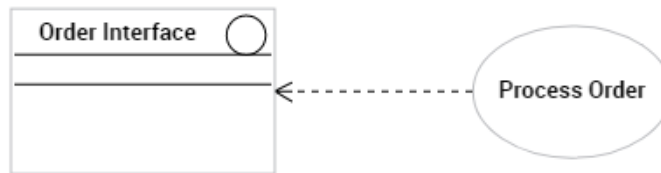


You can detail which operations the use case can carry out via this interface.
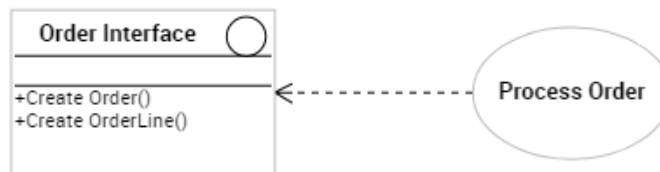


✓ For more details on required and supported interfaces, see Linking interfaces to other objects.

# 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:

- ✓ Presentation of the Class Diagram
- ✓ Creating a Class Diagram
- ✓ Classes
- ✓ Attributes
- ✓ Operations
- ✓ Signals
- ✓ Associations
- ✓ Generalizations
- ✓ Specifying Interfaces
- ✓ Specifying Dependencies
- ✓ Specifying Parameterized Classes
- ✓ Constraints
- ✓ 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 Data 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

## 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.
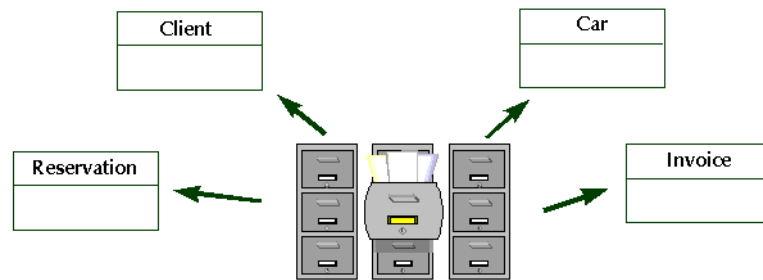


Classes can be technical objects used for programming.

        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.
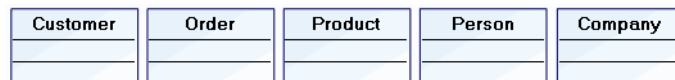


A class can also express a process, such as "Confirm client request", or implement a business rule, such as "Consistency in cost accounts".

> ☞  *See the glossary at the end of this guide for definitions of these and other concepts.*

*284*

# 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.

    ☺ *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 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.
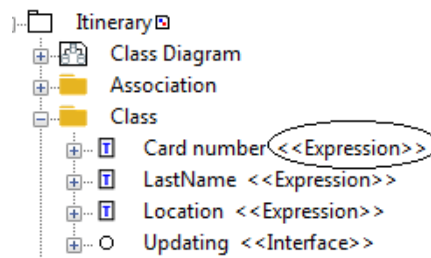


To activate this option:

1. Open the **Options** window.
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

✓  Definition: Attribute
✓  Specifying Class Attributes
✓  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).
```
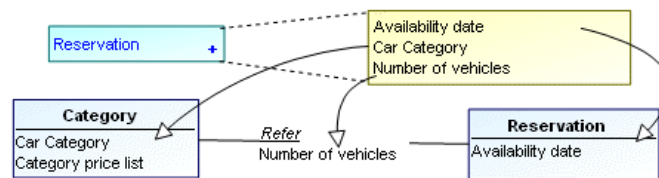


Classes and association classes may be characterized by attributes.

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.
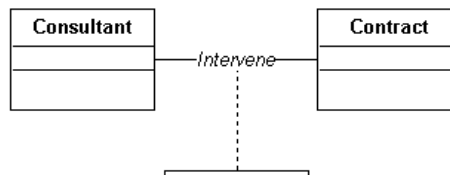
# Specifying Class Attributes

## Creating a standard attribute

To create an attribute on an 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 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.

## Creating 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 create an attribute on an 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** calculated.
   The new attribute appears.
4. Open the properties of the attribute to specify:
   - the list of input parameters
   - the description of the rule

### 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 **Components** property page the holding class, in the **Attributes** section, select the attribute in question.
2. 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

- ✓ Definition of an Operation
- ✓ Specifying Class Operations
- ✓ Operation Properties
- ✓ Operation or Signal Signatures
- ✓ Operation Parameters
- ✓ Operation Methods (opaque behavior)
- ✓ Object Diagram
- ✓ Operation Exceptions
- ✓ 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 Data Architecture** tools (synchronization, generation etc.).*

## Specifying Class Operations

To specify class operations:

1. Select the class concerned and display its properties.
2. In the **Components** property page, expand the **Operations** section, click **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 **Components** property page the holding class, expand the **Operations** section, select the operation in question.
2. 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 | | |
|---|---|---|
| ⊕ New   🔗 Connect   📑 Reorganize   |   💬 Properties   ⊗ Remove   |   ☰ |
| **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:typeexp
ression2,...):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 **Characteristics** property page of the operation.
2.  In the **Method** section, click **Add**.
    The dialog box for adding a method appears.
3.  Enter the name of the method to be created or search for an existing operation.
4.  Click **OK**.

To enter the body of the text and the method that implements the operation:

1.  Open the **Characteristics** property page.
2.  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  **Conditions** property page of the operation.

   **2.** In the **Conditions** section, select the condition type:
- precondition
- condition on the body
- postcondition

   **3.** Click **New**.
The dialog box for adding a restriction appears.

   **4.** Enter the name of the restrictions to be created or search for an existing operation.

   **5.** 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.*
>
> ☺ *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.

    &#x1F4D6; *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*)..

    &#x1F4D6; *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.

    &#x1F4D6; *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.*

    &#x1F4D6; *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.
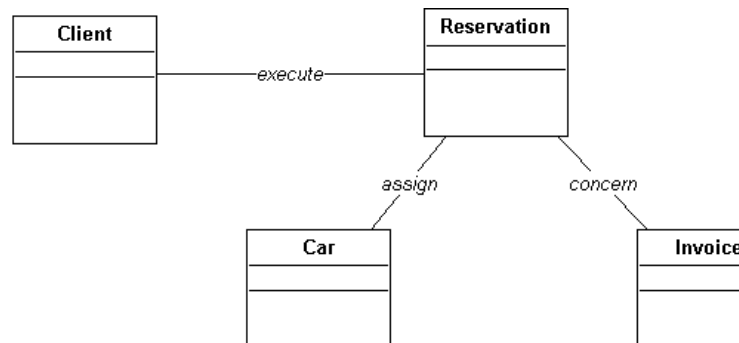


The following drawing provides a three-dimensional view of the situations a class diagram can store.
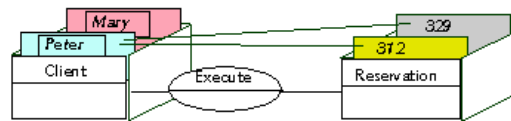


Peter and Mary are clients. Peter has made reservations numbers 312 and 329.

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.
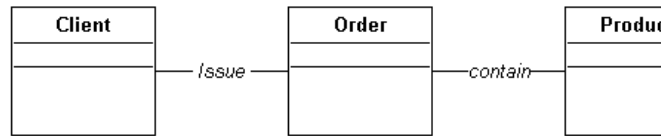


- A person *works* for a company.
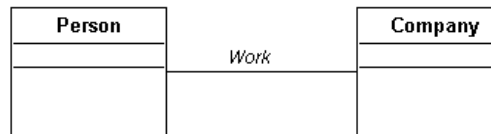


- An alarm is triggered by a sensor.

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*.
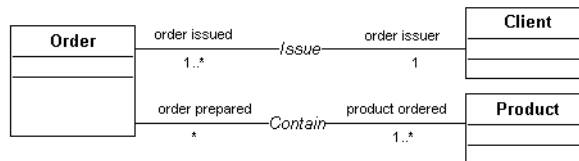


A person is an *employee* of a company.

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.
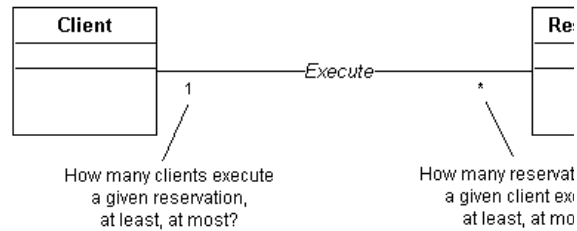


The usual multiplicities are "1", "0..1", "*" or "0..*", "1..*", and "M..N" where "M" and "N" are integers:

- 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:
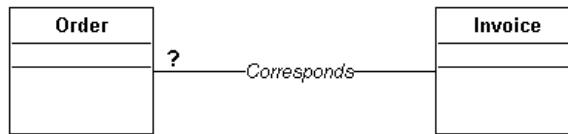


- • 0..1: An order corresponds to zero or at most one invoice.
- • *: No restriction is placed on the number of invoices corresponding to an order.
- • 1: Each order has one and only one corresponding invoice.
- • 1..* : Each order has one or more corresponding invoices.

Other examples of multiplicity:
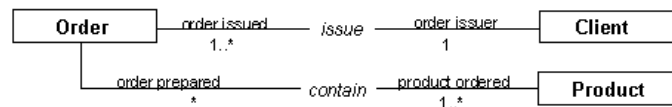


- • 1..* : A client can issue one or more orders.
- • 1: An order is issued by one and only one client.
- • 1..* : An order contains one or more products.
- • *: A product can be contained in any number of orders, including no orders.
- • 0..1 : A person works for a company.
- • 1..* : An alarm is triggered 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.*

| ASSOCIATIONEND | | |
|---|---|---|
| Whole/Part | ▶ | |
| Multiplicity | ▶ | (None) |
| Min Participation | ▶ | * |
| Max Participation | ▶ | 1..* |
| isNavigable | ▶ | ✓ 1 |
| Documentation | ▶ | 0..1 |
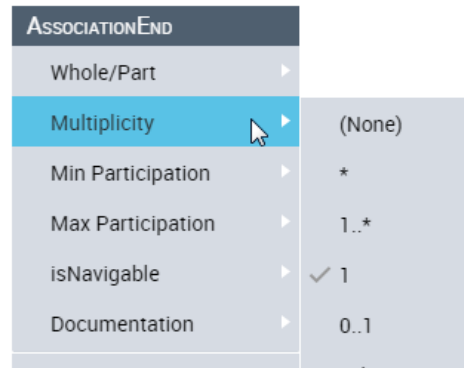
The multiplicity now appears on the role.
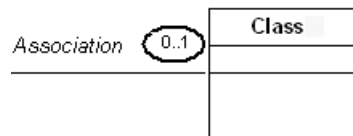
> ☺ *All the information specified with the pop-up menu can also be viewed and modified in the role properties dialog box.*

# 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.
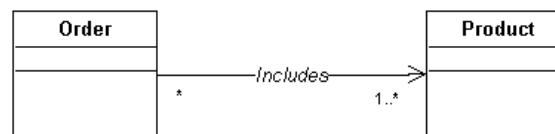
### Specifying navigability for a role

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.
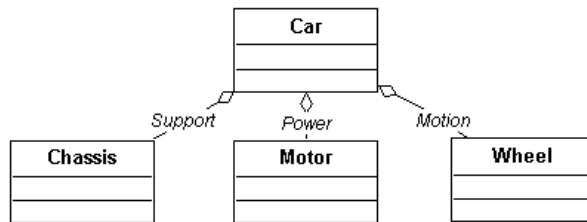


### Specifying role aggregation

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.
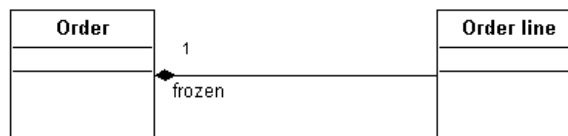


You can indicate whether a role is changeable using the role pop-up menu or the role properties dialog box.

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 is 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.
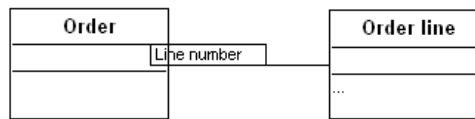


To define a qualifier:

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.
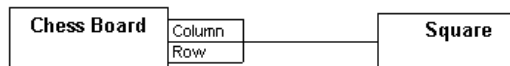
# Overloading a Role

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.
```
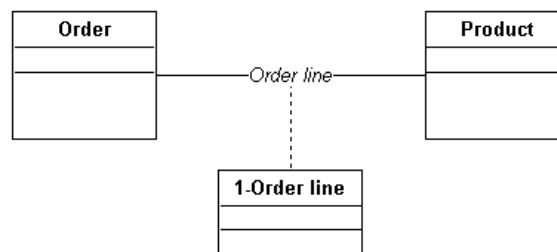


To create an association class:

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.
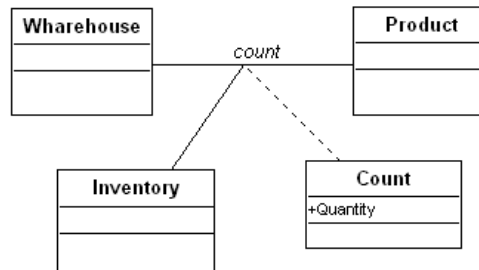


To create a ternary association:

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
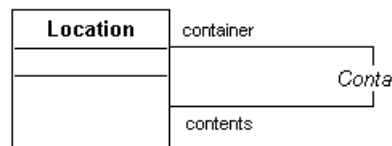


Certain associations use the same class several times.

A classroom, a building, and a school are all locations.
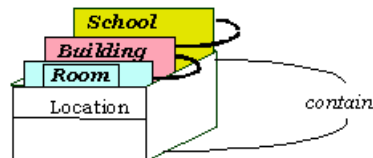


A classroom is contained in a building, which is contained in a school.

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.
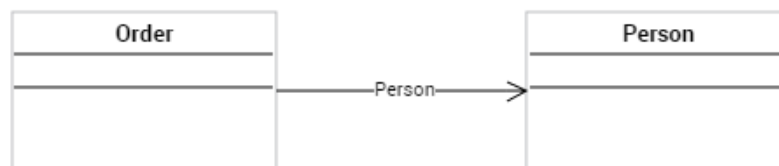


## Creating a Part between two Classes

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 "Oder line" class can be defined from the "Order" class through the "Order line" part.
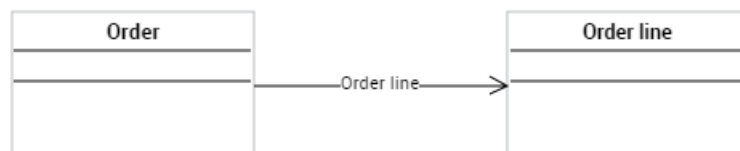


To define the identifier of the "Order line" class:
1. Display the properties of the "Order line" class.
2. Select the **Identifier** page.

**3.** Right-click the **Members** folder and select **Connect** > **Part**.
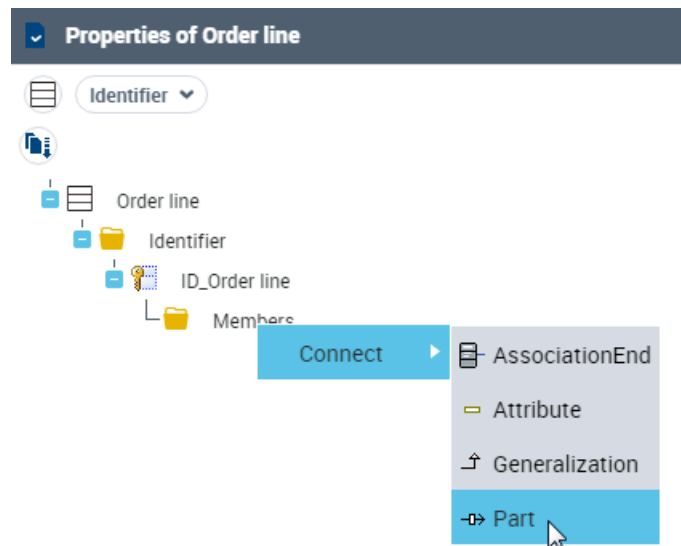


**4.** Select the proposed part.
**5.** Click **OK**.

# 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.
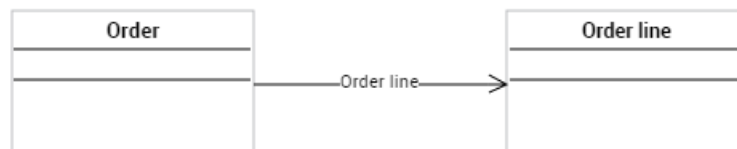


**2.** Select **Multiplicity** then the desired value.

### 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.
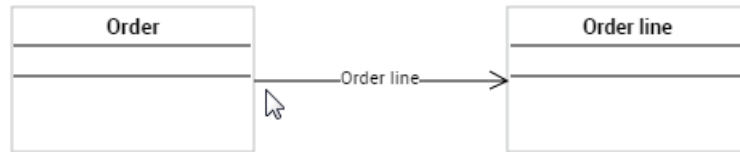


**2.** In the pop-up menu that appears, select **Multiplicity** then the desired value.

---

# 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.*
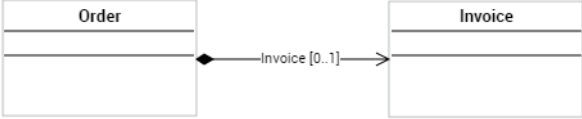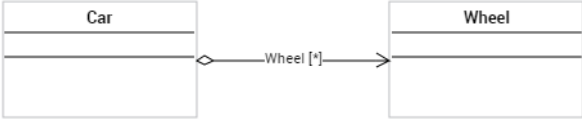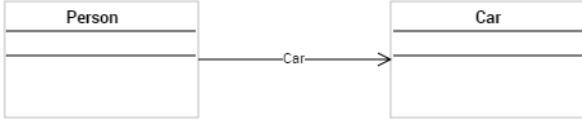
     *A composition is a strong aggregation where the lifetime of the components coincides with that of the composite. A composition is a fixed aggregation with a multiplicity of 1.*

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-sponding multiplicity | Example |
|---|---|---|---|
| Composition |  | 1 |  Order —◆—Invoice [0..1]—→ Invoice<br><br>1, 1..* |
| Aggregation |  | 0 / 1 |  Car —◇—Wheel [*]—→ Wheel<br><br>0..1, 1..* |
| None |  | * |  Person —Car—→ Car<br><br>*, * |

# 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.

## 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).
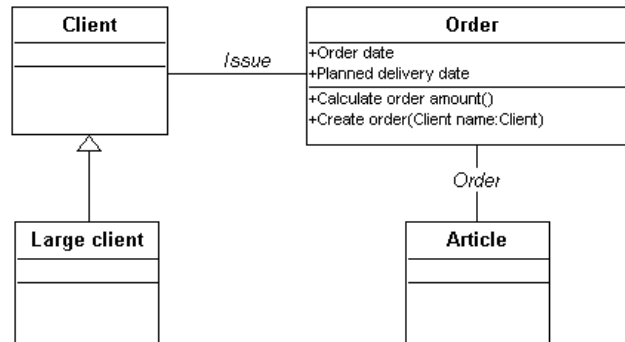


In the above example, the associations and attributes specified for "Client" are also valid for "Large client".

Other examples of generalizations:

- Prospect and client are two subclasses of "person".
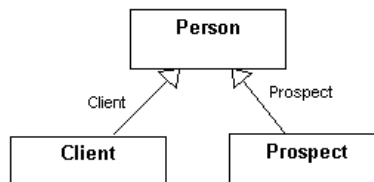


- Export order is a subclass of the "order" class.
- Individual person and corporate person are two subclasses of the "person" class.
- Polygon, ellipse, and circle are subclasses of the "shape" class.
- Oak, elm, and birch are subclasses of the "tree" class.
- Motor vehicle, all-terrain vehicle, and amphibious vehicle are subclasses of the "vehicle" class.
- Truck is a subclass of the "motor vehicle" class.

# Multiple Subclasses
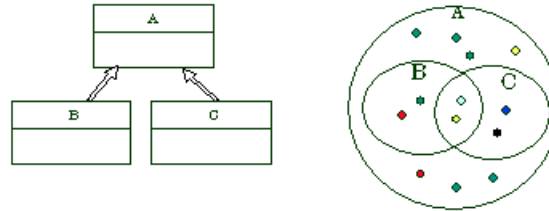


When a class has multiple subclasses, they:
- are not necessarily exclusive.
- do not necessarily partition the set.

# Advantages of Subclasses
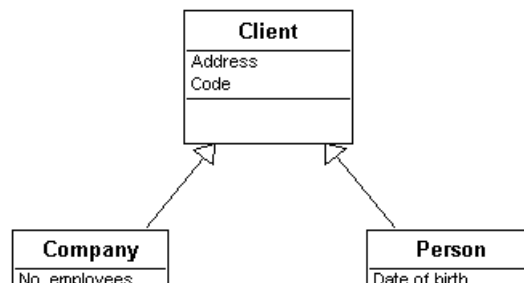


A subclass inherits all the attributes, operations, and associations of its superclass, but can have its own attributes or associations that the superclass does not have.

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*.
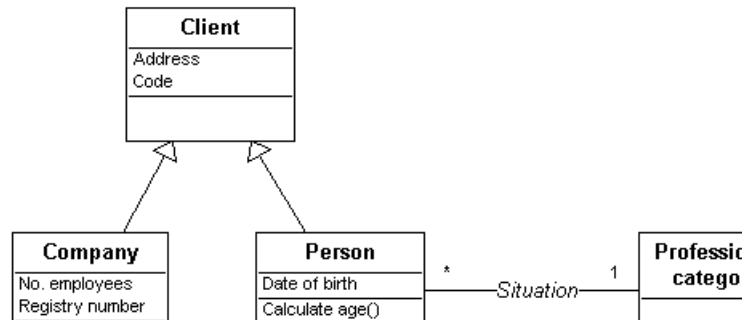


- A "person" falls into a "socio-professional group": "manager", "employee", "shop keeper", "grower", etc. This classification makes no sense for a "company". There is also a classification for companies, but it differs from that for persons.

## 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.
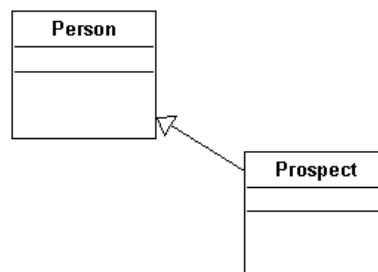
# Discriminator

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.
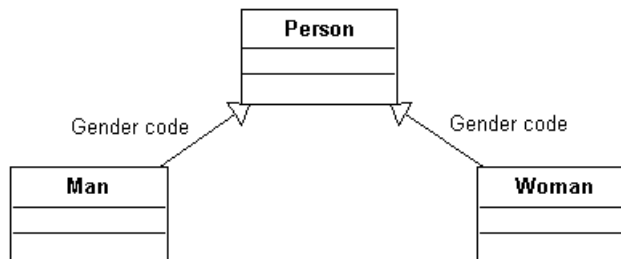


You can define discriminator(s) in the generalization properties dialog box, under the **Discriminators** page.

> ☞ *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**.
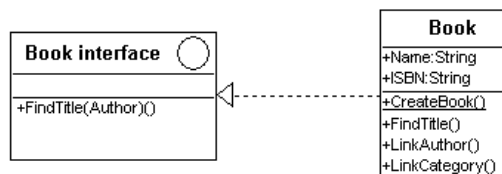
# SPECIFYING DEPENDENCIES

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 **Characteristics** property page of the class.
2.  You can enter the parameters and specify their type if necessary.

**TemplateParameters:** Object, ListTyp

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.
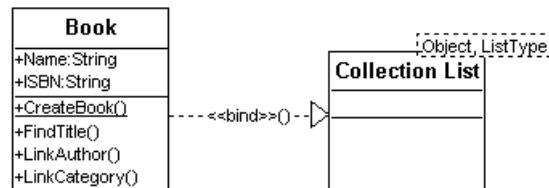
# CONSTRAINTS

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.
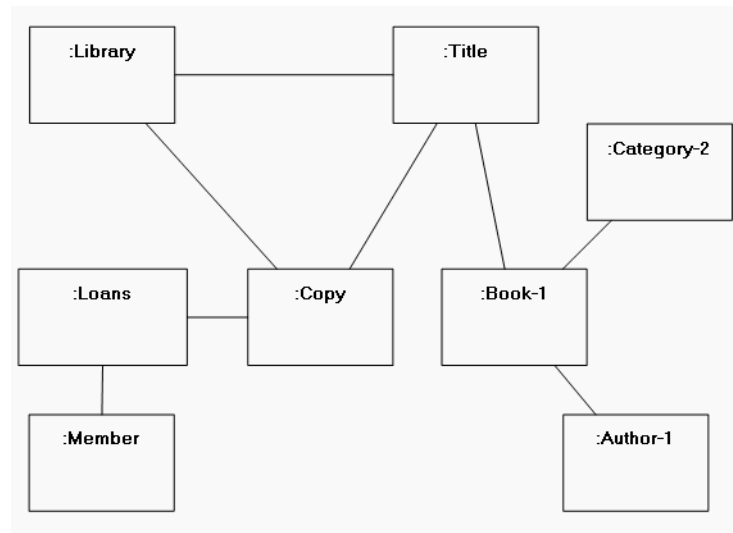


## Objects

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:

- ✓ The Package Diagram, enabling organization of elements of the model
- ✓ The Component Diagram, highlighting dependency relationships between components
- ✓ 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.
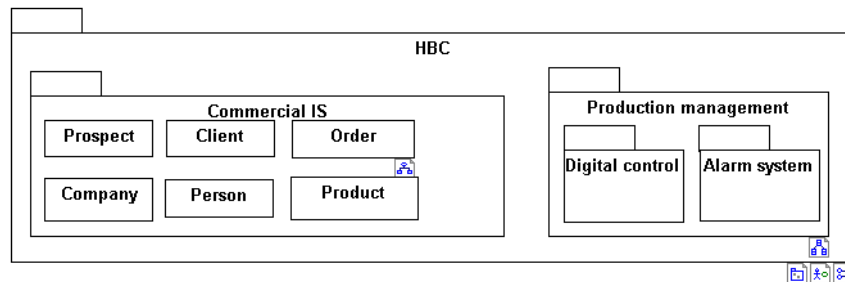


## Creating a Package Diagram

A package diagram is created from a package.

To create a packages diagram with **Hopex IT Architecture** from **Design (UML)** navigation pane:

1. Click **Packages** sub-menu.
2. Select the package stream that interests you and click **New Diagram**.
3. Select **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 Search** to open the search 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.
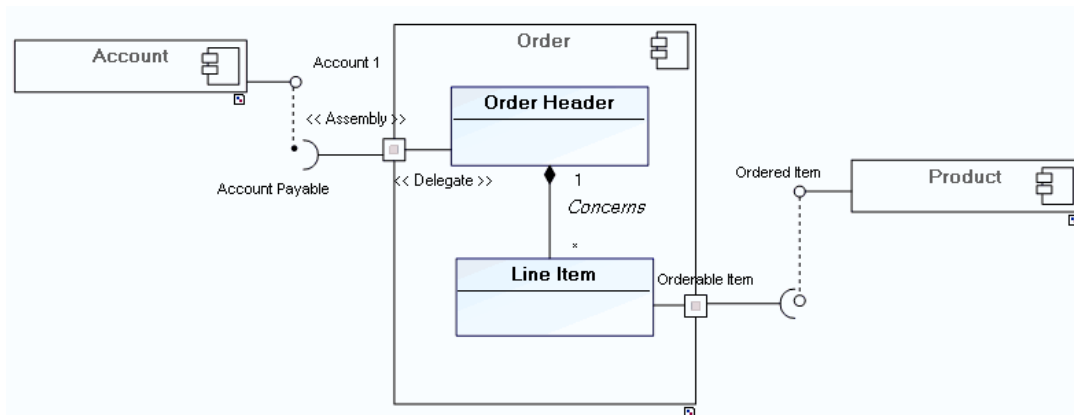


## Creating a Component Diagram

In **Hopex IT Architecture**, you can create a component diagram using a component or package.

To create a component diagram with **Hopex IT Architecture** from **Design (UML)** navigation menu:

1.  From the navigation sub-menu, click **Packages**.
2.  Select the package stream that interests you and click **New Diagram**.
3.  Select **Components 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.
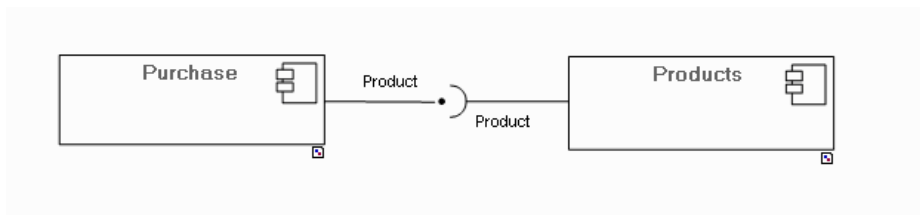


# Ports

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.
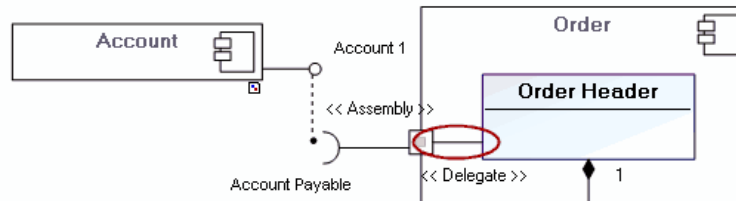


## Assembly connector

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.
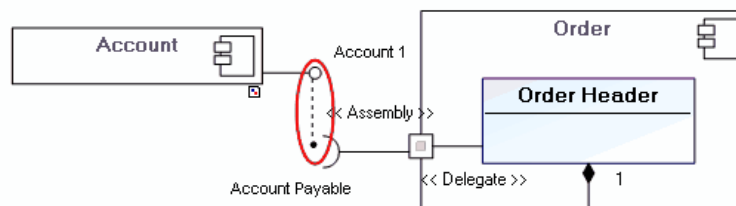
# COMPOSITE STRUCTURE DIAGRAM

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.
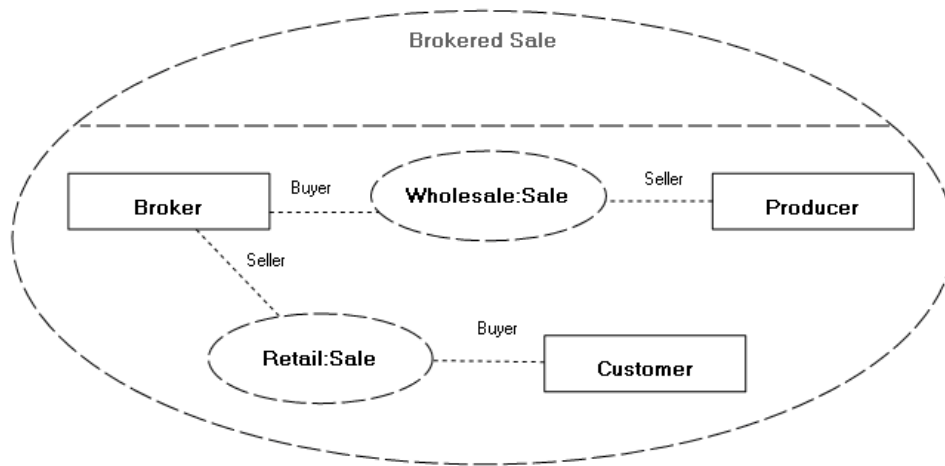


## Creating a Composite Structure Diagram

To create a composite structure diagram with **Hopex IT Architecture** from **Design (UML)** navigation menu:

1. From the navigation sub-menu, click **Components**.
2. Select component stream that interests you and click **New Diagram**.
3. Select **Composite structure diagram**.
   The diagram opens 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:

✓  Connectors
✓  Dependency links
✓  Ports.

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.
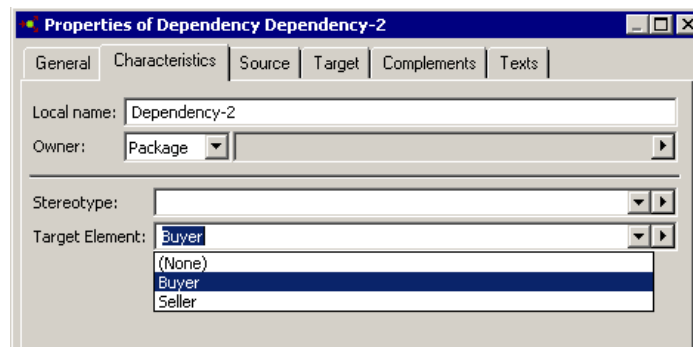


# Dependency links

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.
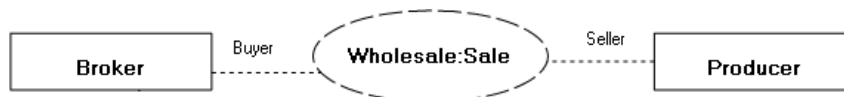
A stereotype on the dependency enables specification of dependency type:

- 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.
- Mapping UML/XML : expression that defines the relationship between elements (classes, attributes, ...) 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:

- ✓ Presentation of the State Machine Diagram
- ✓ Creating a State Machine Diagram
- ✓ States
- ✓ 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:
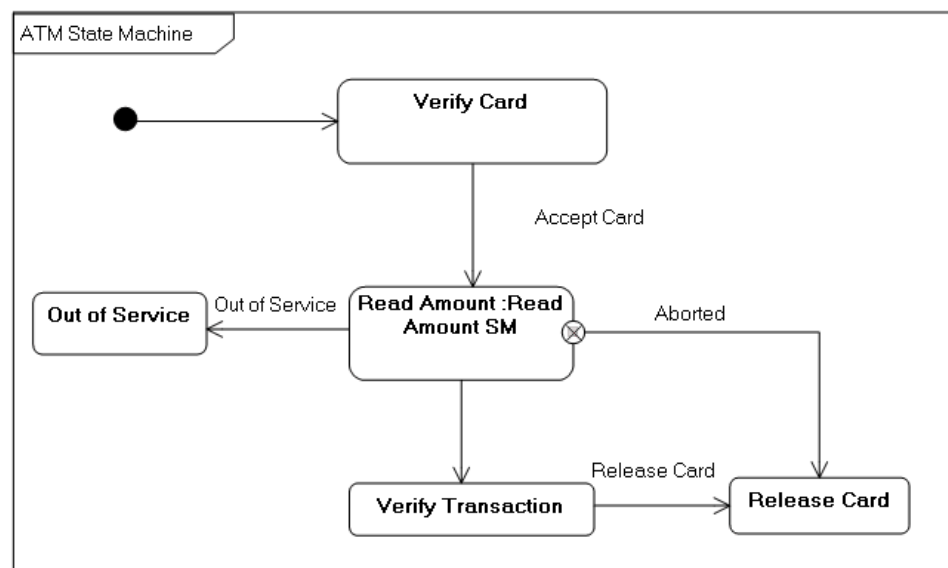


## Creating a State Machine Diagram

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 with  **Hopex IT Architecture** from **Design (UML)** navigation menu:

1. From the navigation sub-menu, click **status**.
2. Select machine stream that interests you and click **New Diagram**.
3. 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*
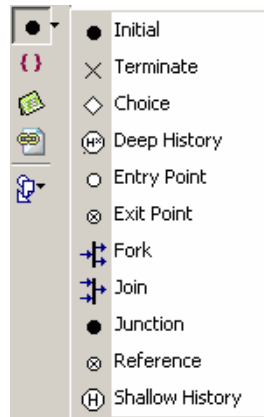


An initial pseudo-state has a single output transition to the initial state of the object at its creation.

### *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 **Characteristics** property page of the described state.
2. 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 **Characteristics** property page of the transition.

### 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 **Characteristics** property page of the transition.
2. 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.

- ✓ Activity Diagram
- ✓ Partitions
- ✓ Nodes
- ✓ 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*
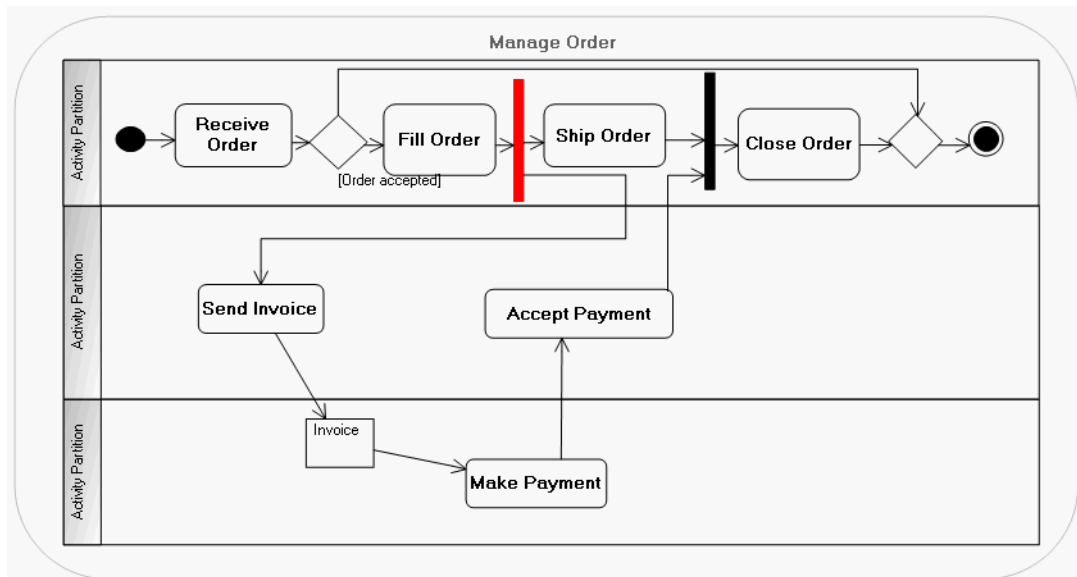


## Creating an Activity Diagram

In **Hopex IT Architecture**, 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.
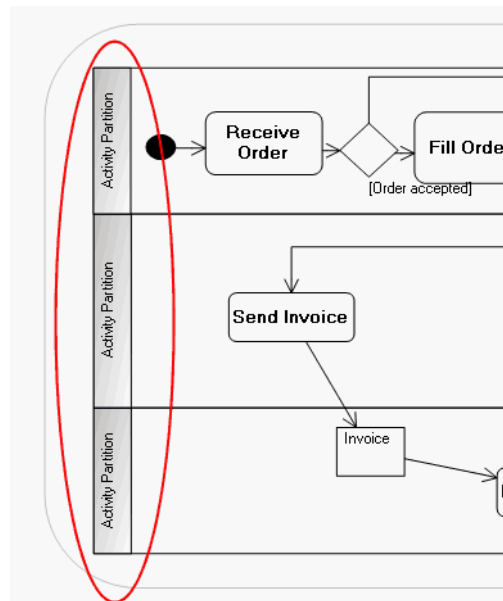


## Creating a Partition

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 **Characteristics** property page of the action, 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.
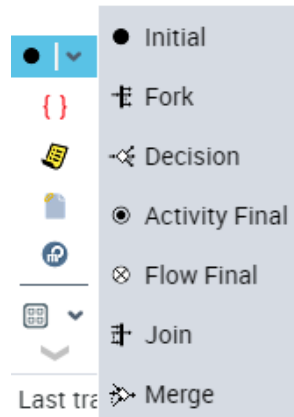


## Control node types

### 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 an activity, all flows of the activity are stopped. Conversely, a final Node 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: :

✓ Interactions
✓ Sequence Diagram
✓ Communication Diagram
✓ 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 with **Hopex IT Architecture** using **Design (UML)** navigation pane:

1. Click **Interaction** sub-menu.
2. Click the **New** button.
3. Enter the name of the interaction and an owner if necessary.
4. 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.
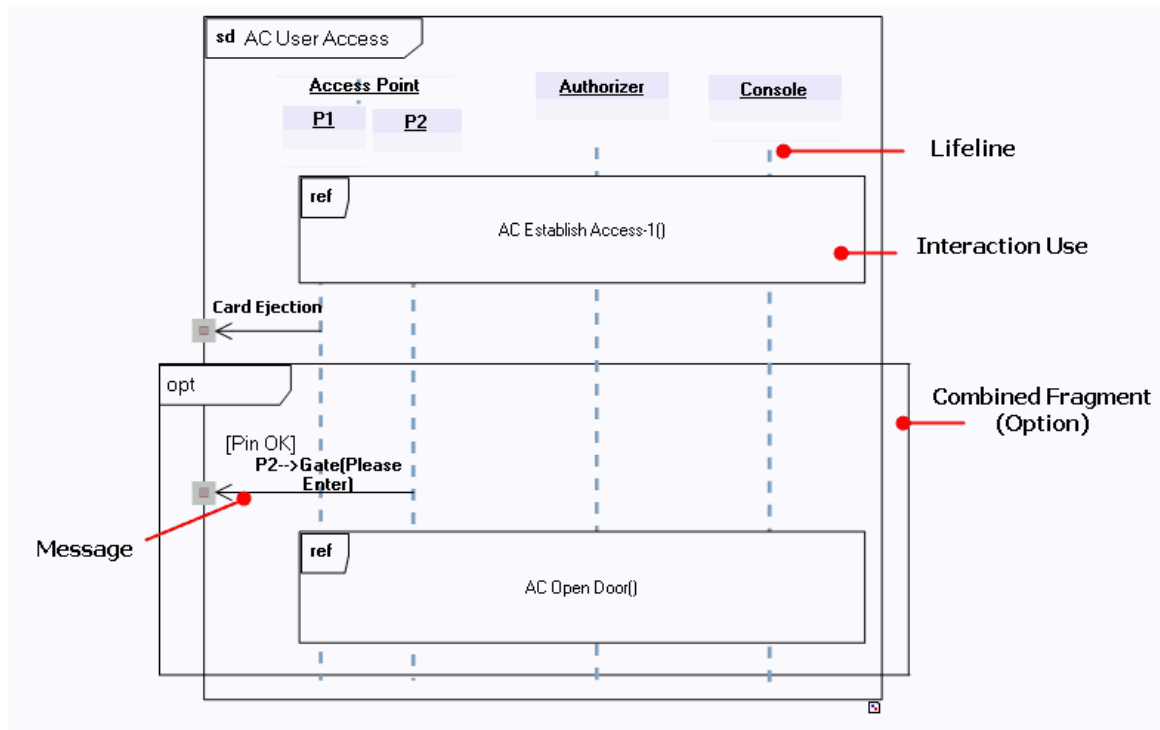
# Creating a Sequence Diagram

To create a sequence diagram in **Hopex IT Architecture** :

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 in 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.
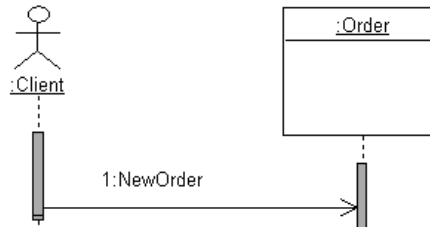


2) The message sent by the "Order" class to the "Product" class calls the "Reduce inventory" operation.
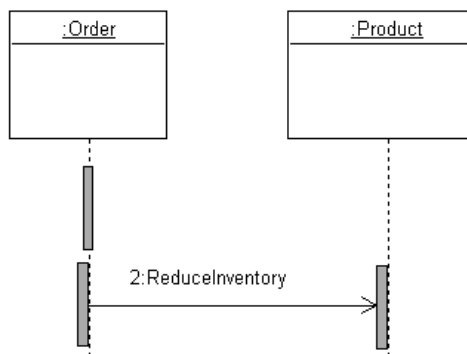


## Creating a message

To create a message in the sequence diagram:

1. Click the **Message** button in the insert toolbar, selecting the required message type.
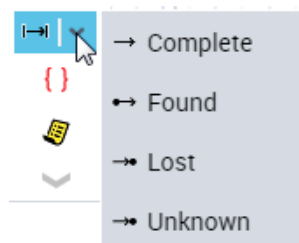


2. Click on the dotted line under the first object, and hold down the mouse button while dragging the cursor to the dotted line under the second object.
The message exchanged between the two objects is drawn.

### 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.
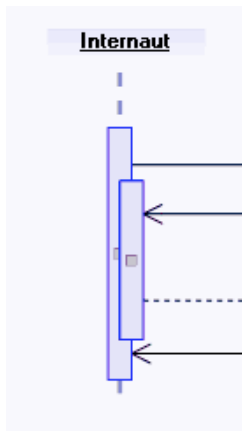


### Creating an execution specification

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.
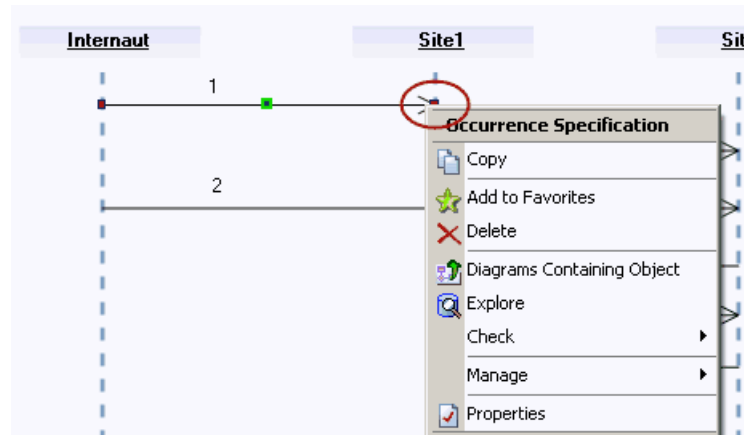


## Calculating sequence numbers

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.
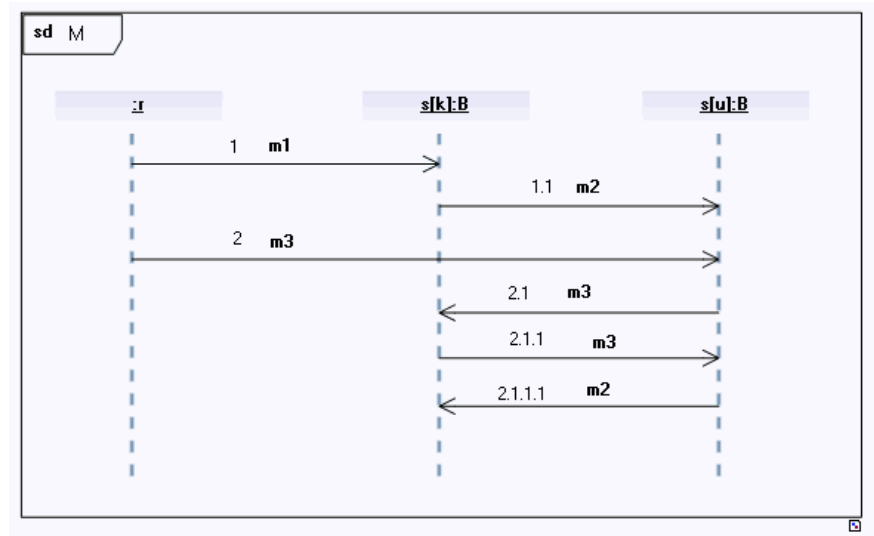
You can manually modify the sequence number of a message in the message properties dialog box:

❱ 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.
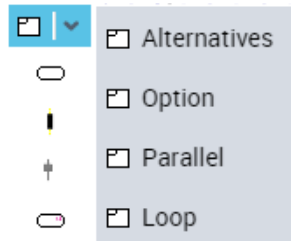


2.  Click in the diagram.
    The combined fragment creation dialog box appears.
3.  Specify its **Name** and the **Interaction Operator Type** if not already indicated.
4.  Click **Finish**.

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).
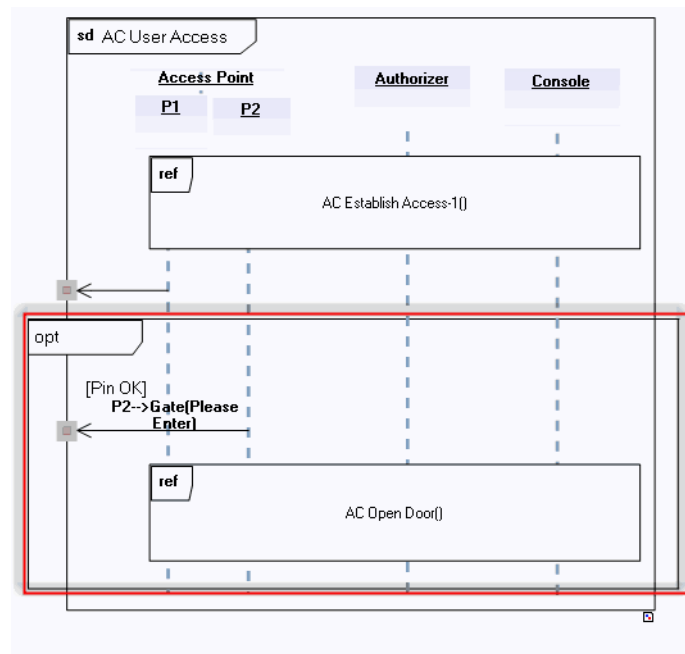


## Interaction operator type

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***
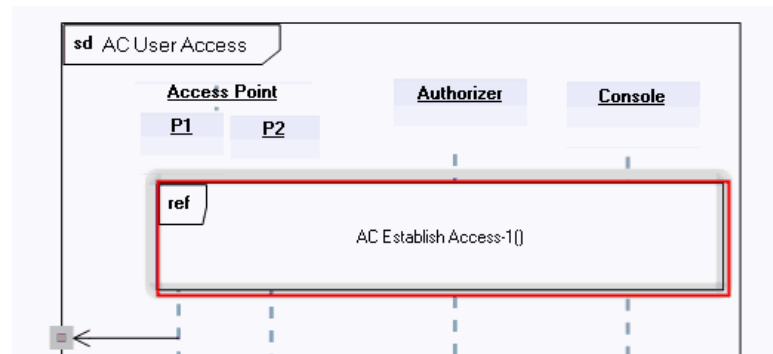


To create an interaction use:

1.  Click the **Interaction Use** button 
2.  Click in 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 **Characteristics** property page of the interaction use.
2.  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 the **Characteristics** property page of the interaction use.
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*
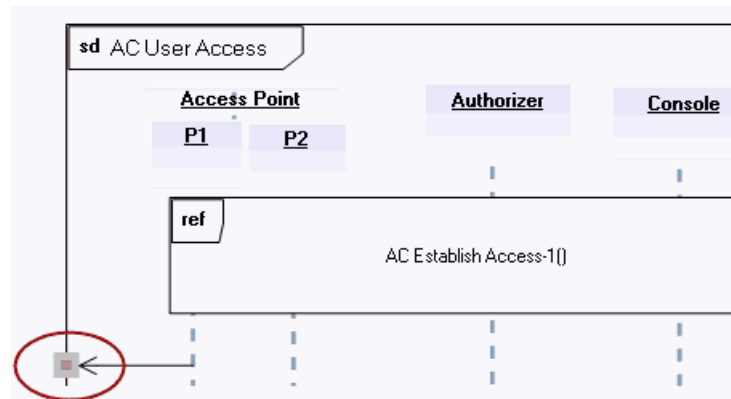


To create a gate in the sequence diagram:

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
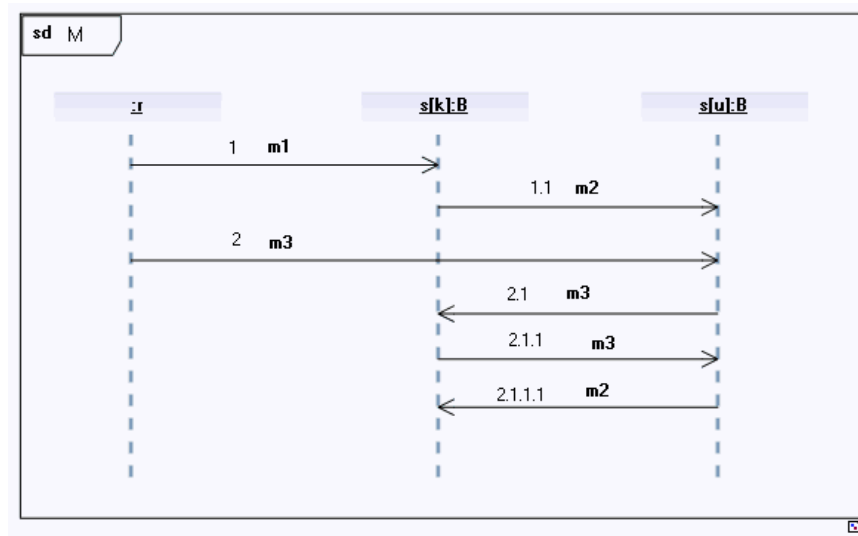
*Communication Diagram*
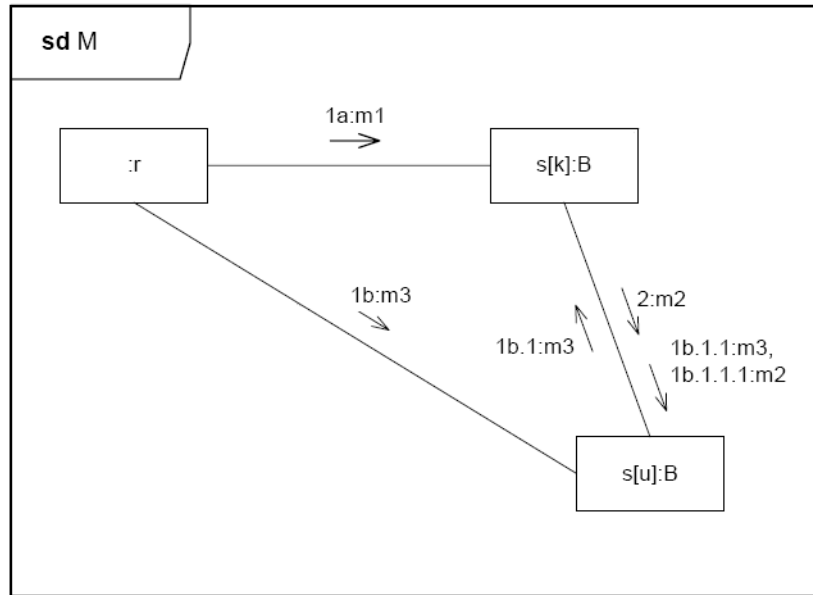


## Diagram objects

Communication diagram objects are lifelines and messages transmitted by connectors.

When you connect two lifelines with a connector [icon] , 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.

# 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.
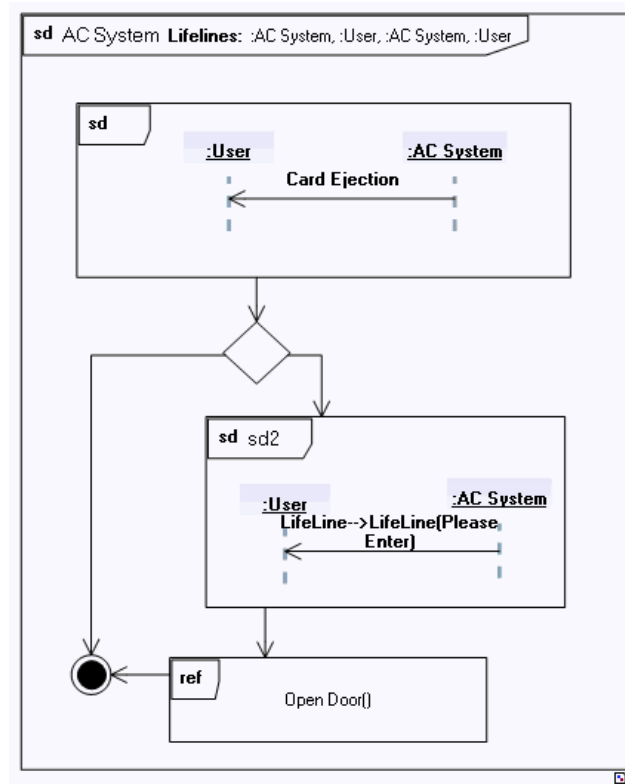


Objects represented in the interaction overview diagram are interactions and interaction uses, lifelines, messages, control nodes and control flows.

# THE DEPLOYMENT DIAGRAM

The deployment diagram complements the component diagram with hardware resources on which components run.

✓   Presentation of the Deployment Diagram.

# PRESENTATION OF 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*
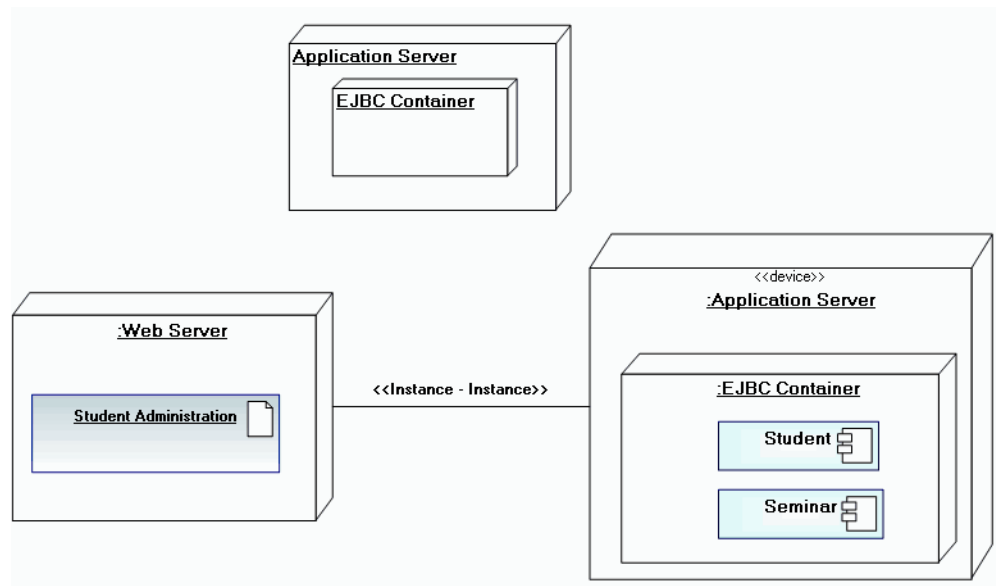


## Creating a Deployment Diagram

In **Hopex IT Architecture**, a deployment diagram is created from a package.

To create deployment diagram with **Hopex IT Architecture** from **Design (UML)** navigation menu:

1. Click **Packages** sub-menu.
2. Select the package stream that interests you and click **New Diagram**.
3. Select **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 Node, either place the component in the node, or connect the component to the node by a dependency link.

See Dependency links.

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 Node.

## Configuration

The configuration button ![button] enables creation of the link between a deployment specification and a deployment.

### *Example*
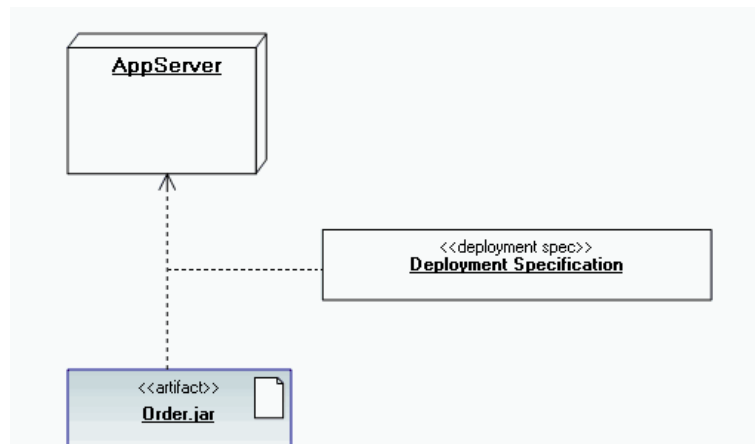
# APPENDIX: ATTRIBUTE TYPE

The following points are covered here:

- ✓ Primitive Types
- ✓ Packages and Primitive Types
- ✓ Defining New Primitive Types

# PRIMITIVE TYPES

A primitive type is used to group characteristics shared by several attributes. Primitive types are implemented as classes.

## Prerequisite: Importing the Primitive Types

To access primitive types in **Hopex IT Architecture**, the administrator must import the "ISQL ANSI" module in your environment. To import a module in **Hopex**, see **Modules > Importing a Module** documentation.

## 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:

| Alphanumeric types | | Other Information |
|---|---|---|
| M-Char | Alphanumeric string of fixed length | Length |
| M-Varchar | Alphanumeric string of variable length | |
| **Numeric types** | | |
| M-Numeric | Number | Length, decimal places |
| M-Amount | Amount expressed as currency | Length, decimal places |

| Date types | | |
|---|---|---|
| M-Date | Date | |
| M-Time | Time | |
| M-Datetime | Date and time | |
| **Binary types** | | |
| M-Timestamp | Identification automatically generated from the date and time, expressed in thousandths of seconds since 01 January 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.
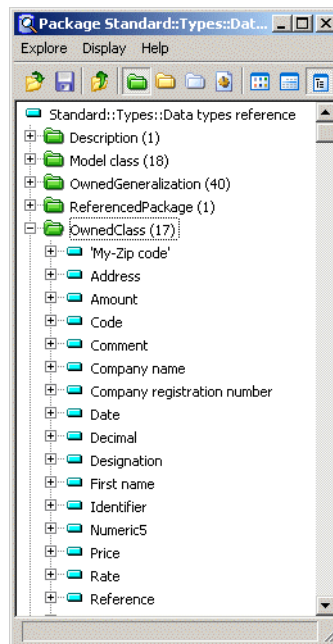


It is referenced by the packages "Library", "Order management", etc.

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.
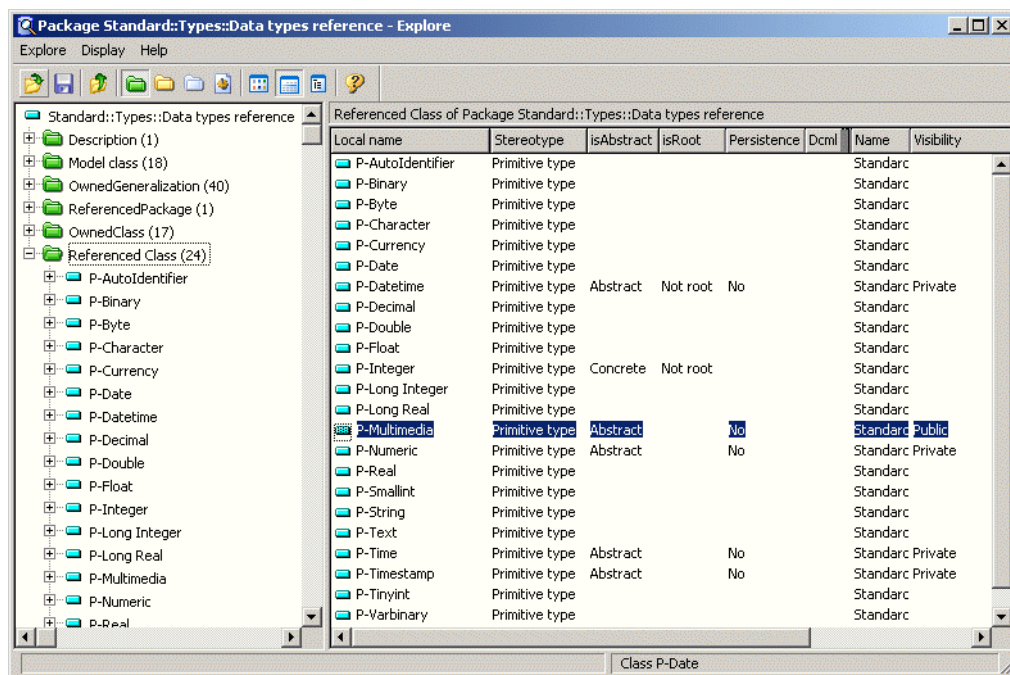
# DEFINING NEW PRIMITIVE TYPES

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:
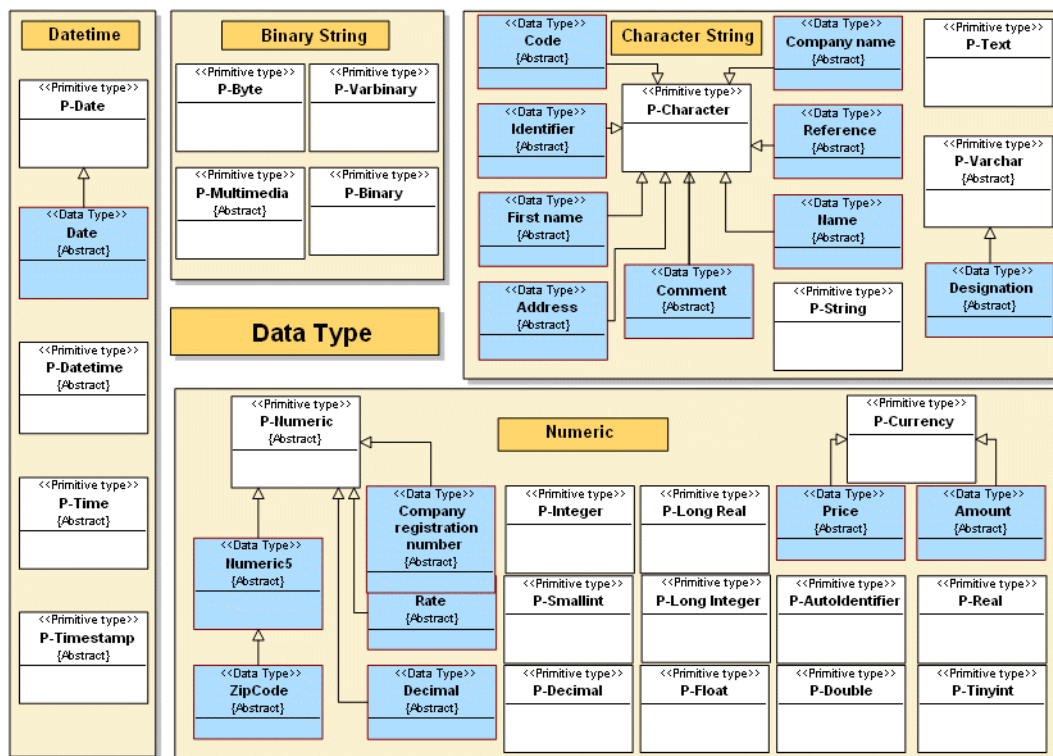


The primitive types defined as subclasses will automatically inherit the characteristics of their superclass. In particular, the datatype conversion rule for the superclass is applied to the subclass.

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".
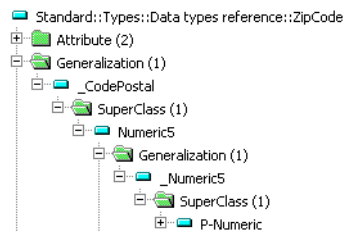


If the new primitive type is not defined directly or indirectly as a subclass of a standard primitive type, the conversion table that maps primitive types to column data types must be updated.

> ☛ *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 "Mappings between Pivot Types and Datatypes" in the* **Hopex Database** *guide).*

# Compound Primitive Type

You can define a compound primitive type by assigning to it a list of attributes.
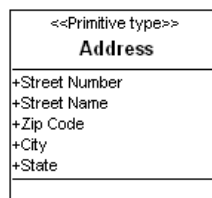


Here the Address type is composed of number, street, zip code, city, and country.

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:
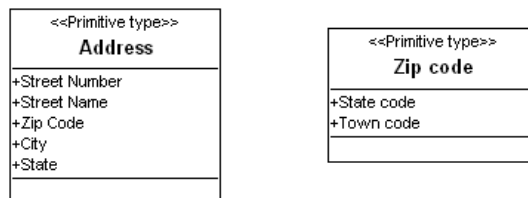
HOPEX XMI 2.1 Import for UML2

# 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.

## 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.
See object details in HOPEX/XMI Object Mapping.

💣✳ **Only objects are imported. The XMI Import tool does not take into account UML profiles, extensions and graphical diagram drawings.**

# 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)**.
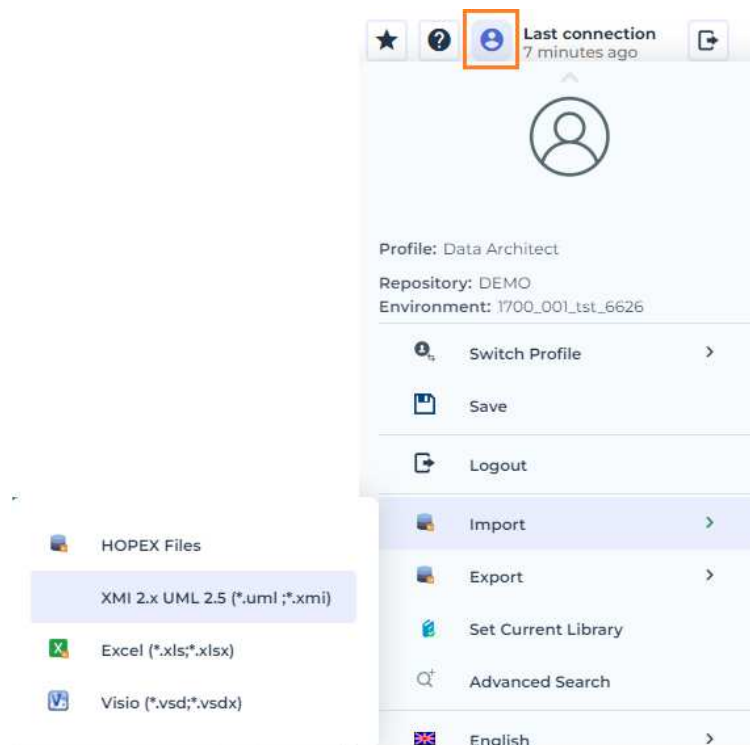


The import wizard appears.

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

Classes.

> Elements such Components, ports packages, interfaces, required interfaces, provided interfaces, Connectors or realized elements may be imported.

### 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 | Nested Class |
| | Visibility | |
| | Abstract | |
| | IsLeaf | |
| | IsActive | |
| | Client Dependency | |
| Interface | xmi_id | Class Target Dependency |
| | Name | Nested Class |
| | Visibility | Association |
| | Comment | Attribute |
| | Abstract | Operation (UML) |
| | IsLeaf | Generalization |
| | Client Dependency | RequiredInterface |
| | | SpecificationInterface |
| Enumeration | _Hexaidabs | Class Target Dependency |
| | Name | Attribute |
| | Visibility | Operation (UML) |
| | Comment | Literal Value |
| | Client Dependency | RequiredInterface |
| | | Specification Interface |
| LiteralValue | _Hexaidabs | Value Slot |
| | Name | |

| MEGA Concepts | MetaAttribute | MetaAssociation(End) |
|---|---|---|
| Expression | xmi_id<br><br>Name<br><br>Visibility<br><br>Comment<br><br>Client Dependency | Class Target Dependency<br><br>Specification Interface |
| Primitive Type | Name<br><br>xmi_id<br><br>Visibility<br><br>Abstract<br><br>IsLeaf<br><br>IsActive | Class Target Dependency<br><br>Nested Class |
| Association | xmi_id<br><br>Name<br><br>Visibility<br><br>Comment<br><br>IsAssociationDerived<br><br>IsNavigable | Connection<br><br>Dependency (Target Association)<br><br>Class via AssociationEnd |
| Association End | xmi_id<br><br>Name<br><br>Aggregation: Composite/Shared | Association<br><br><br>Dependency |
| Association Class | xmi_id | Class Target Dependency |

| MEGA Concepts | MetaAttribute | MetaAssociation(End) |
|---|---|---|
| | Name | Nested Class |
| | Visibility | AssociationEnd |
| | Comment | Association |
| | IsLeaf | Association Class |
| | Abstract | Class via AssociationEnd |
| | IsActive | Attribute |
| | IsAssociationDerived | Operation (UML) |
| | IsNavigable | |
| Attribute | xmi_id | Dependency (Target Attribute) |
| | Name | AttributType |
| | Visibility | |
| | Comment | |
| | IsLeaf | |
| | IsOrdred | |
| | Uniqueness | |
| | ReadOnly | |
| | IsDerived | |
| | InitialValue | |
| | Multiplicity: UpperValue, LowerValue | |
| Operation (UML) | xmi_id | Precondition |
| | Name | Postcondition |
| | Visibility | Parameter |
| | Comment | ReturnType |

| MEGA Concepts | MetaAttribute | MetaAssociation(End) |
|---|---|---|
| | Abstract<br><br>IsQuery | Target Dependency |
| Dependency | _Hexaidabs<br><br>Name<br><br>Visibility<br><br>Comment | Class Source<br><br>Class Target<br><br>Stereotype |
| Generalization | xmi_id<br><br>Name<br><br>Comment | Super Class<br><br>UML constraint |
| GeneralizationSet | xmi_id<br><br>Name<br><br>Comment<br><br>IsComplete<br><br>IsDisjoint | Generalization |
| Constraint | xmi_id<br><br>Name<br><br>Comment<br><br>MaxInt<br><br>MinInt<br><br>Specification | ConstrainedClass<br><br>ConstrainedGeneralization<br><br>ConstrainedElement<br><br>Actor (UML)<br><br>Package<br><br>UseCase<br><br>UseCaseParticipation |

| MEGA Concepts | MetaAttribute | MetaAssociation(End) |
|---|---|---|
| Parameter | _Hexaidabs<br><br>Name<br><br>Comment | Parameter Type |
| Behavior (UML) | _Hexaidabs<br><br>Name | |

## Use Case Diagram

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| UseCase | xmi_id<br><br>Name<br><br>Visibility<br><br>Comment | UsesUseCase<br><br>OwnedExtension<br><br>ExtensionPoint<br><br>Behavior: State Machine<br><br>Behavior: Protocol State Machine<br><br>Behavior: Interaction UML<br><br>Behavior: Activity UML<br><br>Behavior: Collaboration UML<br><br>Constraint<br><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 |
| | Multiplicity | |
| Extension | 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 | |

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| | IsOrdered<br><br>Multiplicity<br><br>Aggregation: Composite/Shared<br><br>Comment | |
| Connector | xmi_id<br><br>Name<br><br>Connector Kind<br><br>IsLeaf | OwnedConnectorEnd |
| ConnectorEnd | xmi_id<br><br>Name<br><br>Multiplicity | Connector |

## State Machine

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| State Machine | xmi_id<br><br>Name<br><br>Comment<br><br>Reentrant | DetailedState<br><br>Region |
| Region | xmi_id<br><br>Name<br><br>Comment | State<br><br>PseudoState<br><br>Transition |
| State (UML) | xmi_id | Detailing Behavior |

| | Name | Outgoing |
|---|---|---|
| | Comment | Incoming |
| | | OwnedRegion |
| | | DoActivity |
| | | ExitActivity |
| | | EntryActivivity |
| | | ConnectionPoint (Entry Point/ Exit Point) |
| Pseudo State | xmi_id | Outgoing Transition |
| | Name | Incoming Transition |
| | Comment | |
| | PseudoStateKind | |
| Transition (UML) | xmi_id | Source |
| | Name | Target |
| | Comment | Source Pseudo State |
| | | Target Pseudo State |
| | | Trigger |
| | | Effect (Behavior) |
| | | Constraint |
| Event (UML) | xmi_id | EventKind |
| | Name | |
| | Comment | |
| Protocol State Machine | xmi_id | Region |
| | Name | |
| | Visibility | |

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| | Comment | |
| Trigger (UML) | xmi_id<br><br>Name<br><br>Comment | Event (UML) |

## Sequence, Communication and Interaction Overview Diagram

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| Interaction UML | xmi_id<br><br>Name<br><br>Comment | Gate<br><br>Fragment (Combined Fragment, State Invariant, ...)<br><br>LifeLine<br><br>Message<br><br>Action<br><br>Parameter<br><br>Operation (UML)<br><br>OwnedInteraction (UML)<br><br>OwnedInteractionOperand |
| Interaction Operand | xmi_id<br><br>Name<br><br>Comment | Fragment (Combined Fragment, State Invariant, ...)<br><br>OwnedInteraction (UML)<br><br>OwnedInteractionOperand |
| OccurrenceSpecification | xmi_id<br><br>Name<br><br>Comment | Event (UML)<br><br>Message |
| ExecutionSpecification | xmi_id<br><br>Name | start<br><br>finish |

| | | |
|---|---|---|
| | Comment | |
| LifeLine | xmi_id<br><br>Name<br><br>Comment | ElementRepresentedByALifeline<br><br>ElementCoveringLifeline<br><br>OwnedSelector |
| Combined Fragment | xmi_id<br><br>Name<br><br>InteractionOperatorKind<br><br>   Comment | InteractionOperand<br><br>CoveredLifeLine |
| Interaction Use | xmi_id<br><br>Name<br><br>Comment | RefersTo<br><br>CoveredLifeLine |
| State Invariant | xmi_id<br><br>Name<br><br>Comment | InvariantConstraint<br><br>CoveredLifeLine |
| Gate | xmi_id<br><br>Name<br><br>Comment | |
| Message UML | xmi_id<br><br>Name<br><br>MessageKind<br><br>MessageSort<br><br>Comment | Receiver  Sender  Connector |

## Activity and Interaction Overview Diagram

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| Action | xmi_id | OutgoingControlFlow |
| | Name | OutgoingObjectFlow |
| | IsLeaf | IncomingControlFlow |
| | ActionKind | IncomingObjectFlow |
| | Comment | OwnerGroup |
| | | RequestOperation |
| | | CalledBehavior |
| | | StructuralFeatureElementManagedByAnAction |
| | | RequestSignal |
| | | Association |
| | | ClassManagedByAnAction |
| | | Variable |
| | | InputPin |
| | | OutputPin |
| | | ProtectingExceptionHandler |
| | | Trigger |
| | | LocalPostCondition |
| | | LocalPreCondition |
| | | Constraint |
| Control Node | xmi_id | OutgoingControlFlow |
| | Name | OutgoingObjectFlow |
| | IsLeaf | IncomingControlFlow |
| | Comment | IncomingObjectFlow |
| | ControlNodeType | OwnerGroup |

| | | |
|---|---|---|
| Input Pin | xmi_id<br><br>Name<br><br>IsLeaf<br><br>InputPinKind<br><br>ControlType<br><br>OrderingKind<br><br>Comment | OutgoingControlFlow<br><br>OutgoingObjectFlow<br><br>IncomingControlFlow<br><br>IncomingObjectFlow<br><br>Constraint |
| Output Pin | xmi_id<br><br>Name<br><br>IsLeaf<br><br>OutputPinKind<br><br>ControlType<br><br>OrderingKind<br><br>Comment | OutgoingControlFlow<br><br>OutgoingObjectFlow<br><br>IncomingControlFlow<br><br>IncomingObjectFlow<br><br>Constraint |
| Exchange Pin | xmi_id<br><br>Name<br><br>IsLeaf<br><br>ControlType<br><br>OrderingKind<br><br>Comment | OutgoingObjectFlow<br><br>IncomingObjectFlow<br><br>OwnerGroup |
| Central Buffer Node | xmi_id<br><br>Name<br><br>IsLeaf | OutgoingObjectFlow<br><br>IncomingObjectFlow<br><br>OwnerGroup |

| | | |
|---|---|---|
| | ControlType | |
| | OrderingKind | |
| | Comment | |
| Data Store Node | xmi_id | OutgoingObjectFlow |
| | Name | IncomingObjectFlow |
| | IsLeaf | OwnerGroup |
| | ControlType | Constraint |
| | OrderingKind | |
| | Comment | |
| Activity Partition | xmi_id | ContainedElement |
| | Name | Constraint |
| | IsDimension | |
| | IsExternal | |
| | Comment | |
| Object Flow | xmi_id | Guard |
| | Name | Weight |
| | IsLeaf | SourceElement |
| | Comment | TargetElement |
| | | Constraint |
| Control Flow | xmi_id | Guard |
| | Name | Weight |
| | IsLeaf | SourceElement |
| | Comment | TargetElement |

| | | Constraint |
|---|---|---|
| Exception Handler | xmi_id<br><br>Name<br><br>Comment | ProtectedNode<br><br>ExceptionInput<br><br>Constraint |
| Activity UML | xmi_id<br><br>Name<br><br>Reentrant<br><br>SingleExecution<br><br>IsLeaf<br><br>Comment | ElementOwnedByAnActivityUML<br><br>Constraint |
| Activity Parameter Node | xmi_id<br><br>Name<br><br>IsLeaf<br><br>ControlType<br><br>OrderingKind<br><br>Comment | OutgoingObjectFlow<br><br>IncomingObjectFlow<br><br>Constraint |
| Structured Activity Node | xmi_id<br><br>Name<br><br>IsLeaf<br><br>MustIsolate<br><br>Comment | OutgoingControlFlow<br><br>OutgoingObjectFlow<br><br>IncomingControlFlow<br><br>IncomingObjectFlow<br><br>OwnerGroup<br><br>ContainedElement<br><br>InputPin |

| | | OutputPin |
|---|---|---|
| | | Constraint |
| Expansion Node | xmi_id | OutgoingControlFlow |
| | Name | OutgoingObjectFlow |
| | IsLeaf | IncomingControlFlow |
| | ControlType | IncomingObjectFlow |
| | OrderingKind | Region |
| | Comment | Constraint |
| Expansion Region | xmi_id | OutgoingControlFlow |
| | Name | OutgoingObjectFlow |
| | IsLeaf | IncomingControlFlow |
| | MustIsolate | IncomingObjectFlow |
| | ExpansionKind | OwnerGroup |
| | Comment | ContainedElement |
| | | ExpansionNode |
| | | InputElement |
| | | OutputElement |
| Interruptible Activity Region | xmi_id | OwnerGroup |
| | Name | ContainedElement |
| | Comment | |
| | | Constraint |
| Loop Node | xmi_id | OutgoingControlFlow |
| | Name | OutgoingObjectFlow |
| | IsLeaf | IncomingControlFlow |

| | | |
|---|---|---|
| | MustIsolate | IncomingObjectFlow |
| | TestedFirst | OwnerGroup |
| | Comment | ContainedElement |
| | | InputPin |
| | | OutputPin |
| | | Constraint |
| | | Test |
| Conditional Node | xmi_id | OwnerGroup |
| | Name | ContainedElement |
| | IsLeaf | InputPin |
| | MustIsolate | OutputPin |
| | Assured | Constraint |
| | Determinate | OutgoingControlFlow |
| | Comment | OutgoingObjectFlow |
| | | IncomingControlFlow |
| | | IncomingObjectFlow |
| Sequence Node | xmi_id | OutgoingControlFlow |
| | Name | OutgoingObjectFlow |
| | IsLeaf | IncomingControlFlow |
| | MustIsolate | IncomingObjectFlow |
| | Comment | OwnerGroup |
| | | ContainedElement |
| | | InputPin |
| | | OutputPin |
| | | Constraint |

## Package Diagram

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| Package | Name | Client Dependency |
| | xmi_id | Package Target Dependency |
| | Visibility | Owned Class |
| | Comment | Owned Package |
| | | Owned Association |
| | | Association Class |
| | | Owned Dependency |
| | | Owned Element (UML): Generalization |
| | | Owned Use Case |
| | | Owned Actor (UML) |
| | | Constraint |
| | | Behavior: State Machine |
| | | Behavior: Protocol State Machine |
| | | Behavior: Activity Uml |
| | | Behavior: Interaction Uml |
| | | Behavior: Collaboration Uml |
| | | Owned Component |
| | | Owned Event |

## Component Diagram

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| Component | xmi_id | required Interface |
| | Name | provided |
| | Comment | Port |

| | Isleaf | OwnedPart |
| --- | --- | --- |
| | Visibility | |
| | Client Dependency | |
| Port | xmi_id | |
| | Name | |
| | Comment | |
| | Client Dependency | |
| Interface | xmi_id | Class Target Dependency |
| | Name | Nested Class |
| | Visibility | Association |
| | Comment | Association Class |
| | Abstract | Attribute |
| | IsLeaf | Operation (UML) |
| | Client Dependency | Generalization |
| | | RequiredInterface |
| | | SpecificationInterface |

## Deployment Diagram

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
| --- | --- | --- |
| Artifact UML | xmi_id | Target Dependency |
| | Name | OwnedAttribute |
| | Comment | OwnedOperation |
| | Client Dependency | |
| | | NestedArtifact |
| Node UML | xmi_id | Deployment |

|  |  |  |
|---|---|---|
|  | Name<br><br>Comment<br><br>Client Dependency |  |
| Device | xmi_id<br><br>Name<br><br>Comment<br><br>Client Dependency | Deployment |
| Execution Environment | xmi_id<br><br>Name<br><br>Comment<br><br>Client Dependency | Deployment |
| Deployment Specification | xmi_id<br><br>Name<br><br>Comment<br><br>Client Dependency | Target Dependency |
| Instance ( Node UML/Device/Execution Specification/Component) | xmi_id<br><br>Name<br><br>Comment<br><br>Client Dependency | Instantiated Element<br><br>Target Dependency |
| Communication Path | xmi_id<br><br>Name<br><br>Comment | Communication Path End |

| | | |
|---|---|---|
| Deployment | xmi_id<br><br>Name<br><br>Comment | Deployed Element<br><br>Deployment Configuration |
| Manifestation | xmi_id<br><br>Name<br><br>Comment | Multiplicity<br><br>Deployed Element<br><br>Deployment Configuration |

## Object Diagram

| MEGA Concepts | MetaAttribut | MetaAssociation(End) |
|---|---|---|
| Instance | xmi_id<br><br>Name<br><br>Comment | |
| Link | xmi_id<br><br>Name<br><br>Comment | LinkEnd |
| LinkEnd | xmi_id<br><br>Name<br><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**.
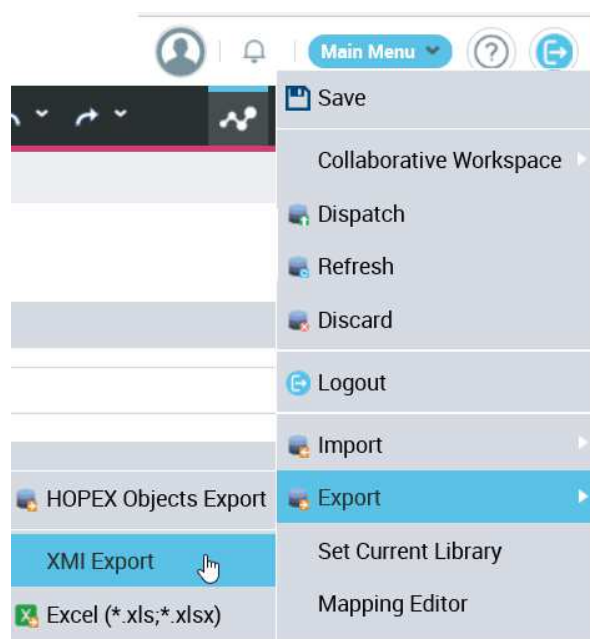


    The export dialog box appears.

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**

Classes.

Elements such Components, ports packages, interfaces, required interfaces, provided interfaces, Connectors or realized elements may be imported.

**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 |
| --- |
| ⛏ Behavior: Collaboration Uml |
| ⛏ Owned Component |
| ⛏ Owned Event |

**Class**

- Name
- _Hexaidabs
- Visibility
- Comment
- Abstract
- IsLeaf
- IsActive
- Client Dependency

⛏ Class Target Dependency

⛏ Realization Class

⛏ Nested Class

⛏ Association

⛏ Connector

⛏ Association Class

⛏ Attribute

⛏ Operation (UML)

⛏ Generalization

⛏ Required Interface

| HOPEX Concepts |
| --- |
| ⊐ Provided Interface |
| ⊐ Constraint |
| ⊐ Method |
| ⊐ Port |
| ⊐ AssociationEnd |
| ⊐ Owned Part |
| ⊐ Behavior: State Machine |
| ⊐ Behavior: Activity Uml |
| ⊐ Behavior: Interaction Uml |
| ⊐ Behavior: Collaboration Uml |
| ▣ **Data Type** |
| ▭ Name |
| ▭ _Hexaidabs |
| ▭ Visibility |
| ▭ Abstract |
| ▭ IsLeaf |
| ▭ IsActive |
| ▭ Client Dependency |
| ⊐ Class Target Dependency |
| ⊐ Nested Class |
| ▣ **Interface** |
| ▭ _Hexaidabs |

## HOPEX Concepts

- Name
- Visibility
- Comment
- Abstract
- IsLeaf
- Client Dependency

Class Target Dependency

Nested Class

Association

Association Class

Attribute

Operation (UML)

Generalization

RequiredInterface

SpecificationInterface

**Enumeration**

- _Hexaidabs
- Name
- Visibility
- Comment
- Client Dependency

Class Target Dependency

| HOPEX Concepts |
|---|
| ⌷ Attribute |
| ⌷ Operation (UML) |
| ⌷ Literal Value |
| ⌷ RequiredInterface |
| ⌷ Specification Interface |
| ▤ **Expression** |
|     ▬ _Hexaidabs |
|     ▬ Name |
|     ▬ Visibility |
|     ▬ Comment |
|     ▬ Client Dependency |
| ⌷ Class Target Dependency |
| ⌷ Specification Interface |
| ▤ **Primitive Type** |
|     ▬ Name |
|     ▬ _Hexaidabs |
|     ▬ Visibility |
|     ▬ Abstract |
|     ▬ IsLeaf |
|     ▬ IsActive |
| ⌷ Class Target Dependency |
| ⌷ Nested Class |

| HOPEX Concepts |
| --- |
| **Association** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Visibility |
| ▭ Comment |
| ▭ IsAssociationDerived |
| ▭ IsNavigable |
| ⛁ Connection |
| ⛁ Dependency (Target Association) |
| ⛁ Class via AssociationEnd |
| **Association Class** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Visibility |
| ▭ Comment |
| ▭ IsLeaf |
| ▭ Abstract |
| ▭ IsActive |
| ▭ IsAssociationDerived |
| ▭ IsNavigable |
| ⛁ Class Target Dependency |
| ⛁ Nested Class |

| HOPEX Concepts |
|---|
| ⛏ AssociationEnd |
| ⛏ Association |
| ⛏ Association Class |
| ⛏ Class via AssociationEnd |
| ⛏ Attribute |
| ⛏ Operation (UML) |
| ▤ **Attribute** |
|     ▭ _Hexaidabs |
|     ▭ Name |
|     ▭ Visibility |
|     ▭ Comment |
|     ▭ IsOrdred |
|     ▭ Uniqueness |
|     ▭ ReadOnly |
|     ▭ IsDerived |
|     ▭ InitialValue |
|     ▭ Multiplicity : UpperValue, LowerValue |
|   ⛏ Dependency (Target Attribute) |
|   ⛏ AttributType |
|   ⛏ OverloadedAttribute |
| ▤ **Operation (UML)** |
|     ▭ _Hexaidabs |

| HOPEX Concepts |
| --- |
| ▭ Name |
| ▭ Visibility |
| ▭ Comment |
| ▭ Abstract |
| ▭ IsQuery |
| ⛁ Precondition |
| ⛁ Postcondition |
| ⛁ Method |
| ⛁ Parameter |
| ⛁ ReturnType |
| ⛁ Target Dependency |
| ▤ **Dependency** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Visibility |
| ▭ Comment |
| ⛁ Class Source |
| ⛁ Class Target |
| ▤ **Generalization** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |

| HOPEX Concepts |
|---|
| ⛏ Super Class |
| ⛏ UML constraint |
| ⊟ **GeneralizationSet** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ▭ IsComplete |
| ▭ IsDisjoint |
| ⛏ Target Dependency |
| ⛏ Generalization |
| ⊟ **Constraint** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ▭ MaxInt |
| ▭ MinInt |
| ⛏ ConstrainedClass |
| ⛏ ConstrainedGeneralization |
| ⛏ ConstrainedElement |
| ⛏ Actor (UML) |
| ⛏ Package |
| ⛏ UseCase |

| HOPEX Concepts |
| --- |
| ⊐ UseCaseParticipation |
| ▤ **Parameter** |
| ⊂⊃ _Hexaidabs |
| ⊂⊃ Name |
| ⊂⊃ Comment |
| ⊐ Parameter Type |
| ▤ **Behavior (UML)** |
| ⊂⊃ _Hexaidabs |
| ⊂⊃ Name |
| ⊐ Specification |
| ▤ **UseCase** |
| ⊂⊃ _Hexaidabs |
| ⊂⊃ Name |
| ⊂⊃ Visibility |
| ⊂⊃ Comment |
| ⊐ UsesUseCase |
| ⊐ OwnedExtension |
| ⊐ ExtensionPoint |
| ⊐ Behavior: State Machine |
| ⊐ Behavior: Protocol State Machine |
| ⊐ Behavior: Interaction UML |
| ⊐ Behavior: Activity UML |

| HOPEX Concepts |
|---|
| ⌻ Constraint |
| ⌻ Generalization |
| ▤ **Actor (UML)** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Visibility |
| ▭ Comment |
| ⌻ Participation |
| ⌻ OwnedExtension |
| ⌻ Constraint |
| ⌻ Generalization |
| ▤ **Participation** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ▭ Multiplicity |
| ⌻ UseCase |
| ⌻ Actor (UML) |
| ⌻ Constraint |
| ▤ **Extension** |
| ▭ _Hexaidabs |
| ▭ Name |

| HOPEX Concepts |
|---|
| ▭ Comment |
| ⌑ Extended Use Case |
| ⌑ Extension Location |
| ▦ **State Machine** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ▭ Reentrant |
| ⌑ DetailedState |
| ⌑ Region |
| ▦ **Region** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⌑ State |
| ⌑ PseudoState |
| ⌑ Transition |
| ▦ **State (UML)** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⌑ Detailing Behavior |

| HOPEX Concepts |
|---|
| ⌸ Outgoing |
| ⌸ Incoming |
| ⌸ OwnedRegion |
| ⌸ OwnedRegion |
| ⌸ DoActivity |
| ⌸ ExitActivity |
| ⌸ EntryActivivity |
| ▣ **Pseudo State** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ▭ PseudoStateKind |
| ⌸ Outgoing Transition |
| ⌸ Incoming Transition |
| ▣ **Transition (UML)** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⌸ Source |
| ⌸ Target |
| ⌸ Source Pseudo State |
| ⌸ Target Pseudo State |

| HOPEX Concepts |
|---|
| ⊨ Trigger<br><br>⊨ Effect (Behavior)<br><br>⊨ Constraint |
| ▤ **Event (UML)**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ Comment |
| ▤ **Protocol State Machine**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ Reentrant<br><br>▭ Comment<br><br>⊨ Region |
| ▤ **Action**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ IsLeaf<br><br>▭ ActionKind<br><br>▭ Comment<br><br>⊨ OutgoingControlFlow<br><br>⊨ OutgoingObjectFlow |

| HOPEX Concepts |
| --- |
| ⛏ IncomingControlFlow |
| ⛏ IncomingObjectFlow |
| ⛏ OwnerGroup |
| ⛏ RequestOperation |
| ⛏ CalledBehavior |
| ⛏ StructuralFeatureElementManagedByAnAction |
| ⛏ RequestSignal |
| ⛏ Association |
| ⛏ ClassManagedByAnAction |
| ⛏ Variable |
| ⛏ InputPin |
| ⛏ OutputPin |
| ⛏ ProtectingExceptionHandler |
| ⛏ Trigger |
| ⛏ LocalPostCondition |
| ⛏ LocalPreCondition |
| ▤ **Control Node** |
|   ▭ _Hexaidabs |
|   ▭ Name |
|   ▭ IsLeaf |
|   ▭ Comment |
| ⛏ OutgoingControlFlow |

| HOPEX Concepts |
|---|
| ⊐ OutgoingObjectFlow |
| ⊐ IncomingControlFlow |
| ⊐ IncomingObjectFlow |
| ⊐ OwnerGroup |
| ▤ **Input Pin** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ IsLeaf |
| ▭ ControlType |
| ▭ OrderingKind |
| ▭ Comment |
| ⊐ OutgoingControlFlow |
| ⊐ OutgoingObjectFlow |
| ⊐ IncomingControlFlow |
| ⊐ IncomingObjectFlow |
| ▤ **Output Pin** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ IsLeaf |
| ▭ ControlType |
| ▭ OrderingKind |
| ▭ Comment |

| HOPEX Concepts |
|---|
| ⊐ OutgoingControlFlow |
| ⊐ OutgoingObjectFlow |
| ⊐ IncomingControlFlow |
| ⊐ IncomingObjectFlow |
| ▤ **Exchange Pin** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ IsLeaf |
| ▭ ControlType |
| ▭ OrderingKind |
| ▭ Comment |
| ⊐ OutgoingControlFlow |
| ⊐ OutgoingObjectFlow |
| ⊐ IncomingControlFlow |
| ⊐ IncomingObjectFlow |
| ⊐ OwnerGroup |
| ▤ **Central Buffer Node** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ IsLeaf |
| ▭ ControlType |
| ▭ OrderingKind |

| HOPEX Concepts |
|---|
| ☐ Comment |
| ⛩ OutgoingObjectFlow |
| ⛩ IncomingObjectFlow |
| ⛩ OwnerGroup |
| ⊟ **Data Store Node** |
| ☐ _Hexaidabs |
| ☐ Name |
| ☐ IsLeaf |
| ☐ ControlType |
| ☐ OrderingKind |
| ☐ Comment |
| ⛩ OutgoingObjectFlow |
| ⛩ IncomingObjectFlow |
| ⛩ OwnerGroup |
| ⊟ **Activity Partition** |
| ☐ _Hexaidabs |
| ☐ Name |
| ☐ IsDimension |
| ☐ IsExternal |
| ☐ Comment |
| ⛩ ContainedElement |
| ⊟ **Object Flow** |

| HOPEX Concepts |
| --- |
| ⬜ _Hexaidabs |
| ⬜ Name |
| ⬜ IsLeaf |
| ⬜ Comment |
| ⌗ Guard |
| ⌗ Weight |
| ⌗ SourceElement |
| ⌗ TargetElement |
| ⬛ **Control Flow** |
| ⬜ _Hexaidabs |
| ⬜ Name |
| ⬜ IsLeaf |
| ⬜ Comment |
| ⌗ Guard |
| ⌗ Weight |
| ⌗ SourceElement |
| ⌗ TargetElement |
| ⬛ **Exception Handler** |
| ⬜ _Hexaidabs |
| ⬜ Name |
| ⬜ Comment |
| ⌗ ProtectedNode |

| HOPEX Concepts |
| --- |
| ⛁ ExceptionInput |
| ▦ **Activity UML** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Reentrant |
| ▭ SingleExecution |
| ▭ IsLeaf |
| ▭ Comment |
| ⛁ ElementOwnedByAnActivityUML |
| ▦ **Activity Parameter Node** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ IsLeaf |
| ▭ ControlType |
| ▭ OrderingKind |
| ▭ Comment |
| ⛁ OutgoingObjectFlow |
| ⛁ IncomingObjectFlow |
| ▦ **Structured Activity Node** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ IsLeaf |

| HOPEX Concepts |
|---|
| ▭ MustIsolate |
| ▭ Comment |
| ⌑ OutgoingControlFlow |
| ⌑ OutgoingObjectFlow |
| ⌑ IncomingControlFlow |
| ⌑ IncomingObjectFlow |
| ⌑ OwnerGroup |
| ⌑ ContainedElement |
| **▣ Expansion Node** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ IsLeaf |
| ▭ ControlType |
| ▭ OrderingKind |
| ▭ Comment |
| ⌑ OutgoingControlFlow |
| ⌑ OutgoingObjectFlow |
| ⌑ IncomingControlFlow |
| ⌑ IncomingObjectFlow |
| ⌑ Region |
| **▣ Expansion Region** |
| ▭ _Hexaidabs |

| HOPEX Concepts |
|---|
| ▭ Name |
| ▭ IsLeaf |
| ▭ MustIsolate |
| ▭ ExpansionKind |
| ▭ Comment |
| ⌦ OutgoingControlFlow |
| ⌦ OutgoingObjectFlow |
| ⌦ IncomingControlFlow |
| ⌦ IncomingObjectFlow |
| ⌦ OwnerGroup |
| ⌦ ContainedElement |
| ⌦ ExpansionNode |
| ▣ **Interruptible Activity Region** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⌦ ContainedElement |
| ▣ **Collaboration uml** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ▭ IsAbstract |

| HOPEX Concepts |
| --- |
| ▭ IsLeaf<br><br>⊐ CollaborationRole<br><br>⊐ OwnedConnector<br><br>⊐ OwnedCollaborationUse |
| ⊟ **Collaboration use**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ Comment<br><br>⊐ Type<br><br>⊐ Dependency |
| ⊟ **Part**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ Visibility<br><br>▭ Client Dependency<br><br>▭ IsUnique<br><br>▭ IsOrdered<br><br>▭ Multiplicity<br><br>▭ Comment<br><br>⊐ ConnectorEnd (of the LifeLine who represents the part)<br><br>⊐ Dependency |
| ⊟ **Connector** |

| HOPEX Concepts |
|---|
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Connector Kind |
| ▭ IsLeaf |
| ▭ Comment |
| ⛶ OwnedConnectorEnd |
| **▤ LifeLine** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⛶ ElementRepresentedByALifeline |
| ⛶ ElementCoveringLifeline |
| ⛶ OwnedSelector |
| **▤ Combined Fragment** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ InteractionOperatorKind |
| ▭ Comment |
| ⛶ InteractionOperand |
| ⛶ CoveredLifeLine |
| **▤ Interaction Use** |
| ▭ _Hexaidabs |

| HOPEX Concepts |
| --- |
| ▭ Name |
| ▭ InteractionOperatorKind |
| ▭ Comment |
| ⇄ RefersTo |
| ⇄ CoveredLifeLine |
| ▤ **State Invariant** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⇄ InvariantConstraint |
| ⇄ CoveredLifeLine |
| ▤ **Gate** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ▤ **Message UML** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ MessageKind |
| ▭ Comment |
| ⇄ Receiver |
| ⇄ Sender |

| HOPEX Concepts |
| --- |
| ⊐ Connector |
| ⊟ **Artifact UML**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ Comment<br><br>▭ Client Dependency<br><br>⊐ Target Dependency |
| ⊟ **Node UML**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ Comment<br><br>▭ Client Dependency<br><br>⊐ Deployment |
| ⊟ **Device**<br><br>▭ _Hexaidabs<br><br>▭ Name<br><br>▭ Comment<br><br>▭ Client Dependency<br><br>⊐ Deployment |
| ⊟ **Execution Environment**<br><br>▭ _Hexaidabs<br><br>▭ Name |

| HOPEX Concepts |
|---|
| ☐ Comment |
| ☐ Client Dependency |
| ⛄ Deployment |
| ☐ **Deployment Specification** |
| ☐ _Hexaidabs |
| ☐ Name |
| ☐ Comment |
| ☐ Client Dependency |
| ⛄ Target Dependency |
| ☐ **Instance ( Node UML/Device/Execution Specification/Component)** |
| ☐ _Hexaidabs |
| ☐ Name |
| ☐ Comment |
| ☐ Client Dependency |
| ⛄ Instantiated Element |
| ⛄ Target Dependency |
| ☐ **Communication Path** |
| ☐ _Hexaidabs |
| ☐ Name |
| ☐ Comment |
| ⛄ Communication Path End |
| ⛄ Target Dependency |

| HOPEX Concepts |
|---|
| ⊟ **Communication Path** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⊐ Multiplicity |
| ⊐ Deployment Target |
| ⊟ **Deployment** |
| ▭ _Hexaidabs |
| ▭ Name |
| ▭ Comment |
| ⊐ Multiplicity |
| ⊐ Deployed Element |
| ⊐ Deployment Configuration |